Computational Methods for Bayesian Inference in Macroeconomic Models

Computational Methods for Bayesian Inference in Macroeconomic Models

Ingvar Strid

Dissertation for the Degree of Doctor of Philosophy, Ph.D.
Stockholm School of Economics 2010.

Computational Methods for Bayesian Inference in Macroeconomic Models

ISBN 978-91-7258-839-4

Keywords:
Prefetching; Metropolis-Hastings; Parallel Computing; DSGE model; Optimal acceptance rate; Markov Chain Monte Carlo (MCMC); Kalman filter; Bayesian estimation; Parallel Particle Filter; Adaptive Metropolis-Hastings; Fortran; Matlab.

Printed in Sweden by:
Intellecta Infolog, Göteborg 2010

Distributed by:
The Research Secretariat
Stockholm School of Economics
P O Box 6501, SE 113 83 Stockholm, Sweden
www.hhs.se

To my Family

"The emergence in the past years of Bayesian analysis in many methodological and applied fields as the solution to the modeling of complex problems cannot be dissociated from major changes in its computational implementation."

C. Andrieu, A. Doucet and C.P Robert

Preface

This report is a result of a research project carried out at the Department of Economic Statistics and Decision Support, at the Stockholm School of Economics.

This volume is submitted as a doctor's thesis at SSE. The author has been entirely free to conduct and present his research in his own ways as an expression of his own ideas.

SSE is grateful for the financial support which has made it possible to fulfill this project.

Filip Wijkström
Associate Professor
SSE Director of Research

Contents

Acknowledgements

I thank my supervisor Sune Karlsson for crucial comments and advice which helped improve the papers in this thesis. Among other things, Sune also taught the excellent course in Bayesian Econometrics which was my first encounter with the subject.

I thank my co-authors Paolo Giordani, Robert Kohn and Karl Walentin for fruitful collaboration. Karl, who is a good friend since many years back, as well as a skilled economist, also provided valuable comments on a large part of the remaining thesis. I met Paolo at Sveriges Riksbank, where I worked as an economist, at the research unit, in 2008-2009. Our conversations on Bayesian methods are always interesting and instructive.

The collaboration with researchers at Sveriges Riksbank made it possible to use the RAMSES model as an example in two of the papers. I thank the developers of the model: Malin Adolfsson, Stefan Laséen, Jesper Lindé and Mattias Villani. In particular, Mattias has been helpful. I also appreciated our discussions on statistics and computers.

The following people provided helpful comments, raised important questions and/or discussed various aspects of this thesis: Gianni Amisano, Martin Andreasen, Miodrag Bolic, Jesus Fernandez-Villaverde, John Geweke, Marco Ratto, Nils Smeds, Ingelin Steinsland, Darren Wilkinson and Paolo Zagaglia. Paolo, a dedicated researcher, has been a co-author on two papers, not included in the thesis, and I have learned a lot from our numerous discussions over the years. I also thank the editors and anonymous referees at the relevant journals for their comments.

Håkan Lyckeborg, the teaching maestro at the Department of Statistics and Decision Support at SSE, has been a constant support throughout the years. I have appreciated the many conversations by the coffee table. Importantly, Håkan called me in 2002 and informed me about the vacant PhD student position at the department.

I thank former and current colleagues at the department for contributing to a stimulating environment, both socially and research-wise: Cristina Amado, Mickael Bäckman, Marcelo Carvalho, Bruno Eklund, Jana Eklund, Andrés González, Changli He, Hans Malmsten, Mika Meitz, Ganeshkumar Munnorcode, Tomoaki Nakatami, Rickard Sandberg, Birgit Strikholm, Ingolf Ståhl, Timo Teräsvirta, Pernilla Watson, and the late Zhengfang Zhao.

Ganesh has become a good friend during these years. I would like to express a special thanks for the help with editing the final document. I have enjoyed the social activities with Rickard, Tomoaki, Birgit, Mika, Andrés and Cristina. Timo and

Changli were vital for the research climate at the department and I benefited from their courses in time series analysis.

I would also like to thank the following people, who were previously at the Department of Economics, for making the years at SSE enjoyable: Max Elger, Martin Flodén, Henrik Lundvall, Erik Mohlin and Per Sonnerby.

The Stockholm School of Economics, through Anders Westlund, provided funding 2002-2007. The current head of the department, Jan Eklöf, arranged additional funding in the spring of 2010, such that I could finally complete the thesis. Tor Jacobsson offered me a job, at Sveriges Riksbank, precisely when I needed one. Tor also introduced me to the sometimes dirty, but fascinating, field of large microdatasets and SAS programming.

The Center for Parallel Computing (PDC) at the Royal Institute of Technology provided computing time for experiments conducted in two of the papers in the thesis, as well as a third, unfinished, paper.

Finally, I thank my family for the support. Moa, Axel and Anna. Aila, Bertil and Johanna. Moa's family.

Stockholm, October 2010
Ingvar Strid

Chapter 1

Introduction

The four papers in this dissertation deal with computational methods for Bayesian inference in dynamic stochastic general equilibrium (DSGE) models. The papers are located at the intersection of Bayesian econometrics, high performance (or parallel) computing and macroeconomics. The relevance of the papers is largely dependent on the continuation of either one, or both, of two major trends: one in economics and one in computing. First it is assumed that the New Macroeconometrics, described below, is here to stay, and that economists will continue to develop large, and complex, models of the macroeconomy in the future, and take these to the data. Second, it is assumed that, due to underlying technological trends in computing, parallel computation will become more widely applied by economists and econometricians, to solve computationally demanding problems.

In section 1.1 I describe the research area of New Macroeconometrics. In section 1.2 the mechanics of estimating DSGE models is briefly reviewed. In section 1.3 I provide a few remarks on the use of parallel computation methods. Finally, in section 1.4, the papers in the thesis are surveyed.

1.1 The New Macroeconometrics

The main macroeconomic model, in use at Sveriges Riksbank since 2006 is called RAMSES, the Riksbank Aggregate Macromodel for Studies of the Economy of Sweden (Adolfson, Laséen, Lindé and Villani (2007)). The model is used to forecast key macroeconomic variables, e.g. inflation and output, and to interpret current and recent economic developments. For example, the low inflation in Sweden in the years 2003-2006 may, at least at a first glance, suggest that monetary policy was too contractionary in this period. An analysis using the model, however, attributes the low inflation to favourable productivity shocks, and furthermore, it suggests that monetary policy was, in fact, unusually expansionary during the period. RAMSES is also used to construct scenarios for the future path of the economy and for policy analysis, activities which are interrelated. This model-based monetary policy analysis, combined with other analyses and judgement, informs the decision on the appropriate level of the Riksbank's key policy variable, the repo rate. (Adolfson, Laséen, Lindé and Villani (2007*b*))

The development of the RAMSES model, and its estimation, is a prominent example from the research area called *New Macroeconometrics*. The name was suggested by Jesús Fernández-Villaverde, in a fairly recent, and enthusiastic, survey of the area, titled *The Econometrics of DSGE Models* (Fernández-Villaverde (2009); see also Fernández-Villaverde, Guerron-Quintana and Rubio-Ramírez (2009)). The field has two key characteristics. First, modelling of the macroeconomy is done using DSGE models. Second, formal econometric methods are applied for inference in these models. Furthermore, the empirical analysis of DSGE models mainly follows a Bayesian approach.

A DSGE model is populated by agents, such as consumers, firms, and the government. These agents solve dynamic optimisation problems, e.g. firms maximise the present discounted value of the stream of future profits. Their optimal decisions

respond to various stochastic shocks, e.g. shocks to productivity and preferences, which act as the drivers of the model. The optimising behaviour of agents is one aspect of their rationality. Another aspect concerns the formation of expectations. The agents are typically assumed to make forecasts of the future, which are consistent with the model itself, i.e. what is called 'rational expectations'. In equilibrium, all markets in the model, e.g. goods and labour markets, simultaneously clear.

In a DSGE model there are a number of functions, e.g. a utility function, production functions, and various adjustment cost functions, which are indexed by parameters. An attractive feature of DSGE models is that these parameters often have a clear interpretation to the economist. The utility function of the household, for example, typically contains a discount factor, and a parameter describing risk aversion. The discount factor tells us how the households in the model value consumption today, versus consumption in the future. The risk aversion parameter informs us about their risk attitude.

Quantitative macroeconomics, following Kydland and Prescott, studies and evaluates DSGE models largely through the 'calibration approach' (Kydland and Prescott (1996)). First, values are chosen for the parameters in the model; this is the calibration. Next, paths for the variables in the model are simulated, and, finally, the properties of the realisations of the model economy are compared to the properties of real world data. The calibration can, for example, be based on evidence from microeconometric studies. Alternatively, it can be based on a comparison of some features of the data, e.g. first moments, with those implied by the model. For example, the household's discount factor is related to the steady state values of GDP growth and the real interest rate, such that sample averages of GDP growth, inflation and the nominal interest rate can be used to pin down a value for this parameter.

The success of a model is then judged by its ability to replicate a limited number of features of the data, i.e. those features which the model was constructed to explain. In the words of Kydland and Prescott, the model should 'mimic the world along a carefully specified set of dimensions.' In an attempt to formalise different approaches to DSGE model evaluation, John Geweke calls this view 'the weak econometric interpretation' of DSGE models. (Geweke (2007))

The strong econometric interpretation of a DSGE model, which provides the foundation for the New Macroeconometrics, says that the model provides an ex ante predictive distribution, $p(y|\theta)$, for some sequence of observables, y, where θ contains the parameters of the model. After observing y, the ex post likelihood function is $L(\theta; y)$. In this perspective, the model should mimic the world along all the dimensions, which are implied by the selection of y. For example, in the case of the RAMSES model, the data, y, consists of 15 key macroeconomic time series for the Swedish economy, and θ is a vector containing some 60 parameters.

Arguably, the weak interpretation, and the accompanying calibration approach, has been popular among macroeconomists since the early 1980s. Even today it is the dominant method for taking models to the data. In the last decade, however, likelihood-based inference for DSGE models has grown in popularity. Fernández-Villaverde and his co-authors provide several explanations for this trend. First, there have been a number of advances of a 'technical' character, i.e. the emergence of fast

and accurate solutions methods for DSGE models and the introduction of filtering methods in economics, which provide the necessary tools to perform the evaluation of the likelihood function. Second, the recent generation of DSGE models incorporate a number of features, such as various real and nominal rigidities, as well as a larger set of shocks, which give the models a better chance of fitting the data reasonably well (Christiano, Eichenbaum and Evans (2005)).

Finally, most of the recent empirical work within the New Macroeconometrics use Bayesian methods, rather than maximum likelihood. The information in the observed variables, which is contained in the likelihood function, $L(\theta; y)$, is combined with prior information on the parameters of the model, $p(\theta)$, and using Bayes' theorem,

$$p(\theta|y) = \frac{L(\theta; y)p(\theta)}{\int L(\theta; y)p(\theta)d\theta},$$

the posterior distribution for the parameters, $p(\theta|y)$, is obtained. In this way the information in the data and the prior information is combined in an optimal way. The prior information could, again, be based on evidence from microeconometric studies, or one could use information from other DSGE model estimations. For example, in estimating a DSGE model for the Swedish economy, prior information on the parameters could be based on estimates using US data for a similar model.

Fernández-Villaverde offers a comprehensive list of arguments for the adaption of a Bayesian approach in estimating DSGE models, when compared to other approaches, such as limited information approaches and maximum likelihood. Here I will merely consider a pragmatic view. Listening to econometricians discussing the likelihood of DSGE models, it may sometimes sound like a group of poets, struggling to provide the most colorful description of the Himalaya chain. The likelihood is of high dimension, with several local maxima and minima, flat surfaces, cliffs and ridges of various shapes. It follows that it is a hard problem to maximise the likelihood function of a DSGE model. Furthermore, even if we manage to locate Mount Everest in this rough terrain, perhaps using simulated annealing or some other advanced optimisation technique, it is not uncommon to see maximum likelihood estimates at the boundary of the parameter space, or estimates that simply do not make sense, from the viewpoint of economics.

Incorporating (informative) prior information typically reduces these problems, e.g. since it may add information in those dimensions where the data does not speak. It also appears quite natural for an economist, already familiar with the strict priors of the calibration approach.

1.2 Bayesian estimation of DSGE models

The Bayesian estimation of a DSGE model consists of three parts. First, an approximation to the policy function of the model is obtained (the solution). The policy function describes how the economic variables in the model evolve over time, and only in rare cases can it be obtained analytically. The approximate policy function

is then cast as the state equation in a state space model. The state space model is completed by specifying an observation equation. This equation relates a set of observed variables to the variables of the economic model, which are contained in the state vector. The state vector consists of a mix of variables which have a direct empirical counterpart, e.g. inflation, and variables which are genuinely unobservable, e.g. the shocks. Second, filtering methods are applied for the evaluation of the likelihood function for the parameters, $L(\theta; y)$, of the DSGE model. Third, the Metropolis-Hastings sampler is employed to sample from the posterior density, $p(\theta|y)$, for the parameters. Two of the papers in this thesis deal with filtering and the likelihood evaluation, and the other two center on the Metropolis-Hastings algorithm.

Most commonly, the policy function is approximated to first order. Further, assuming that the shock innovations are normally distributed, as is usually done, the resulting state space model is linear and Gaussian, and the Kalman filter can be applied to evaluate the likelihood function, for a given vector of parameters. The Kalman filter (or smoother) recursively calculates the distribution of the state vector, given the available data up to some point in time. For example, the filter can provide us with the probability distribution for the output gap, in the first quarter of 2011, given data up until the fourth quarter of 2010. This is an example of a one-step ahead predictive density for the output gap, in a model of quarterly data frequency. In computing this distribution we use the Q4 2010 distribution for the state vector, given data up to Q4 2010, which is the filtering state density for the preceding quarter, which in turn is based on the predictive Q4 2010 distribution, based on data until Q3 2010 (this is the meaning of 'recursive'). For the linear and Gaussian state space (LGSS) model all the distributions of the states, given the data, are Gaussian. The Kalman filter recursively calculates the means and covariances of these distributions, since the normal distribution is completely characterised by these two quantities.

We note that the LGSS model, which arises from the DSGE model, is a particular kind of LGSS model. The coefficients of the DSGE-LGSS model are highly nonlinear functions of the underlying parameters, θ, of the economic model. The mapping from the parameters, θ, of the DSGE model, to the coefficients of the LGSS model, is provided by the solution algorithm.

If the policy function approximation is non-linear, the resulting state space model is non-linear and/or non-Gaussian, and the Kalman filter recursions do not apply. Instead particle filtering methods can be used. The idea of particle filters is to approximate the relevant densities in the state space model, e.g. the one-step ahead predictive densities mentioned above, using discrete, weighted, samples. The approximations are obtained by sequential application of the importance sampling technique. Again, the likelihood can be obtained as a by-product of filtering. In the particle filtering case, however, we obtain an (unbiased) estimate of the likelihood, rather than an exact value. The consequences for sampling, i.e. of using an estimate of the likelihood instead of an exact value, have only recently begun to be investigated, see Flury and Shephard (2008).

Finally, the objective of sampling algorithms is to generate a sequence of draws, θ_i, $i = 1, \ldots, R$, from the posterior density, $p(\theta|y)$. In the context of DSGE model estimation, almost everyone have been using the single block random walk Metropolis-

Hastings (RWM) algorithm to sample from the posterior. In the RWM algorithm a proposal, θ_p, is generated using a symmetric proposal density, $q(.|\theta_i)$, where θ_i is the current state of the chain. Typically, the proposal density is a normal distribution, $q(.|\theta_i) = N(\theta_i, \Sigma)$. The proposal, θ_p, is accepted with probability

$$\alpha_{i+1} = \min\left\{1, \frac{p(\theta_p|y)}{p(\theta_i|y)}\right\}.$$

If the proposal is accepted we let $\theta_{i+1} = \theta_p$, and otherwise $\theta_{i+1} = \theta_i$. The other main type of Metropolis-Hastings algorithm is the independence chain variant, where the proposal is generated from $q(.)$. In this case the acceptance probability is given by

$$\alpha_{i+1} = \min\left\{1, \frac{p(\theta_p|y)/q(\theta_p)}{p(\theta_i|y)/q(\theta_i)}\right\}.$$

1.3 Parallel computing

Eight years ago, when I began my PhD studies, a standard desktop computer was equipped with a single processor. Today, it is quite common to have a computer with a dual- or quad-core processor, or even two quad-core processors. In other words, today the computers at our desks are parallel computers, and the increase in the number of processors and cores is likely to continue. This development reflects underlying technological trends, and mainly the difficulty of further increasing clock speed or transistor counts, in order to increase the performance of the processor. As a consequence, the dramatic rate of improvement in processor speed, seen in the last decades, is slowing (Creel and Goffe (2008)). In order to perform a given set of computations, e.g. to estimate a non-linearly approximated DSGE model using the particle filter, significantly faster in the future, it will therefore be necessary to utilise the power of several processors, or processor cores.

It is thus quite obvious that everyone will use multi-processor computers in the future. It is harder to predict to what extent economists and econometricians will apply parallel programming techniques to solve their computational problems. For example, on my quad-core computer I sometimes run four instances of Matlab, with a common program to sample from the posterior density of a model, while using four different starting values for the sampler, one on each core. In this case there is no need for parallel programming, and performance is increased 'horizontally', i.e. the individual chains are not computed faster. There are numerous examples in economics and statistics, where the multi-core computer could be employed in a similar way. A reasonable conjecture is that the large majority of applications in these sciences will attempt to benefit from multiple cores in the described way, i.e. without recourse to parallel programming.

In this thesis I take the perspective that the Bayesian estimation of DSGE models presents us with a computational task of such a magnitude, that it can benefit from application of the tools of high performance computing, i.e. parallel algorithms and programs. Let me provide an example to motivate this perspective. In a very

recent paper, a DSGE model with 30 parameters is estimated nonlinearly using the particle filter, and $N = 10,000$ particles (Fernández-Villaverde, Guerron-Quintana and Rubio-Ramírez (2010)). The reported time of evaluating the likelihood function is 22 seconds, using a 'highly optimised' parallel program written in Fortran and OpenMP, and executed on a Dell server with $P = 8$ processors. Obtaining $R = 5,000$ draws from the posterior, using the RWM algorithm, takes 38 hours. For comparison, in the original estimation of the linearised RAMSES model, 500,000 draws from the posterior were obtained. In the non-linear estimation case, this number of draws would yield a computing time of 100×38 hours, which equals roughly five months of computing time. Furthermore, one may add that the first attempt at sampling from the posterior is not always successful.

Three of the articles in the thesis are concerned with the development of parallel algorithms. One of the algorithms performs the particle filter likelihood evaluation faster, and the two other performs sampling from the posterior faster. These algorithms are, with one exception, communication intensive, i.e. they require frequent communication between processors, and the communication pattern must be explicitly coded. The algorithms are expressed using the, fairly intuitive, language of message passing, and the communication between processors is implemented using the Message Passing Interface (MPI). The communication intensive nature of the algorithms implies that a cluster, with a reasonably fast interconnection network, is a natural hardware environment. However, two examples of execution on multi-core computers are also provided.

1.4 The papers in the thesis

The evaluation of the likelihood function, using the Kalman filter, accounts for the major part of computational time, in estimating large (log-)linearised DSGE models. Furthermore, a large fraction of Kalman filtering time is spent on the prediction step of the filter, and a matrix multiplication of the type TPT', where P is a covariance matrix, and T is the state transition matrix in the LGSS model.

The basic idea in the paper *Block Kalman filtering for large-scale DSGE models*, which is written together with Karl Walentin, is to exploit the structure of the DSGE-specific LGSS model, and primarily the block structure of the matrix T, in order to perform the filter recursions fast. The first part of the paper consists of describing a common structure of DSGE models. In the remaining part of the paper the implied Kalman filter recursions are written down, such that the efficiency of the computations is maximised. In the computational experiment various filter implementations are tested on three DSGE models. The block filter, when applied to the large-scale RAMSES model, roughly halves computational time, in comparison with a standard Kalman filter implementation.

The evaluation of the likelihood function, using the particle filter, is computationally very burdensome, as we saw in the previous section. To illustrate this further, I estimate a small DSGE model linearly, using the Kalman filter, and nonlinearly (based on a second order Taylor approximation of the model's policy function), using

the particle filter. The speed difference between the linear and nonlinear estimation routines is a factor larger than $1,000$, and more than 98% of the nonlinear estimation is spent on the particle filter likelihood evaluation.

The paper *Parallel particle filters for likelihood evaluation in DSGE models* may serve as an introduction to parallel particle filtering methods, applied in the context of DSGE model likelihood evaluation, and it is intended for an audience of economists/econometricians. While the Kalman filter provides an exact value of the likelihood, for a given vector of parameters, the particle filter provides an estimate of the likelihood. The likelihood estimate error variance is governed by the number of particles used for the particle filter density approximations. Parallelisation of the particle filter is a brute force approach, which can serve to reduce the time of the likelihood evaluation and/or to increase the accuracy of the likelihood estimate.

The computational experiments in the paper illustrate the absolute gains that can be achieved by applying parallel computation methods to particle filtering. Two parallel particle filter (PPF) algorithms are compared: a 'simple' algorithm, and a 'complex', where the latter algorithm reduces inter-processor communication in comparison with the former. The algorithms are implemented using Fortran and MPI and are tested on three computing clusters. The execution times suggest that the simpler algorithm is a good choice, in estimating small DSGE models. An advantage of the complex algorithm, however, is that the execution time does not depend on the statistical performance of the particle filter, e.g. as measured by the effective sample size, a common metric to assess the performance of the filter. Relating to the paper mentioned in the previous section (Fernández-Villaverde et al. (2010)), I am fairly certain that these PPF algorithms could perform the likelihood evaluation in their example in less than one second, without large sacrifices in parallel efficiency, provided the appropriate hardware is used.

The paper makes some more general contributions to the parallel particle filtering literature. First, a detailed treatment of message passing inter-processor communication in the PPF algorithm is provided. Second, a parallel residual systematic resampling algorithm is presented. Third, our experiments indicate that the bottleneck of the PPF algorithm is the collective communication, rather than the transfer of particles between processors associated with resampling.

The next paper has its origin in some back-of-the-envelope calculations, which are reproduced here. Applying the PPF algorithm within the RWM sampler, one may ask what an 'acceptable' level of efficiency for the former algorithm might be. Assume that the PPF executes in 1.5 seconds, 0.056 s, and 0.05 s, using 1, 32 and 64 processors, respectively (these numbers are taken from the paper just discussed, for a particular setup). The relative speedups, with 32 and 64 processors, are $S(32) = T(1)/T(32) = 1.5/0.056 = 26.8$ and $1.5/0.05 = 30$. Clearly, the marginal contribution to performance, from the last 32 processors, is low. Should we use all the 64 processors for a single likelihood evaluation in this case? An answer to this question can be obtained by considering an alternative parallelisation strategy.

The targeted acceptance rate of the RWM sampler, which was briefly described above, is typically chosen to be close to $\alpha = 0.25$, such that every fourth proposal is accepted, on average. This choice is motivated by the optimal acceptance rate

result derived by Roberts, Gelman and Gilks (1997). Now, consider making two simultaneous proposals using the proposal density, $q(.|\theta_i)$, and evaluating the posterior density for these proposals, in parallel, on two processors. If the first proposal is rejected, which occurs with probability $1 - \alpha = 0.75$, the evaluation of the second proposal becomes useful, since the state of the Markov chain has not changed (we are at $\theta_{i+1} = \theta_i$). In this case two draws from the sampler are obtained, in one unit of time. If, on the other hand, the first proposal is accepted, the evaluation of the second proposal becomes useless, and one draw from the sampler is obtained, in one unit of time. The expected relative speedup, which equals the expected number of draws generated by the two proposals, is equal to $0.75 \times 2 + 0.25 \times 1 = 1.75$. Combining the speedup numbers, one concludes that it is more efficient to perform two PPF likelihood evaluations, in parallel, using two groups with 32 processors each, than to apply all the 64 processors for a single PPF likelihood evaluation. (The result here is thus a parallel-within-parallel algorithm.)

The approach of evaluating several RWM proposals in parallel, and 'ahead of time', has been given the name prefetching, by Brockwell (2006). In the paper *Efficient parallelisation of Metropolis-Hastings algorithms using a prefetching approach* the prefetching approach is generalised and improved. The key idea of prefetching is to predict, or anticipate, where the Markov chain is moving in the near future. Above we used the targeted acceptance rate of the sampler, α, to make the prediction. Since the probability of a reject was higher, we employed the second processor to prepare for this outcome.

There are two main approaches to prefetching presented in the paper. Optimal static prefetching assumes that proposals will always be rejected, and therefore evaluates P proposals, which are all based on the current, and therefore the same, state of the chain, θ_i, where P is the number of processors applied with the algorithm. It is shown that the optimal acceptance rate decreases in P. I define the *optimal acceptance rate in a parallel setting* to be the acceptance rate which minimises the time of obtaining a sample of a fixed size and quality, e.g. a fixed number of independent draws, from the posterior, conditional on the number of processors, P, used. In optimal static prefetching we therefore choose the target acceptance rate, based on P. For example, with $P = 40$ processors the optimal acceptance rate is roughly 0.04, which is far below the optimal acceptance rate of 0.234 on a single processor.

The 'most likely path' approach to prefetching uses an approximation, $\hat{p}$, to the posterior in order to deliver the predictions. The RWM algorithm is run for P steps, using $\hat{p}$ instead of the posterior p, and the P proposed parameters are recorded. In a second stage the posterior is evaluated, in parallel, for these P proposals. A key here is that the evaluation of $\hat{p}$, e.g. a normal density, is much faster than evaluation of p. The success of the approach depends on the quality of the approximation, $\hat{p}$. A funny thing to note about this algorithm though, is that close to linear speedup, $S(P) \approx P$, is no cause for celebration. It simply means that $\hat{p}$ is a very good approximation to p, which implies that other sampling approaches, e.g. independence chain Metropolis-Hastings (IMH) or importance sampling, should be more efficient than the RWM algorithm.

The RWM algorithm is a sequential algorithm, since the proposal, θ_p, depends

on the current state of the chain, θ_i. The IMH sampler, on the other hand, is non-sequential, since the proposals are generated independently, using $q(.)$. This implies that a very simple and efficient parallel IMH algorithm can be constructed. The algorithm is tested using a DSGE model which is quite similar to the model presented by Smets and Wouters (2003), the older, but smaller, cousin of the RAMSES model. Estimating the 23 parameters of the linearised DSGE model with 15 state variables, using the parallel IMH algorithm with 64 processors, the time of obtaining one million draws from the posterior is roughly three minutes, and the relative speedup is 63.

A comparison of the RWM prefetching algorithm and the parallel IMH algorithm illustrates a more general aspect of parallel Markov Chain Monte Carlo (MCMC) algorithms. In the words of Darren Wilkinson, in his handbook chapter on parallel MCMC, 'the trade-offs one would adopt for the design of an efficient MCMC algorithm ... in a nonparallel environment may well be different in a parallel setting' (Wilkinson (2006)). Let us try to illustrate this statement, using the two parallel Metropolis-Hastings algorithms.

The Bayesian econometrician is assumed to be interested in maximising the number of independent equivalent draws per time unit, $Y(P)$, using P processors, and we let A_S be the number of independent equivalent draws per time unit for a serial sampling algorithm, S. For the parallel IMH algorithm, we have the 'constant returns to scale production function'

$$Y_{IMH}(P) \approx A_{IMH}P,$$

whereas for the most likely path RWM prefetching algorithm it can be shown that

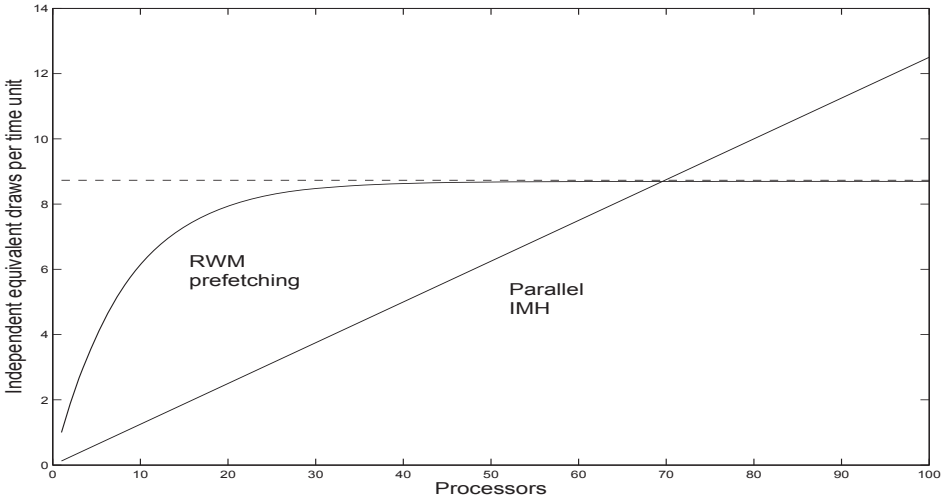
$$Y_{RWM}(P) = A_{RWM}\frac{1-(1-\tilde{\alpha})^P}{\tilde{\alpha}},$$

where $\tilde{\alpha}$ is a 'failure probability'. The interpretation of $\tilde{\alpha}$ is the following: it is the probability of making the wrong accept/reject decision when the approximation, $\hat{p}$, is used in place of the posterior, p, in the RWM algorithm. In estimating the 15-state DSGE model referred to above, I find that $\tilde{\alpha} \approx 0.115$. Based on a comparison of the inefficiency factors produced by the RWM and IMH algorithms, I also find that $A_{RWM}/A_{IMH} \approx 8$, where the IMH algorithm uses a multivariate t distribution to generate the proposals (we should note that the performance of the IMH algorithm is unreliable in this example). In figure 1.1 the function $Y(P)$ is plotted, where the normalisation $A_{RWM} = 1$ is applied.

The main observation is that both algorithms have a region for the number of processors, P, where it dominates the other algorithm. For example, the RWM prefetching algorithm should be preferred when using $P < 70$ processors.

The final paper, *Adaptive hybrid Metropolis-Hastings algorithms for DSGE models*, is written together with Paolo Giordani and Robert Kohn. The purpose of our paper is to suggest an alternative to the widely used RWM algorithm, in the context of sampling from the posterior of DSGE models. The sampler of main interest in the paper has two components, i.e. it is a hybrid sampler. The main component is an adaptive independence proposal, which is chosen to be a t copula with mixture of

Figure 1.1 Independent equivalent draws per time unit, using the random walk prefetching and parallel independence chain samplers



normal marginal densities, q^*. This proposal density is re-estimated at various points in the chain, based on the previously obtained draws; hence the sampler is adaptive.

The success of an independence sampler depends on the ability of the proposal density, q, to approximate the posterior density, p. The key idea here is to use a flexible multivariate density, q^*, which will hopefully provide a good approximation to the posterior, as the number of draws from the sampler, used to estimate q^*, accumulates.

The problem with independence samplers, especially in high dimensions, is that they sometimes get trapped for very long spells at 'difficult points' in the parameter space, i.e. when the posterior-to-proposal ratio, $p(\theta_i)/q(\theta_i)$, is large. This motivates the usage of a second component, the RWM step, to escape such points. Intuitively, the main component is expected to perform well most of the time, while the RWM step is our insurance policy in case the sampler gets stuck.

An attractive feature of this algorithm is that it displays very good parallel efficiency properties, at least if the fraction of draws generated by the RWM component is kept low. An insurance always come with a cost, and the cost of the hybrid sampler is a decrease in parallel efficiency, in comparison with a pure independence sampler, such as the IMH sampler.

Applying the main sampler to estimate the 15-state DSGE model used above, we find that $A_{HA}/A_{RWM} \approx 7$, based on a comparison of inefficiency factors, where A_{HA} are the number of independent equivalent draws per time unit for the hybrid adaptive sampler. The overall fraction of RWM proposals used in this case is roughly 2%. A

simple theoretical analysis of the parallelisation properties of the algorithm yields the production function

$$Y_{HA}(P) \approx A_{HA} \frac{P}{0.98 + 0.02P}.$$

To summarise, in this example the suggested sampler is substantially more efficient than the RWM algorithm, based on the comparison of inefficiency factors, and furthermore, it is much better suited for parallelisation. In fact, in this particular example it is difficult to draw the production function, $Y_{HA}(P)$, in the same picture as the other two functions above.

Bibliography

Adolfson, M., Laséen, S., Lindé, J. and Villani, M. (2007*a*), 'Bayesian Estimation of an Open Economy DSGE Model with Incomplete Pass-Through', *Journal of International Economics* **72(2)**, 481–511.

Adolfson, M., Laséen, S., Lindé, J. and Villani, M. (2007*b*), 'Ramses: A new general equilibrium model for monetary policy analysis', *Sveriges Riksbank Economic Review* **2**, 5–40.

Brockwell, A. (2006), 'Parallel Markov Chain Monte Carlo Simulation by Pre-fetching', *Journal of Computational and Graphical Statistics* **15(1)**, 246–261.

Christiano, L. J., Eichenbaum, M. and Evans, C. (2005), 'Nominal rigidities and the dynamic effects of a shock to monetary policy', *Journal of political economy* **113(1)**, 1–45.

Creel, M. and Goffe, W. (2008), 'Multi-core CPUs, Clusters, and Grid Computing: A Tutorial', *Computational economics* **32(4)**, 353–382.

Fernández-Villaverde, J. (2009), 'The Econometrics of DSGE Models', *NBER Working Paper, No. w14677* .

Fernández-Villaverde, J., Guerron-Quintana, P. and Rubio-Ramírez, J. F. (2009), 'The New Macroeconometrics: A Bayesian Approach'.

Fernández-Villaverde, J., Guerron-Quintana, P. and Rubio-Ramírez, J. F. (2010), 'Fortune or Virtue: Time-Variant Volatilities Versus Parameter Drifting in US Data', *NBER Working Paper No. 15928* .

Flury, T. and Shephard, N. (2008), 'Bayesian Inference Based Only on Simulated Likelihood: Particle Filter Analysis of Dynamic Economic Models'. Economics working paper 413, Oxford University.

Geweke, J. (2007), 'Models, Computational Experiments and Reality'. University of Iowa, mimeo.

Kydland, F. and Prescott, E. (1996), 'The computational experiment: An econometric tool', *Journal of Economic Perspectives* **1**, 69–85.

Roberts, G., Gelman, A. and Gilks, W. (1997), 'Weak Convergence and Optimal Scaling of Random Walk Metropolis Algorithms', *The Annals of Applied Probability* **7(1)**, 110–120.

Smets, F. and Wouters, R. (2003), 'An Estimated Stochastic Dynamic General Equilibrium Model of the Euro Area', *Journal of the European Economic Association* **1(5)**, 1123–1175.

Wilkinson, D. (2006), Parallel Bayesian Computation, *in* E. J. Kontoghiorghes, ed., 'Handbook of Parallel Computing and Statistics', Chapman and Hall, chapter 16, pp. 477–508.

Chapter 2

Block Kalman filtering for large-scale DSGE models

Block Kalman filtering for large-scale DSGE models[1]

abstract

In this paper block Kalman filters for Dynamic Stochastic General Equilibrium models are presented and evaluated. Our approach is based on the simple idea of writing down the Kalman filter recursions on block form and appropriately sequencing the operations of the prediction step of the algorithm. It is argued that block filtering is the only viable serial algorithmic approach to significantly reduce Kalman filtering time in the context of large DSGE models. For the largest model we evaluate the block filter reduces the computation time by roughly a factor 2. Block filtering compares favourably with the more general method for faster Kalman filtering outlined by Koopman and Durbin (2000) and, furthermore, the two approaches are largely complementary.

[1]This paper is a joint work with Karl Walentin, Sveriges Riksbank. It has appeared in *Computational Economics* 33, 277-304, 2008.
We are grateful for helpful comments from Mattias Villani, Steve Lionel, Duncan Po and seminar participants at Sveriges Riksbank. The views expressed in this paper are solely the responsibility of the authors and should not be interpreted as reflecting the views of the Executive Board of Sveriges Riksbank.

2.1 Introduction

Dynamic Stochastic General Equilibrium (DSGE) models are nowadays routinely estimated using Bayesian methods. The size of some models, e.g. those developed at various central banks, is becoming larger and computational time is perceived as a concern [Adolfson, Lindé and Villani (2007*a*)], [Azzini, Girardi and Ratto (2007)], [Christoffel, Coenen and Warne (2007)]. Bayesian estimation of linearised DSGE models using the Metropolis-Hastings algorithm and the Kalman filter for likelihood evaluation typically require at least, say, 100.000 draws from the posterior. For larger models several days of computing time may be needed and most of this time is spent on Kalman filtering.

This motivates the development of efficient Kalman filter algorithms and implementations tailored to the particular linear and Gaussian state-space model (LGSS) associated with a wide-class of linearly approximated DSGE models. Faster Kalman filtering, if possible, increases the quality and/or reduces the time of the likelihood-based analysis of DSGE models.

The purpose of this paper is to present and evaluate block Kalman filters for DSGE models which exploit the symmetry of the filter and the particular DSGE model block structure and sparse structure of some system matrices. Our DSGE model specific approach, which is straightforward and mainly attempts to reduce the time of the Kalman filter prediction step, is integrated with and compared to the general strategy for faster Kalman filtering outlined by Koopman and Durbin (2000). Their approach focuses on the updating step of the filter, implying that the two approaches are largely complementary.[2]

The usefulness of the block Kalman filter approach is illustrated using three well-known macroeconomic models and particular interest is devoted to the large-scale open-economy model developed at Sveriges Riksbank [Adolfson, Laséen, Lindé and Villani (2007)]. The evaluation exercise clarifies the interrelationship between model size, block structure, algorithm, implementation language, compiler, matrix library and Kalman filtering time for these three representative models. This exercise, we believe, provides a useful guide to the researcher who wants to obtain maximal computational efficiency in estimating linearised DSGE models.

Our main results are, first, that block filtering is the only algorithmic approach that, in itself, can deliver a significantly lower Kalman filtering wall-clock time for the large-scale DSGE model. Second, quicker execution of the updating step can largely be achieved without resorting to the univariate filtering approach of Koopman and Durbin (2000). Third, for smaller models the choice of implementation language appears much more important than the choice of Kalman filter algorithm.

The practical perspective of the paper guides the choice of programming language for implementations of the filters. The filters are programmed in Matlab, the dominant language among economists, and as Fortran Mex functions to be called from Matlab. The standard implementation of the Kalman filter is presumably close to the ideal application for Matlab. However, Matlab is not the ideal language if

[2]The terms *updating* and *prediction* are explained in section 2.4 where the Kalman filter is presented.

one wants to maximise the performance of Kalman filtering for DSGE models. In contrast, a language like Fortran (and presumably C) appears better suited for the implementations suggested in this paper.

The Kalman filter implementations that come with the paper are easy to use. From a user's perspective the only requirement is that the state vector of the economic model is ordered in a particular way. The Kalman filter interface is uniform across algorithms/implementations such that details of algorithms can largely be hidden from the user.

The paper proceeds as follows. In section 2.2 the computational problem is briefly described. In section 2.3 it is shown how to cast the state-space model in the form required for block filtering. In section 2.4 the Kalman filter is presented in a form suitable for the presentation of the block filters in section 2.5. Three example models and the setup of the computational experiment are described in section 2.6, and in section 2.7 results from tests of the filters are presented.

2.2 The computational kernel

For large DSGE models Kalman filtering time is dominated by the matrix multiplication associated with the prediction step of the Kalman filter

$$P^* = TPT^T \tag{2.1}$$

where P is an $m \times m$ symmetric matrix and T is the $m \times m$ state transition matrix. As an example, the open-economy DSGE model presented in Adolfson, Laséen, Lindé and Villani (2007) contains $m = 65$ state variables. For this model, and an efficient Matlab implementation of the Kalman filter, on a standard desktop computer more than 60% of Kalman filtering time is spent on the above multiplication.

The univariate filtering approach [Koopman and Durbin (2000)] works on the updating step of the Kalman filter and is therefore, in itself, expected to provide only limited time gains when the state dimension is large relative to the observation dimension, which is often the case for DSGE models. However, the generality of the approach, its apparent success in significantly reducing Kalman filter time for many models and the unavailability of other approaches to faster Kalman filtering suggest it as a natural benchmark algorithm.

The block approach, on the other hand, aims mainly to reduce the time of the prediction step by exploiting the structure of the matrices T and P and therefore works complementary to univariate filtering. It should be noted that the matrix T is not sparse but that it has a specific block structure for DSGE models, e.g. for the open-economy model mentioned above 35% of the elements of T are nonzero.

The block filter performs the above multiplication on blocks of the matrices T and P. First, the symmetry of P (and P^*) is "manually" taken into account. Second, a number of multiplications involving zero submatrices are disposed of in the process.

2.3 State-space representation

A wide class of linearly approximated DSGE models can be cast in the general linear state space form

$$X_t = c + T(\theta) X_{t-1} + R(\theta) \epsilon_t \tag{2.2}$$

$$Y_t = d(\theta) + Z X_t + v_t \tag{2.3}$$

where [2.2] is the state transition equation and [2.3] is the observation equation. The state vector X_t has dimension m, the measurement vector Y_t has dimension N and the vector of fundamental innovations ϵ_t has dimension g. The dimensions of the matrices T, R and Z are $m \times m$, $m \times g$ and $N \times m$. The structural and auxiliary parameters of the DSGE model are collected in the vector θ. The distributions of the fundamental innovations ϵ_t and the measurement error v_t are $N(0, Q)$ and $N(0, H)$ respectively and ϵ_t and v_s are assumed to be independent.

If Kalman filtering time is a serious concern it can be expected that at least some effort has been made to reduce the dimension of the state vector, i.e. to remove any variable X_{jt} for which the j^{th} column of T and the j^{th} column of Z are zero vectors (static endogenous variables which are not observed). Keeping these variables in the system obviously is contradictory to fast filtering although it could be convenient for other purposes. Since this reduction of the state dimension is easily performed manually prior to estimation it will not be considered further here.

It will be assumed that the covariance matrices

$$P_{t|s} = E\left[\left(X_t - E\left[X_t|Y_{1:s}\right]\right)\left(X_t - E\left[X_t|Y_{1:s}\right]\right)^T |Y_{1:s}\right]$$

$s = t-1, t, t+1$,$t = 1, \ldots, T$, are not necessarily positive definite since many DSGE models are represented as state-space models with singular $P_{t|s}$.

The most general state-space model of interest to us here has four blocks. The state vector is partitioned as

$$X^T = \left[X_1^T \; X_2^T \; X_3^T \; X_4^T\right] \tag{2.4}$$

with dimensions $m_j, j = 1, 2, 3, 4$, such that $g = m_1 + m_2 + m_3$ and $m = g + m_4$.

The first block contains exogenous AR(1) processes and the second block contains exogenous VAR(1) processes. Exogenous variables which appear in the observation equation, i.e. variables for which the corresponding column in Z contain non-zero entries, are collected in the third block and endogenous variables appear in the fourth block. The block structure of a model is succinctly captured by the vector $\mathbf{m}_4 = (m_1, m_2, m_3, m_4)$. The four-block structure of the model is described by the matrices

$$T = \begin{bmatrix} A_1 & 0 & 0 & 0 \\ 0 & A_2 & 0 & 0 \\ 0 & 0 & A_3 & 0 \\ B_1 A_1 & B_2 A_2 & B_3 A_3 & C \end{bmatrix} \tag{2.5}$$

$$R = \begin{bmatrix} I_{m_1} & 0 & 0 \\ 0 & I_{m_2} & 0 \\ 0 & 0 & I_{m_3} \\ B_1 & B_2 & B_3 \end{bmatrix} \tag{2.6}$$

$$Q = \begin{bmatrix} Q_1 & 0 & 0 \\ 0 & Q_2 & 0 \\ 0 & 0 & Q_3 \end{bmatrix} \tag{2.7}$$

$$Z = \begin{bmatrix} 0 & 0 & Z_3 & Z_4 \end{bmatrix} \tag{2.8}$$

where the matrices A_1 and Q_1 are assumed to be diagonal, the matrices Q_2 and Q_3 are symmetric and I_{m_i} is the $m_i \times m_i$ identity matrix. In the block filters potential diagonality of H is not exploited, implying that the presence of correlated measurement errors does not present any additional issues.

The only requirement for application of the filters in this paper is that the LGSS model has the structure described by [2.5]-[2.8]. An unordered model, i.e. a model for which the state vector have not been cast in the form [2.4], described by the matrices $\tilde{T}, \tilde{R}, \tilde{Q}$ and $\tilde{Z}$ can be transformed to the required form by defining the $m \times m$ reordering matrix M.

Assume that the original, unordered, state vector is given by $\tilde{X}$. Let $M_{ij} = 1$ if the variable in place i in $\tilde{X}$ obtains the new position j in X and $M_{ij} = 0$ otherwise. The ordered model is then obtained utilizing the relationships

$$T = M^T \tilde{T} M \tag{2.9}$$

$$R = M^T \tilde{R} \tilde{M} \tag{2.10}$$

$$Q = \tilde{M}^T \tilde{Q} \tilde{M} \tag{2.11}$$

$$Z = \tilde{Z} M \tag{2.12}$$

where the $g \times g$ matrix $\tilde{M}$ is obtained as a submatrix of M through deletion of rows and columns associated with endogenous variables. It is thus easy to transform the state-space representation of a DSGE model to the required form.

Faster Kalman filtering in this context is essentially based on two approaches:

- Taking into account the symmetries in the filter, i.e. only calculating the upper (or lower) triangular part of covariance matrices $P_{t|s}$, $s = t-1, t$, $t = 1, \ldots, T$. This can be performed (i) using special matrix routines for symmetric matrices and/or (ii) manually, as in the block filter.

- Exploiting the DSGE model specific structure of T and Z. This goes beyond the symmetric filter and further motivates the block filters presented below.

Many DSGE models only consist of blocks 1 and 4. Therefore, in section 2.5.1, we first present the 2-block filter which is suitable for models which do not contain an exogenous VAR block. In section 2.5.2 the more general 4-block filter is presented. A 3-block filter, which is obtained by merging blocks 3 and 4 of the 4-block filter into one block is presented in the Appendix. A notable example of a DSGE model, in addition to [Adolfson, Laséen, Lindé and Villani (2007)], that consists of more than two blocks is the New Area-Wide Model (NAWM) developed at the ECB [Christoffel et al. (2007)].

Before moving on we briefly comment on two issues. First, the discussion is restricted to time-invariant (constant coefficient) models. However, it would be straightforward to extend the filters to the time-varying case. In fact, if time variation is restricted to certain blocks of a model this could be an additional argument in favour of block filtering but this issue is not discussed further here.

Second, if the covariance matrix P has converged to its steady state solution avoiding updating it saves a lot of computational effort. For reasons of simplicity and transparency it will be assumed in our experiments that P has not converged during the time span of the filter. This is normally the case for large DSGE models and the relevant sample lengths. For smaller models convergence is presumably more common and steady state Kalman filtering can sometimes save a lot of computational effort.

2.4 Kalman filter

2.4.1 Kalman filter equations

The Kalman filter equations are included to make the paper self-contained (see e.g. Harvey (1989) ch. 3). Let

$$X_{t|s} = E\left[X_t|Y_{1:s}\right]$$

denote the optimal estimator of the state vector X_t based on the observations up to time s

$$Y_{1:s} = (Y_1, Y_2, \ldots, Y_s)$$

and let

$$P_{t|s} = E\left[\left(X_t - E\left[X_t|Y_{1:s}\right]\right)\left(X_t - E\left[X_t|Y_{1:s}\right]\right)^T |Y_{1:s}\right]$$

The *updating* (or *contemporaneous filtering*) equations are

$$X_{t|t} = X_{t|t-1} + K_t F_t^{-1}\left(Y_t - ZX_{t|t-1} - d\right) \tag{2.13}$$

and

$$P_{t|t} = P_{t|t-1} - K_t F_t^{-1} K_t^T \tag{2.14}$$

where

$$F_t = V\left(Y_t|Y_{1:t-1}\right) = ZK_t + H \tag{2.15}$$

is assumed to be positive definite and

$$K_t = P_{t|t-1}Z^T \tag{2.16}$$

The *prediction* equations are

$$X_{t+1|t} = TX_{t|t} + c \tag{2.17}$$

and

$$P_{t+1|t} = TP_{t|t}T^T + RQR \tag{2.18}$$

As the state dimension of the LGSS model increases the matrix multiplication in [2.18] will dominate computational time, as previously discussed. For DSGE models the selector matrix Z is typically sparse, i.e. it contains relatively few non-zero entries, such that [2.15] and [2.16] are sparse-dense type of multiplications. This sparseness of Z can sometimes be exploited for more efficient computing.

2.4.2 Implementation issues

The usual (1-block) Kalman filter is implemented exactly as described above. In Matlab this can be done in essentially one way. Using Fortran the filter can be implemented using BLAS routines for symmetric matrices, e.g. *symm*, *syrk* and *syr2k*, calculating only the upper (or lower) part of the matrices $P_{t|t-1}$ and $P_{t|t}$. This will be referred to as a *symmetric filter implementation*. The alternative is to implement the filter in correspondence with its implementation in Matlab, i.e. using the standard matrix multiplication, *gemm*, everywhere, a *non-symmetric filter implementation*.

It should be noted that the block filter, to some extent, reduces the rationale for applying symmetric routines since the block filter approach manually deals with the symmetry of the covariance matrices $P_{t|s}$. The added value of the block filter is of course that, in addition, the computations associated with the zero blocks of the state transition matrix, T, need not be performed.

The perspective of the paper is to take the set of matrix operation routines available in BLAS, as implemented in MKL or IMSL (or similar and/or overlapping libraries) as given building blocks in constructing efficient Fortran implementations of the filters.[3] In other words there is no ambition to improve upon the design of the basic routines of these libraries.

Interestingly, the particular operation of main interest for Kalman filtering [2.1] has not yet, to our knowledge, been implemented as a separate routine in any of the mentioned libraries. Therefore we compare several ways of performing this operation based on routines in the matrix libraries, over a range of matrix sizes, in order to obtain a reasonably efficient implementation. Our symmetric filter implementations perform the type of multiplication in [2.1] via calls to *symm* and *syr2k*.

[3]The BLAS (Basic Linear Algebra Subprograms) are routines that provide standard building blocks for performing basic vector and matrix operations (see e.g. www.netlib.org). MKL (Intel Math Kernel Library) is a library of math routines and it includes BLAS routines optimised for Intel processors. IMSL (International Mathematics and Statistics Library) is a library of routines for numerical analysis, also containing the BLAS routines.

2.5 Block Kalman filter

2.5.1 2-block filter

In this section the Kalman filter is written down on 2-block form. The DSGE model is assumed to consist of m_1 exogenous AR(1) shocks (block 1) and m_2 endogenous variables (block 2). Many small and intermediate scale DSGE models have this structure. If the model contain exogenous VAR processes and/or AR/VAR variables which appear in the observation equation these are placed in the second block. In this case a simple adjustment to the filter is required (see below).

The model is thus assumed to be on the form

$$T = \begin{bmatrix} A & 0 \\ BA & C \end{bmatrix} \tag{2.19}$$

$$R = \begin{bmatrix} I_{m_1} \\ B \end{bmatrix} \tag{2.20}$$

$$Q = \begin{bmatrix} Q_1 & 0 \\ 0 & Q_2 \end{bmatrix} \tag{2.21}$$

$$Z = \begin{bmatrix} Z_1 & Z_2 \end{bmatrix} = \begin{bmatrix} 0 & Z_2 \end{bmatrix} \tag{2.22}$$

where A is diagonal.

The mean vectors $X_{1t|s}$ and $X_{2t|s}$ and the blocks $P_{11t|s}, P_{12t|s}$ and $P_{22t|s}$ of the covariance matrices

$$P_{t|s} = \begin{bmatrix} P_{11t|s} & P_{12t|s} \\ & P_{22t|s} \end{bmatrix}$$

are propagated instead of $X_{t|s}$ and $P_{t|s}$, $s = t-1, t$ and $t = 1, \ldots, T$.

Again, the purpose of the block filter approach is to (i) avoid calculation of $P_{21} = P_{12}^T$, (ii) to dispose of a set of unnecessary multiplications, i.e. those involving zero submatrices (here the upper-right submatrix of T), (iii) to exploit the special structure of the lower-left submatrix of T and (iv) to exploit the diagonality of the matrix A. Whether this is useful of course depends on the sizes m_1 and m_2. For DSGE models m is typically significantly larger than N and the shocks of the model constitute a significant part of the state vector.

The equations presented here are obtained by simply assuming the 2-block structure, [2.19]-[2.22], and writing down the Kalman filter recursions [2.13]-[2.18] while appropriately sequencing the operations of the prediction step, i.e. to minimise the number of matrix multiplications performed. Although the filtering equations are expressed in terms of the matrices A, B, C and Z_2 it is seen above that these are easily extracted from the matrices T, R and Z such that the input to the Kalman filter can be kept the same as for the 1-block filter if desired.

Updating equations

The updating step of the filter consists of the following set of equations, which are obtained by writing out the updating equations of the Kalman filter, [2.13] and [2.14], on block form. The contemporaneous filtering covariance matrices are obtained as

$$P_{11t|t} = P_{11t|t-1} - P_{12t|t-1} Z_2^T F_t^{-1} Z_2 P_{12t|t-1}^T \tag{2.23}$$

$$P_{12t|t} = P_{12t|t-1} - P_{12t|t-1} Z_2^T F_t^{-1} Z_2 P_{22t|t-1} \tag{2.24}$$

$$P_{22t|t} = P_{22t|t-1} - P_{22t|t-1} Z_2^T F_t^{-1} Z_2 P_{22t|t-1} \tag{2.25}$$

and the means as

$$X_{1t|t} = X_{1t|t-1} + P_{12t|t-1} Z_2^T F_t^{-1} \left(Y_t - d - Z_2 X_{2t|t-1} \right) \tag{2.26}$$

$$X_{2t|t} = X_{2t|t-1} + P_{22t|t-1} Z_2^T F_t^{-1} \left(Y_t - d - Z_2 X_{2t|t-1} \right) \tag{2.27}$$

where

$$F_t = Z P_{t|t-1} Z^T + H = Z_2 P_{22t|t-1} Z_2^T + H \tag{2.28}$$

is assumed to be positive definite.

Let

$$F_t^{-1} = \tilde{F}_t^T \tilde{F}_t \tag{2.29}$$

and use the matrices

$$P_{12}^* = P_{12t|t-1} Z_2^T \tilde{F}_t^T \tag{2.30}$$

$$P_{22}^* = P_{22t|t-1} Z_2^T \tilde{F}_t^T \tag{2.31}$$

and the vector

$$v_t^* = \tilde{F}_t \left(Y_t - d - Z_2 X_{2t|t-1} \right) \tag{2.32}$$

for the matrix multiplications above and for the computation of the likelihood at time t (see below). In our implementations the matrix $\tilde{F}_t$ is obtained as the inverse of the Cholesky factor of F_t, i.e. it is not the Cholesky factor of F_t^{-1}.

Prediction equations

The prediction step is given by the following set of equations which are obtained by writing out equations [2.17] and [2.18] on block form.

First

$$P_{11t+1|t} = A P_{11t|t} A^T + Q$$

which, due to the assumed diagonality of A, is obtained as

$$P_{11t+1|t} = P_{11t|t} \odot A_{11} + Q \tag{2.33}$$

where

$$A_{11} = diag\,(A)\,diag\,(A)^T$$

(here $\odot$ denotes element-by-element multiplication, Hadamard product). Next

$$P_{12t+1|t} = AP_{11t|t}A^TB^T + AP_{12t|t}C^T + QB^T =$$

$$P_{11t+1|t}B^T + AP_{12t|t}C^T$$

or

$$P_{12t+1|t} = P_{11t+1|t}B^T + L_t \tag{2.34}$$

where

$$L_t = \left(P_{12t|t}C^T\right) \odot A_{12}$$

and

$$A_{12} = diag\,(A)\,1_{m_2}^T$$

where 1_{m_2} is a (column) vector of dimension m_2 containing ones.

The block corresponding to the endogenous variables is obtained as

$$P_{22t+1|t} = BAP_{11t|t}A^TB^T + BAP_{12t|t}C^T + BQB^T +$$

$$+CP_{12t|t}^TA^TB^T + CP_{22t|t}C^T =$$

$$= BAP_{11t|t}A^TB^T + BQB^T + M_t + M_t^T + CP_{22t|t}C^T$$

where

$$M_t = BL_t$$

or

$$P_{22t+1|t} = BP_{12t+1|t} + M_t^T + CP_{22t|t}C^T \tag{2.35}$$

As a computational detail, if we let

$$G_t = BP_{12t+1|t} + M_t = B\left(P_{12t+1|t} + L_t\right)$$

then

$$P_{22t+1|t} = \frac{1}{2}\left(G_t + G_t^T\right) + CP_{22t|t}C^T$$

The one-step ahead forecast of the state vector is given by

$$X_{1t+1|t} = AX_{1t|t} = diag\,(A) \odot X_{1t|t} \tag{2.36}$$

and

$$X_{2t+1|t} = BAX_{1t|t} + CX_{2t|t} = BX_{1t+1|t} + CX_{2t|t} \tag{2.37}$$

Likelihood

The log likelihood at time t is given by

$$\log L_t = -N \log(2\pi) + \log\left(\left|F_t^{-1}\right|\right) -$$

$$-\frac{1}{2}\left(Y_t - d - Z_2 X_{2t|t-1}\right)^T F_t^{-1}\left(Y_t - d - Z_2 X_{2t|t-1}\right) =$$

$$= -N \log(2\pi) + 2\sum_{i=1}^{N} \log\left(\tilde{F}_{ii,t}\right) - \frac{v_t^{*T} v_t^*}{2} \tag{2.38}$$

where $\tilde{F}_{ii,t}$ is element (i,i) of the matrix $\tilde{F}_t$ and the log likelihood is obtained as

$$\log L = \sum_{t=1}^{T} \log L_t$$

Theoretical gains from 2-block filtering

Consider the matrix multiplication [2.1] in a 2-block context

$$P^* = \begin{bmatrix} T_{11} & T_{12} \\ T_{21} & T_{22} \end{bmatrix} \begin{bmatrix} P_{11} & P_{12} \\ P_{21} & P_{22} \end{bmatrix} \begin{bmatrix} T_{11}^T & T_{21}^T \\ T_{12}^T & T_{22}^T \end{bmatrix} \tag{2.39}$$

Without any assumptions on the matrices P and T, performing the multiplication [2.39] naively involves $8+8=16$ ordinary matrix multiplications for the submatrices of the partitioned matrices. Recognising the symmetry of P and the output matrix P^* this is reduced to 14 matrix multiplications, some of those involving symmetric input or output matrices.[4] Further assuming that T_{11} is diagonal and that $T_{12}=0$ the number of multiplications is reduced to 2 Hadamard products plus 6 multiplications. Finally, also taking into account the special structure of T_{21} (see expression [2.19]) allows us to perform the multiplication [2.39] using 2 Hadamard products and 5 multiplications, as shown above in writing out the prediction equations for the 2-block Kalman filter.

For large matrices and $m_1 \approx m_2$ these matrix multiplication counts would provide a good estimate of the time savings from 2-block filtering. In practice, e.g. for the DSGE models considered in the experiments of this paper, we expect the time savings to be smaller than indicated by the counts. The main value of this theoretical exercise is instead that it clarifies the contribution of the various DSGE model specific assumptions in reducing the number of operations of the Kalman filter prediction step.

[4]Clearly there exist more sophisticated schemes which reduce the number of multiplications required, e.g. the Strassen algorithm for general matrix multiplication which would require $7+7=14$ multiplications if no assumptions are imposed on P and T. However, we believe that the Strassen algorithm, or related algorithms, is of little practical interest in our context.

Moving exogenous variables

Assume that the DSGE model contain exogenous AR(1) shock processes which appear in the observation equation, i.e. exogenous variables for which the corresponding column in Z contain non-zero entries. For example, if the model contain unit root shocks this situation emerges. Further assume that the exogenous variables have been ordered such that the k exogenous variables that appear in the observation equation are the k last variables in the vector X_1 such that

$$Z_1 = \begin{bmatrix} \bar{Z}_1 & \bar{Z}_2 \end{bmatrix} = \begin{bmatrix} 0 & \bar{Z}_2 \end{bmatrix}$$

where $\bar{Z}_1$ has dimension $N \times \tilde{m}_1$ where $\tilde{m}_1 = m_1 - k$. These k exogenous AR(1) variables are moved into the vector of endogenous variables forming the new blocks, $\tilde{X}_1$ and $\tilde{X}_2$ with dimensions $\tilde{m}_1$ and $\tilde{m}_2$ where $\tilde{m}_2 = m_2 + k$. The first vector thus contains exogenous variables which do not enter the observation equation and the second vector contains exogenous variables that appear in the observation equation and endogenous variables.

Let

$$\tilde{Z}_2 = \begin{bmatrix} \bar{Z}_2 & Z_2 \end{bmatrix}$$

and define $\tilde{A}\left(\tilde{m}_1 \times \tilde{m}_1\right), \tilde{B}\left(\tilde{m}_2 \times \tilde{m}_1\right), \tilde{C}\left(\tilde{m}_2 \times \tilde{m}_2\right)$ via

$$T = \begin{bmatrix} A & 0 \\ BA & C \end{bmatrix} = \begin{bmatrix} \tilde{A} & 0 \\ \tilde{B}\tilde{A} & \tilde{C} \end{bmatrix}$$

where $\tilde{B}$ is the lower left $\tilde{m}_2 \times \tilde{m}_1$ matrix of R.

Further, let $\tilde{Q}\left(\tilde{m}_1 \times \tilde{m}_1\right)$ be the upper-left submatrix of Q. Also, let R^*_{22} denote the $\left(\tilde{m}_2 \times \tilde{m}_2\right)$ lower-right submatrix of RQR^T. The modified 2-block filter is now obtained by replacing A, B, C, Q, P_{ij}, X_i and Z_2 with $\tilde{A}, \tilde{B}, \tilde{C}, \tilde{Q}, \tilde{P}_{ij}, \tilde{X}_i$ and $\tilde{Z}_2$ in the 2-block filter presented above and making the adjustment in the prediction step for $\tilde{P}_{22}$

$$\begin{aligned} \tilde{P}_{22t+1|t} &= \tilde{B}\tilde{A}\tilde{P}_{11t|t}\tilde{A}^T\tilde{B}^T + \tilde{B}\tilde{A}\tilde{P}_{12t|t}\tilde{C}^T + R^*_{22} + \tilde{C}\tilde{P}^T_{12t|t}\tilde{A}^T\tilde{B}^T + \tilde{C}\tilde{P}_{22t|t}\tilde{C}^T = \\ &= \tilde{B}\tilde{P}_{12t+1|t} + \tilde{C}P^T_{12t|t}\tilde{A}^T\tilde{B}^T + \tilde{C}\tilde{P}_{22t|t}\tilde{C}^T + R^*_{22} - \tilde{B}\tilde{Q}\tilde{B}^T \end{aligned} \tag{2.40}$$

To aid understanding, note that if $k = 0$

$$R^*_{22} = \tilde{B}\tilde{Q}\tilde{B}^T = BQB^T$$

A similar modification could be applied if the model contains an exogenous VAR block. The VAR block would be moved into the endogenous variables block independent of whether the variables appear in the observation equation or not. In practice the modification becomes interesting for models which contain at most a few - say one or two - exogenous variables in the observation equation. If many exogenous variables appear in the observation equation and/or the model contain many exogenous VAR variables the more general 4-block filter presented in the next section should instead be applied.

From a user's perspective the modification of the filter is not an issue. All that needs to be kept in mind is that exogenous variables which appear in the observation equation are placed in the second block.

2-block filter with a univariate filtering step

In this section we show how to use the approach of Koopman and Durbin (2000) with the 2-block filter. In their univariate approach the observations are introduced one at a time in the updating step. In this way two matrix multiplications and the inversion of the matrix F_t can be avoided in the Kalman filter. For DSGE models the matrix Z is typically sparse and hence the multiplications in [2.15] and [2.16] can be performed efficiently, perhaps reducing the rationale for univariate filtering.

Define

$$X_{it|t,1} = E\left[X_{it,1}|Y_{1:t-1}\right] = X_{it|t-1} \tag{2.41}$$

$$P_{ijt|t,1} = E\left[\left(X_{it,1} - X_{it|t,1}\right)\left(X_{jt,1} - X_{jt|t,1}\right)|Y_{1:t-1}\right] = P_{ijt|t-1} \tag{2.42}$$

$$X_{it|t,k} = E\left[X_{it,k}|Y_{1:t-1}, y_{t,1}, \ldots, y_{t,k-1}\right] \tag{2.43}$$

$$P_{ijt|t,k} = E\left[\left(X_{it,k} - X_{it|t,k}\right)\left(X_{jt,k} - X_{jt|t,k}\right)|Y_{1:t-1}, y_{t,1}, \ldots, y_{t,k-1}\right] \tag{2.44}$$

for $i = 1, 2$, $j = i, 2$, $k = 1, \ldots, N$ and where $y_{.,k}$ denotes the k^{th} element of the observation vector such that

$$X_{it|t} = X_{it|t,N+1} \tag{2.45}$$

$$P_{ijt|t} = P_{ijt|t,N+1} \tag{2.46}$$

The updating equations can then be written as

$$P_{ijt|t,k+1} = P_{ijt|t,k} - K_{it,k}F_{t,k}^{-1}K_{jt,k}^{T} \tag{2.47}$$

$$X_{it|t,k+1} = a_{it|t,k} + K_{it,k}F_{t,k}^{-1}v_{t,k} \tag{2.48}$$

where

$$v_{t,k} = y_{t,k} - Z_{2,k}X_{2t|t,k} \tag{2.49}$$

$$F_{t,k} = Z_{2,k}K_{2t,k} \tag{2.50}$$

$$K_{1t,k} = P_{12t,k}Z_{2,k}^{T} \tag{2.51}$$

$$K_{2t,k} = P_{22t,k}Z_{2,k}^{T} \tag{2.52}$$

Here $Z_{2,k}$ denotes the k^{th} row of Z_2 such that the vectors K_1 and K_2 have dimensions m_1 and m_2 respectively and F and v are scalars. The 2-block filter with univariate filtering is thus obtained by replacing [2.23]-[2.28] with [2.45]-[2.52]. The experiments using the univariate filtering approach in this paper only considers the case when the covariance matrix of the measurement errors, H, is diagonal.

2.5.2 4-block filter

Assume that the matrices T, Z, R and Q have the structure [2.5]-[2.8] where A_1 is a diagonal $m_1 \times m_1$ matrix, A_2 and A_3 have dimensions $m_2 \times m_2$ and $m_3 \times m_3$ respectively, the matrices B_1, B_2, B_3 and C have dimensions $m_4 \times m_1, m_4 \times m_2, m_4 \times m_3$ and $m_4 \times m_4$ respectively and Z_3 and Z_4 have dimensions $N \times m_3$ and $N \times m_4$.

The block structure of the covariance matrices is described by

$$P_{t|s} = \begin{bmatrix} P_{11t|s} & P_{12t|s} & P_{13t|s} & P_{14t|s} \\ & P_{22t|s} & P_{23t|s} & P_{24t|s} \\ & & P_{33t|s} & P_{34t|s} \\ & & & P_{44t|s} \end{bmatrix}$$

Again, the block filter is obtained by simply writing out the Kalman filter recursions under the assumed state-space structure and appropriately sequencing the operations of the prediction step.

Updating equations

Define

$$\tilde{P}_{it} = \tilde{P}_{i3t|t-1} Z_3^T + \tilde{P}_{i4t|t-1} Z_4^T \tag{2.53}$$

for $i = 1, 2, 3, 4$ such that

$$F_t = Z_3 \tilde{P}_{3t} + Z_4 \tilde{P}_{4t} + H \tag{2.54}$$

Note again that if sparseness of Z_3 and Z_4 is exploited [2.53] and [2.54] can be performed particularly efficiently.

The updating covariance matrix blocks are then obtained as

$$P_{ijt|t} = P_{ijt|t-1} - P_{it} P_{jt}^T \tag{2.55}$$

for $i = 1, 2, 3, 4$ and $j = i, \ldots, 4$ where

$$P_{it} = \tilde{P}_{it} \tilde{F}_t^T$$

and

$$F_t^{-1} = \tilde{F}_t^T \tilde{F}_t$$

The means are given by

$$X_{it|t} = X_{it|t-1} + P_{it} v_t^* \tag{2.56}$$

where

$$v_t^* = \tilde{F}_t \left(Y_t - d - Z_2 X_{2t|t-1} \right)$$

and the likelihood is obtained as described previously for the 2-block filter.

Prediction equations

First, define

$$A_{11} = diag\,(A_1)\,diag\,(A_1)^T \tag{2.57}$$

and

$$A_{1i} = diag\,(A_1)\,1_{m_i}^T \tag{2.58}$$

for $i = 2, 3, 4$ such that A_{1i} is an $m_1 \times m_i$ matrix.

The prediction covariance matrix blocks are obtained via the following sequence of steps. The first row of blocks, i.e. P_{11} to P_{14}, is obtained via

$$P_{11t+1|t} = A_{11} \odot P_{11t|t} + Q_1 \tag{2.59}$$

$$P_{12t+1|t} = A_{12} \odot \left(P_{12t|t} A_2^T\right) \tag{2.60}$$

$$P_{13t+1|t} = A_{13} \odot \left(P_{13t|t} A_3^T\right) \tag{2.61}$$

$$P_{14t+1|t} = L_{1t} + \sum_{i=1}^{3} P_{1it+1|t} B_i^T \tag{2.62}$$

where

$$L_{1t} = A_{14} \odot \left(P_{14t|t} C^T\right)$$

The second row of blocks, P_{22} to P_{24}, is obtained as

$$P_{22t+1|t} = A_2 P_{22t|t} A_2^T + Q_2 \tag{2.63}$$

$$P_{23t+1|t} = A_2 P_{23t|t} A_3^T \tag{2.64}$$

$$P_{24t+1|t} = L_{2t} + \sum_{i=1}^{3} P_{2it+1|t} B_i^T$$

$$= L_{2t} + P_{12t+1|t}^T B_1^T + P_{22t+1|t} B_2^T + P_{23t+1|t} B_3^T \tag{2.65}$$

where

$$L_{2t} = A_2 P_{24t|t} C^T$$

The third row of blocks, P_{33} and P_{34}, is obtained through

$$P_{33t+1|t} = A_3 P_{33t|t} A_3^T + Q_3 \tag{2.66}$$

$$P_{34t+1|t} = L_{3t} + \sum_{i=1}^{3} P_{3it+1|t} B_i^T \tag{2.67}$$

where

$$L_{3t} = A_3 P_{34t|t} C^T$$

Finally, let

$$M_t = \sum_{i=1}^{3} B_i \left(P_{i4t+1|t} + L_{it} \right)$$

Then

$$P_{44t+1|t} = \frac{1}{2} \left(M_t + M_t^T \right) + C P_{44t|t} C^T \tag{2.68}$$

The prediction means are obtained as

$$X_{1t+1|t} = diag\left(A_1\right) \odot X_{1t|t} \tag{2.69}$$

$$X_{2t+1|t} = A_2 X_{2t|t} \tag{2.70}$$

$$X_{3t+1|t} = A_3 X_{3t|t} \tag{2.71}$$

and

$$X_{4t+1|t} = C X_{4t|t} + \sum_{i=1}^{3} B_i X_{it+1|t} \tag{2.72}$$

Univariate filtering step: 4-block filter

The univariate filtering equations are analogous to the 2-block case. The definitions [2.41]-[2.46] and the relations [2.47] and [2.48] now hold for $i = 1, 2, 3, 4$, $j = i, \ldots, 4$ $k = 1, \ldots, N$ where v, F and K_i are instead defined as

$$v_{t,k} = y_{t,k} - Z_{3,k} X_{3t|t,k} - Z_{4,k} X_{4t|t,k} \tag{2.73}$$

$$F_{t,k} = Z_{3,k} K_{3t,k} + Z_{4,k} K_{4t,k} \tag{2.74}$$

$$K_{it,k} = P_{i3t,k} Z_{3,k}^T + P_{i4t,k} Z_{4,k}^T \tag{2.75}$$

Although the block filter and univariate filtering approaches are largely complementary in increasing Kalman filtering speed, as the number of blocks increase it is clear that in practical implementations the efficiency of univariate filtering is negatively affected by working on several smaller arrays.

2.6 Experimental setup

The computational experiment has two objectives. First, and most importantly, we want to assess the performance of the block Kalman filter approach to faster Kalman filtering. Second, and more broadly, we want to provide a fuller picture of the many factors that affect Kalman filtering performance for DSGE models in practice.

Three DSGE models of different sizes are used to evaluate the computational performance of the filter implementations, where the large model is of primary interest. For each model a benchmark problem is constructed which consists of evaluating the likelihood with the Kalman filter a number of times.[5]

The filters are coded in Matlab, in Fortran and as Fortran Mex functions to be called from Matlab. The default software setup consists of Matlab R2008a, Intel Visual Fortran (IVF) compiler 10.1.019 and the BLAS routines available in Matlab (MKL, *libmwblas.lib*). The Fortran Mex functions are always compiled with the default *mexopts.bat* compiler options provided by the Mathworks.[6]

Results are presented for two computers: (i) AMD Opteron 275, 2.2 GHz and (ii) Dell 690, with Intel Xeon 5160 3.00 GHz. It can be expected that the block filter has better performance for machines with a relatively limited cache memory since it generally works on smaller matrices. The Dell 690 is a high performance workstation with a large cache memory (4MB L2 cache) and it is therefore interesting to present results for this machine.

The Kalman filter implementations are categorized according to four criteria:

1. The block structure, i.e. the number of blocks of the filter.
2. Symmetric/non-symmetric. This refers to whether the filter is implemented using symmetric (*dsymm*, *dsyrk*, *dsyr2k*) or non-symmetric (*gemm*) BLAS routines.
3. Whether or not sparseness of Z is exploited in computations.
4. Whether or not the univariate filtering approach of Koopman and Durbin (2000) is applied.

This yields $4 \times 2 \times 2 \times 2 = 32$ possible implementations and a subset of these are considered in our experiments. On a priori grounds we restrict the number of non-symmetric implementations considered.

The benchmark Fortran mex filter, referred to below, is the 1-block filter which utilizes symmetric BLAS routines but neither exploits sparseness of Z nor

[5]Since Kalman filtering time is independent of the parameter point, θ, at which the likelihood is evaluated, assuming convergence to the steady state solution does not occur, we simply choose an arbitrary point for evaluation. An alternative would be to use the filter in the Metropolis-Hastings algorithm.

[6]We have established that for this software setup execution in Fortran and execution of Fortran Mex files in Matlab yield similar execution times. Therefore no results from runs using stand-alone Fortran are presented. However, in less up-to-date software and/or hardware environments there sometimes appears to be quite a significant cost of using Fortran routines via Matlab in comparison to stand-alone Fortran.

use univariate filtering. The benchmark Matlab filter is the 1-block Matlab implementation of the Kalman filter.

Concerning the Matlab implementations it should be noted that explicit special routines for symmetric matrices are unavailable in Matlab and that exploitation of sparseness of Z and univariate filtering substantially *increase* Kalman filtering time for the three cases considered here. Therefore, results are presented for Matlab implementations which differ only along the block structure dimension.

Below the three example models are briefly described. The first two models have two blocks and the last model contains four blocks. If the $(m+1)-$block filter is applicable for a model, then the $m-$block filter is also applicable. The standard, 1-block, Kalman filter is applied to all models.

2.6.1 A small scale model (An 2005)

The New Keynesian model used by An (2005) features price stickiness via quadratic adjustment costs in price setting. The block structure of the model is described generally by $\mathbf{m}_4 = (2, 0, 1, 4)$ or in the "natural" 2-block form as $\mathbf{m}_2 = (2, 5)$ and $N = 3$ observed variables are used to estimate the model. The presence of an exogenous shock in the observation equation is due to the assumption of a unit root technology shock. The 2-block Kalman filter (with the modification described in section 2.5.1) is applied to this model.

2.6.2 An intermediate scale model (Smets and Wouters 2003)

Next the 2-block filter is applied to a slightly smaller version of the model presented by Smets and Wouters (2003). The block structure of the model is described by $\mathbf{m}_4 = (8, 0, 0, 11)$ or $\mathbf{m}_2 = (8, 11)$ and $N = 5$ observed series are used for estimation. The 2-block filter is applied to the model.

2.6.3 A large scale model (Adolfson et.al 2007)

The open economy DSGE model presented in Adolfson, Laséen, Lindé and Villani (2007) has the block structure $\mathbf{m}_4 = (11, 14, 16, 24)$. With two or three blocks the block dimensions are $\mathbf{m}_2 = (11, 54)$ or $\mathbf{m}_3 = (11, 14, 40)$. The 2-, 3- and 4-block filters are applied to this model.

2.7 Computational performance

2.7.1 Small scale model

For the small scale model the benchmark problem consists of evaluating the likelihood $R = 20\,000$ times on the AMD Opteron machine (using one processor) and 12 variants of 1- and 2-block Kalman filters, implemented as Matlab or Fortran mex files, are compared in the Matlab environment. The results are collected in Table 2.1 where

Table 2.1 Kalman filtering time, small model

Language	Block	Symm.	Sparse Z	DK	Time
Matlab	1	-	No	No	1369
Matlab	2	-	No	No	2107
Fortran	1	No	No	No	151
Fortran	1	Yes	No	No	188
Fortran	1	Yes	Yes	No	153
Fortran	1	Yes	No	Yes	117
Fortran	1	Yes	Yes	Yes	100
Fortran	2	No	Yes	No	187
Fortran	2	Yes	No	No	265
Fortran	2	Yes	Yes	No	226
Fortran	2	Yes	No	Yes	179
Fortran	2	Yes	Yes	Yes	148

AMD Opteron, 2.2 Ghz, Matlab version 7.6, R2008a, Intel Visual Fortran 10.1.019.

the best performing algorithm/implementation has been given the normalized wall-clock time 100.

The best performing filter is the Fortran mex 1-block filter with a univariate filtering step and which exploits the sparseness of Z. This filter produces $R = 20\ 000$ likelihood evaluations in 11.5 seconds and it is more than 13 times faster than the benchmark Matlab filter and $(188 - 100)/188 = 47\%$ faster than the benchmark Fortran mex filter. The filter with a univariate filtering step is $(188 - 117)/188 = 38\%$ faster than the Fortran mex benchmark. As anticipated, for a model of this size and block structure the 2-block filter does not improve performance.

For small scale models the most important aspect thus appears to be the choice of implementation language and algorithmic refinements are of secondary importance. If Kalman filtering speed is an issue in the case of a small model, e.g. because of extensive simulation, univariate filtering appears to be the single most important algorithmic ingredient in increasing speed.

2.7.2 Intermediate scale model

For the intermediate scale model the benchmark problem consists of evaluating the likelihood $R = 5\ 000$ times on the AMD Opteron (using one processor) and, as above, 12 variants of the Kalman filter are compared. To illustrate the sometimes complex interactions between algorithm, implementation and software two setups are compared using this model:

i) "New": Matlab 7.6 (R2008a), IVF 10.1, Matlab's MKL library

ii) "Old": Matlab 7.0 (R14), Compaq Visual Fortran 6.6, IMSL library

The results are presented in Table 2.2. Note that the standardised times are not comparable across columns.

Table 2.2 Kalman filtering time, intermediate scale model

Language	Block	Symm.	Sparse Z	DK	(i) Time	(ii) Time
Matlab	1	-	No	No	441	359
Matlab	2	-	No	No	586	417
Fortran	1	No	No	No	121	207
Fortran	1	Yes	No	No	139	186
Fortran	1	Yes	Yes	No	121	162
Fortran	1	Yes	No	Yes	120	169
Fortran	1	Yes	Yes	Yes	105	149
Fortran	2	No	Yes	No	106	128
Fortran	2	Yes	No	No	137	128
Fortran	2	Yes	Yes	No	121	109
Fortran	2	Yes	No	Yes	118	122
Fortran	2	Yes	Yes	Yes	100	100

AMD Opteron, 2.2 Ghz

.

The results for the "new" setup, which are arguably those of main interest for the practitioner, are given in column (i) of Table 2.2. The best performing filter for this model and block structure is the 2-block Fortran mex filter with a univariate filtering step and exploitation of sparseness of Z. This filter implementation performs $R = 5$ 000 Kalman filter likelihood evaluations in 11.0 seconds and it is $(139 - 100)/139 = 28\%$ faster than the benchmark Fortran filter. The small performance gains come from exploiting sparseness of Z and univariate filtering whereas the block filter approach yields virtually no gain.

The results for the "old" setup, as given in column (ii) of the table, are quite different. The absolute performance of the best performing filter is now 12.6 seconds. Using this software setup, exploitation of the model's block structure is the single most important factor in increasing Kalman filtering speed. Moving to the 2-block from the 1-block filter decreases filtering time by $(186 - 128)/186 = 31\%$ whereas the univariate filtering approach, by itself, decreases time by $(186 - 169)/186 = 9\%$.

Why is this comparison interesting? To us it shows that the payoff from applying a more complex algorithm is substantially smaller when using an up-to-date Fortran compiler and BLAS library. In the "new" setup the total gain is $(139-100)/139 = 28\%$ whereas in the "old" setup it is $(186 - 100)/186 = 46\%$.

From a practical point of view, as was the case for the small-scale model, the choice of implementation language is much more important than the particularities of the algorithm. The non-symmetric Fortran 1-block filter is $441/121 = 3.6$ times faster than the corresponding Matlab implementation, a *language gain*. The *algorithm gain* of 28% is marginal in comparison. Compared with the small-scale model example the relative performance of Matlab thus becomes better as the time spent on matrix computations increases.

2.7.3 Large scale model

For the large scale model the benchmark problem consists of evaluating the likelihood $R = 1000$ times on the AMD Opteron and the Intel Xeon and 20 variants of the Kalman filter are compared. Results are presented in Table 2.3. Note again that the reported standardised times are not comparable across columns. The inclusion of the result for the Matlab implementation of the Kalman filter in Dynare merely serves the purpose of validating that our benchmark Matlab implementation is sufficiently efficient.[7]

On both computers the best performing filter is a non-symmetric filter which exploits the block structure and sparseness of Z. The time of 1000 likelihood evaluations using this filter is 29.3 seconds on the Opteron and 14.8 seconds on the Xeon. Since the relative performance of the filter versions are quite similar for the Opteron and Xeon the discussion below is largely restricted to results for the latter computer.[8]

First we look at the contribution of each algorithmic feature in isolation, using the 1-block symmetric Fortran mex implementation as the point of reference. Exploitation of sparseness of Z leads to a speed gain of $(199 - 174)/199 = 13\%$, the gain from univariate filtering is $(199 - 188)/199 = 6\%$ and the gain from block filtering is $(199 - 128)/199 = 36\%$. The corresponding numbers on the Opteron are 12%, 6% and 44%. This shows that the block filter approach is the single most important algorithmic ingredient in increasing Kalman filtering speed for this model. It can also be noted that the gain from sub-optimally employing a 2-block structure, 17%, is larger than that from univariate filtering or sparse Z-exploitation.

The best performing filter yields a total *algorithm gain* of $(184 - 100)/100 = 46\%$ on the Xeon and 52% on the Opteron. This illustrates the complementarity of block filtering and the other approaches in increasing speed. Of practical interest is also the *language gain*, i.e. the gain from using Fortran instead of Matlab, which is $(280 - 184)/280 = 34\%$ on the Xeon and 25% on the Opteron.

Finally, concerning Matlab two things are noted. First, the Matlab block filter implementations yield time gains, although limited, when compared to the benchmark 1-block Matlab implementation. This was not the case for the smaller models considered previously. Second, using "automatic" parallelisation by enabling multithreading in Matlab and executing using the four available processors on the Opteron yields an execution time for the best performing multithreaded implementation which is *larger* than for the best performing serial implementation reported here.

2.7.4 Discussion

The Kalman filter consists of a series of operations on matrices and is therefore very suitable for implementation using Matlab. In fact, it is presumably very close to the ideal application for Matlab. Matrix operations in Matlab are performed using high

[7] The timing result for Dynare refers to the Kalman filter implementation *DiffuselikelihoodH1.m*.

[8] Tests have been carried out using other, lower-performing, hardware and the relative performance of filter implementations is similar to that reported here for the Opteron and Xeon.

Table 2.3 Kalman filtering time, large model

Language	Block	Symm.	Sparse Z	DK	Opteron Time	Xeon Time
Dynare						374
Matlab	1	-	No	No	279	280
Matlab	2	-	No	No	257	282
Matlab	3	-	No	No	233	212
Matlab	4	-	No	No	233	217
Fortran	1	No	No	No	210	184
Fortran	1	Yes	No	No	224	199
Fortran	1	Yes	Yes	No	198	174
Fortran	1	Yes	No	Yes	211	188
Fortran	1	Yes	Yes	Yes	188	164
Fortran	2	No	Yes	No	158	141
Fortran	2	Yes	No	No	187	173
Fortran	2	Yes	Yes	No	165	150
Fortran	2	Yes	No	Yes	183	169
Fortran	2	Yes	Yes	Yes	163	146
Fortran	3	Yes	No	No	148	140
Fortran	3	Yes	Yes	No	133	126
Fortran	4	No	Yes	No	100	100
Fortran	4	Yes	No	No	125	128
Fortran	4	Yes	Yes	No	109	110
Fortran	4	Yes	Yes	Yes	119	119

Matlab R2008a, IVF 10.1, Matlab BLAS library.

.

performance routines, the MKL implementation of BLAS and LAPACK.[9] However, for small models, and hence operations on small matrices, Fortran performs much better than Matlab since the relative time spent on matrix operations is smaller.

The key result to emerge from our experiments above is that the block filter approach is the only serial algorithmic approach that has the potential to significantly reduce Kalman filtering time for large scale DSGE models. Our experiments indicate that block filtering pays off exactly when computational time is becoming a concern.

The univariate filtering approach only yields small time gains for the large model. Furthermore, simply exploiting the sparseness of Z apparently reduces the rationale for univariate filtering in the context of DSGE models. However, since the different approaches are largely complementary in reducing filtering time they should be combined for maximal performance.

The univariate filtering approach and the sparse Z-exploitation is not suitable for implementation in Matlab. Although not reported in detail here we have established, for the examples considered above, that attempts to implement these features using

[9]Previous versions of Matlab use the ATLAS (Automatically Tuned Linear Algebra Software) library.

Matlab significantly *increases* computational time. The reason, in simple terms, is that the cost of introducing loops and operating on smaller arrays outweighs any computational savings.

The block filter approach can be implemented successfully in Matlab, as the experimental results for the large model above show. However, the gains from implementing a block filter in Matlab are smaller and, more importantly, the other approaches can not be implemented successfully.

The practical view adopted in this paper essentially means that we focus on Kalman filtering wall-clock time in a practically relevant and easy-to-use software and hardware environment. Abstract performance measurement, e.g. counting matrix multiplications saved, may be of limited value if computing time is the ultimate concern.

Some implications of this perspective are the following: First, for relatively small DSGE models the payoff from using a more refined Kalman filter algorithm are small and simply using the right implementation language appears to be much more important. Second, to be able to compete with recent Matlab versions in the case of large models an up-to-date Fortran compiler and high-performing implementation of BLAS must be used. Third, applying matrix routines for symmetric matrices, as implemented in state-of-the-art matrix libraries (i.e. Intel MKL), does not payoff for models of the sizes considered in this paper. Non-symmetric implementations based on *gemm* are both easier to code and apparently more efficient in practice.[10]

More generally, the size of the time gains will depend on the model size and block structure and on software and hardware characteristics. Naturally, the value of the block filter approach increases with model size and the extent to which the block structure of a model is exploitable. Our work is projected on the assumption that DSGE models will continue to be valuable tools for central banks, as well as other institutions, and that these models will grow both in size and complexity. Our experiments also indicate that the gains from using more complex Kalman filter algorithms become smaller in practice when better performing software (compiler, matrix libraries) and hardware is used.

2.8 Conclusion

The Kalman filter can be viewed as the key computational kernel in Bayesian estimation of linearised DSGE models and Kalman filtering time is becoming a concern in the context of large-scale DSGE models. This view is reinforced by efforts to parallelise the estimation of DSGE models.[Azzini et al. (2007)] However, if the parallel computing avenue is opened it is our perspective that the parallel algorithms should be based on as close to optimal as possible serial algorithms.

The block Kalman filter is based on the simple idea of writing down the Kalman filter recursion on block form and appropriately sequencing the operations of the

[10] Performance of BLAS routines for symmetric matrices is discussed in Goto and van de Geijn (2007). The empirical results in that paper confirm our own experiences, suggesting that the application of symmetric routines becomes interesting for matrix sizes above, say, 200.

prediction step. The block Kalman filter presented in this paper appear to be the only viable serial algorithmic approach to significantly reduce Kalman filtering time for large-scale DSGE models. For the large-scale example model Kalman filtering time decreased by roughly a factor 2 and most of this time decrease was attributed to block filtering. The approaches directed at the updating step of the Kalman filter, univariate filtering [Koopman and Durbin (2000)] and sparse Z-exploitation, apparently provide more limited gains for large-scale DSGE models. However, due to complementarities in decreasing Kalman filtering time the algorithmic features should be combined for maximal efficiency.

From a practical perspective, and not surprisingly, we note that the choice of implementation language and matrix library, i.e. the particular implementation of BLAS, is very important for performance. Our experimental results, using up-to-date software and optimised implementations, provide a useful guide to what is important for faster Kalman filtering for DSGE models in practice.

Bibliography

Adolfson, M., Laséen, S., Lindé, J. and Villani, M. (2007), 'Bayesian Estimation of an Open Economy DSGE Model with Incomplete Pass-Through', *Journal of International Economics* **72(2)**, 481–511.

Adolfson, M., Lindé, J. and Villani, M. (2007), 'Bayesian Inference in DSGE Models - Some Comments', *Econometric Reviews* **26(2-4)**, 173–185.

An, S. (2005), 'Bayesian Estimation of DSGE Models: Lessons from Second-order Approximations'. Working Paper, University of Pennsylvania.

Azzini, I., Girardi, R. and Ratto, M. (2007), 'Parallelization of Matlab codes under Windows platform for Bayesian estimation: a Dynare application'. Working Paper 1, Euro-area Economy Modelling Centre.

Christoffel, K., Coenen, G. and Warne, A. (2007), 'Conditional versus Unconditional Forecasting with the New Area-Wide Model of the Euro Area', *Mimeo, ECB* .

Goto, K. and van de Geijn, R. (2007), 'High performance implementation of the level-3 BLAS', *ACM Transactions on Mathematical Software* **35**(1).

Harvey, A. (1989), *Forecasting, Structural Time Series Models and the Kalman filter*, Cambridge University Press.

Koopman, S. and Durbin, J. (2000), 'Fast Filtering and Smoothing for Multivariate State Space Models', *Journal of Time Series Analysis* **21**, 281–296.

Smets, F. and Wouters, R. (2003), 'An Estimated Stochastic Dynamic General Equilibrium Model of the Euro Area', *Journal of the European Economic Association* **1(5)**, 1123–1175.

Appendix : 3-block filter

The 3-block filter is obtained by merging blocks 3 and 4 in the 4-block filter into one block. The size of this block is then $\tilde{m}_3 = m_3 + m_4$. Define the $\tilde{m}_3 \times m_1$ matrix

$$\tilde{B}_1 = \begin{bmatrix} 0 \\ B_1 \end{bmatrix}$$

the $\tilde{m}_3 \times m_2$ matrix

$$\tilde{B}_2 = \begin{bmatrix} 0 \\ B_2 \end{bmatrix}$$

the $\tilde{m}_3 \times \tilde{m}_3$ matrix

$$\tilde{C} = \begin{bmatrix} A_3 & 0 \\ B_3 A_3 & C \end{bmatrix}$$

and the $N \times \tilde{m}_3$ matrix

$$\tilde{Z}_3 = \begin{bmatrix} Z_3 & Z_4 \end{bmatrix}$$

and let R_3 be the $\tilde{m}_3 \times \tilde{m}_3$ lower-right submatrix of RQR. The matrices T and Z then have the structures

$$T = \begin{bmatrix} A_1 & 0 & 0 \\ 0 & A_2 & 0 \\ \tilde{B}_1 A_1 & \tilde{B}_2 A_2 & \tilde{C} \end{bmatrix} \tag{2.76}$$

$$Z = \begin{bmatrix} 0 & 0 & \tilde{Z}_3 \end{bmatrix} \tag{2.77}$$

The block form of the covariance matrix is then given by

$$P_{t|s} = \begin{bmatrix} P_{11t|s} & P_{12t|s} & P_{13t|s} \\ & P_{22t|s} & P_{23t|s} \\ & & P_{33t|s} \end{bmatrix}$$

Filtering equations

Let

$$\tilde{P}_{it} = P_{i3t|t-1} \tilde{Z}_3^T$$

such that

$$F_t = \tilde{Z}_3 \tilde{P}_{3t} + H$$

and, further, let

$$P_{it} = \tilde{P}_{it} \tilde{F}_t^T$$

where

$$F_t^{-1} = \tilde{F}_t^T \tilde{F}_t$$

The filtering covariance matrices are obtained from

$$P_{ijt|t} = P_{ijt|t-1} - P_{it} P_{jt}^T \tag{2.78}$$

and the filtered state as

$$X_{it|t} = X_{it|t-1} + P_{it} v_t^* \tag{2.79}$$

where

$$v_t^* = \tilde{F}_t \left(Y_t - d - Z_3 X_{3t|t-1} \right)$$

for $i = 1, 2, 3$ and $j = i, 3$.

Prediction equations

The first row of blocks, P_{11}, P_{12} and P_{13} is obtained via

$$P_{11t+1|t} = P_{11t|t} \odot A_{11} + Q_1 \tag{2.80}$$

$$P_{12t+1|t} = A_{12} \odot \left(P_{12t|t} A_2^T \right) \tag{2.81}$$

$$P_{13t+1|t} = P_{11t+1|t} \tilde{B}_1^T + P_{12t+1|t} \tilde{B}_2^T + \tilde{L}_{1t} \tag{2.82}$$

where

$$\tilde{L}_{1t} = \tilde{A}_{13} \odot \left(P_{13t|t} C^T \right)$$

and

$$\tilde{A}_{13} = diag\left(A_1 \right) 1_{\tilde{m}_3}^T$$

The second row of covariance matrix blocks, P_{22} and P_{23}, are obtained as

$$P_{22t+1|t} = A_2 P_{22t|t} A_2^T + Q_2 \tag{2.83}$$

and

$$P_{23t+1|t} = P_{12t+1|t}^T \tilde{B}_1^T + P_{22t+1|t}^T \tilde{B}_2^T + \tilde{L}_{2t} \tag{2.84}$$

where

$$\tilde{L}_{2t} = A_2 P_{23t|t} \tilde{C}^T$$

Finally

$$P_{33t+1|t} = \tilde{M}_t + \tilde{C} P_{33t|t} \tilde{C}^T + \tilde{R}_3 \tag{2.85}$$

where

$$\tilde{M}_t = \frac{1}{2} \left(M_t + M_t^T \right) \tag{2.86}$$

and

$$M_t = \tilde{B}_1 \left(P_{13t+1|t} + \tilde{L}_{1t} \right) + \tilde{B}_2 \left(P_{23t+1|t} + \tilde{L}_{2t} \right) \tag{2.87}$$

and

$$\tilde{R}_3 = R_3 - \tilde{B}_1 Q_1 \tilde{B}_1^T - \tilde{B}_2 Q_2 \tilde{B}_2^T$$

To aid understanding, note that if the model contains no exogenous variables that appear in the observation equation and no exogenous VAR processes, then $\tilde{R}_3 = 0$.

It can also be mentioned that the *dsyr2k* BLAS routine performs exactly the computation required for obtaining the symmetric matrix $\tilde{M}_t$ in the symmetric case, i.e. when only the upper (or lower) part of P_{33} is calculated. Equations [2.86] and [2.87] shows how it is implemented using non-symmetric routines, e.g. how it is implemented in the Matlab versions of the filter.

Chapter 3

Parallel particle filters for likelihood evaluation in DSGE models

Parallel particle filters for likelihood evaluation in DSGE models: An assessment[1]

abstract

Likelihood-based estimation of nonlinearly approximated DSGE models using particle filter methods is computationally very demanding. Parallel particle filters (PPF) are introduced to speedup the likelihood evaluation. The algorithms, which are based on previous work in the field of signal processing, are presented, implemented and evaluated in the context of distributed memory and the message passing model of programming. The main body of our paper focuses on parallel resampling and on the design of PPF algorithms which minimise inter-processor communication. Attention is restricted to approaches which, by construction, do not affect the statistical properties of the particle filter in any important sense.

The algorithms are applied for the estimation of a small scale standard New Keynesian model. Experiments with two PPF algorithms on three different computing clusters illustrate the computational gains that can be achieved using parallel methods. Code profiling reveals that MPI collectives performance is critical in determining the scalability properties of the filters. An overall assessment, taking into account both computational speed and code development effort, suggests that a simple parallel particle filter algorithm is a good choice in the nonlinear DSGE model estimation case.

[1]Thanks to Miodrag Bolić, Jesús Fernández-Villaverde, Sune Karlsson, Marco Ratto, Nils Smeds, Kalle Walentin for comments. The Center for Parallel Computers at the Royal Instititute of Technology in Stockholm provided computing time for experiments in the paper.

3.1 Introduction

Likelihood-based estimation of dynamic stochastic general equilibrium (DSGE) models has become increasingly popular in the last few years. The solution to a DSGE model is represented as a state-space model and filtering methods are applied for evaluation of the likelihood function. In the classical framework maximum likelihood (ML) is applied to estimate the model's structural parameters, and in the Bayesian setting a Markov Chain Monte Carlo (MCMC) algorithm is used to sample from the posterior density. If a nonlinear solution method is applied the resulting state space model is nonlinear, and possibly also non-Gaussian. In this case a particle filter (PF) can be employed to obtain an estimate of the likelihood function (henceforth we refer to this as *nonlinear estimation*).

The particle filter recursively estimates the sequence of state filtering distributions, via sequential application of the importance sampling technique, while also producing an estimate of the likelihood. Fernández-Villaverde and Rubio-Ramírez (2007) estimate a Real Business Cycle (RBC) model with stochastic volatility, using second-order perturbation methods to solve the model, and a particle filter to estimate the likelihood. Following their work An and Schorfeide (2007*a*), An (2005), Amisano and Tristani (2007), Pichler (2008), Flury and Shephard (2008) and Andreasen (2010) use similar approaches to estimate business cycle models.

The main purpose of our paper is to introduce parallel particle filter (PPF) algorithms, in the context of nonlinear DSGE model estimation. The motivation for using parallel computing methods is the computational requirement associated with particle filtering. As an example, a standard small-scale New Keynesian model is estimated using the random walk Metropolis algorithm, and a PF with $N = 40k$ particles, following An (2005). In this case we note that, (i) estimation based on a second-order approximation of the policy function is more than $1,000$ times slower than estimation of the corresponding linearised model using the Kalman filter, and (ii), more than 98% of overall nonlinear estimation time is spent on particle filtering.

An important objective in nonlinear estimation is to reduce the computing time needed to achieve a given accuracy in the particle filter likelihood estimate, or, more generally, to obtain a more favourable time-accuracy trade-off. Particle filtering time is linear in the number of particles, N, and the standard deviation of the likelihood estimate is proportional to $1/\sqrt{N}$. Parallelisation of the PF is a purely computational, brute force, approach, which 'automatically' achieves this objective. A statistical approach instead focuses on the design of improved particle filters; see Amisano and Tristani (2007) and Andreasen (2010) for attempts in this direction, in the DSGE model context. In this article we restrict attention to the parallelisation of a baseline PF, of the type used by Fernández-Villaverde and Rubio-Ramírez (2007), while noting that the methods developed could also be applied with more elaborate variants of the PF.

The presentation and development of parallel particle filter algorithms is based on previous work in the field of signal processing (Bolić (2004); Bolić, Djurić and Hong (2003); Bolić, Djurić and Hong (2005); Bolić, Djurić and Hong (2004); Brun, Teulière and Garcia (2002); Teulière and Brun (2003)). The key aspect of a PPF

algorithm concerns the parallelisation of the resampling step of the filter. Resampling is necessary in a PF, in order to avoid degeneracy of the importance sampling weights. The drawback of resampling, from a parallel computing efficiency perspective, is that it unavoidably creates a need for extensive inter-processor communication. The main body of our paper focuses on the issue of minimising communication in the PPF, subject to the restriction that the parallel resampling approach should not affect the statistical properties of the particle filter, in any important sense. Concretely, the variance of the PF likelihood estimator should not depend on the number of processors applied with the PPF algorithm.

The presentation of PPF methods proceed in the form of a comparison of two algorithms, a simple algorithm and a more complex one. The main difference between these algorithms is related to the implied inter-processor communication requirement, which is lower for the latter algorithm. The framework for the discussion is the message passing model of parallel programming, and our PPF implementations are intended for application on computing clusters with a reasonably fast interconnection network. The two algorithms are implemented using Fortran and the Message Passing Interface (MPI), and are tested on three computing clusters, using a standard New Keynesian (NK) model.

The contributions of this paper can be divided into two parts, one part which concerns the nonlinear estimation of DSGE models, and a second part, which concerns parallel particle filtering on a more general level. The application of the PPFs to a well-known small scale macroeconomic model provide economists with a case study, which illustrates the computational gains that can be attained using parallel methods. The PPF implementations are 'stress tested', using up to $P = 128$ processors, and $N = 640k$ particles, for a model setup, where the statistical performance of the PF (as measured by the variance of the likelihood estimate) is very poor. This setup implies an extreme inter-processor communication requirement. An overall assessment of the two algorithms, taking into account both computing speed and code development effort, suggests that the simpler PPF algorithm is a good choice in estimating small-scale DSGE models.

The resampling part of the particle filter is not related to the particular state space model under consideration. As a consequence the PPF algorithms suggested here can, in principle, be applied with any nonlinear and non-Gaussian state space (NLSS) model. The contributions, which concern parallel particle filtering on a more general level, are the following. First, we provide a detailed discussion on inter-processor communication in the particle filter, and suggest algorithms which attempt to minimise the communication requirement, without affecting the statistical performance of the PF. Second, previously suggested PPFs (see the references above) are interpreted within our framework. Third, a novel parallel residual systematic resampling algorithm is suggested, and contrasted with the algorithm proposed by Bolić et al. (2005). The idea of the former algorithm is to construct a parallel resampling algorithm, which performs the systematic resampling method, independently of the number of processors applied with the algorithm.

A key aspect in practise concerns the performance of MPI collective communication routines (collectives). Code profiling reveals that these routines do not reach the

optimal theoretical performance in our experiments, which implies that they account for a larger than expected fraction of the loss in parallel efficiency, as the number of processors applied with the PPF increases. This observation is robust to the use of different MPI implementations (Mpich and ScaliMPI) in our experiments, as well as to possible variations in the specific collectives used in the PPF implementations. From the applied scientist's perspective this is regrettable, especially since it is costly to perform extensive empirical performance measurement, to locate the parallel algorithm bottlenecks. However, we expect these practical experiences to be 'part of the game' in applied parallel computing.

The paper proceeds as follows. In section 3.2 the DSGE state space model is presented. In section 3.3 the particle filter is discussed, with an emphasis on resampling, since it is the most interesting part from the perspective of parallelisation. In section 3.4 the two variants of the parallel particle filter algorithm are presented, and in section 3.5 computational experiments are performed.

3.2 DSGE model solution and state-space representation

The equilibrium of a DSGE model is described by a set of nonlinear expectational equations. These consist of constraints and first-order conditions, derived from the optimising behaviour of various agents, such as consumers and firms, in a model economy. The solution to this set of equations is a policy function, g, in terms of the state variables. The policy function is defined implicitly by the equilibrium equations, and can not usually be derived in closed form, implying that numerical approximation methods must be used. The approximate policy function, $\hat{g}$, can be cast as the state equation in a nonlinear, and possibly also non-Gaussian, state-space (NLSS) model,

$$X_t = \hat{g}(X_{t-1}, \varepsilon_t; \theta), \tag{3.1}$$

where the measurement equation

$$Y_t = h(X_t, v_t; \theta),\ t = 1, \ldots, T, \tag{3.2}$$

relates the (partially) latent state vector X_t (of dimension n_X) to a vector of observed variables Y_t (n_Y). The vector ε_t (n_ε) contains the innovations and v_t (n_v) is the vector of measurement errors. The structural and auxiliary, i.e. shock, parameters of the economic model are collected in the vector θ (n_θ). To avoid stochastic singularity the condition $n_\varepsilon + n_v \geq n_Y$ has to be satisfied and for simplicity we let $n_v = n_Y$.

The appearance of the nonlinear state-space model depends on the solution method employed and on the distributional assumptions for ε and v. Perturbation, finite elements, and spectral methods are the available approaches for nonlinear approximation of the policy function (see e.g. Judd (1998), Aruoba, Fernández-Villaverde and Rubio-Ramírez (2006), Winschel (2005)). For the illustration of the parallel particle filter in this paper, the approach of Schmitt-Grohe and Uribe (2004) is used to obtain a second-order Taylor approximation of the policy function, around

the deterministic steady state. The state equation for the DSGE model is then given by a linear equation for the exogenous variables, X_1 (n_{X_1}), i.e. the shocks,

$$X_{1t} = \hat{g}_1(X_{1,t-1}, \varepsilon_t; \theta) = AX_{1,t-1} + \varepsilon_t, \tag{3.3}$$

where A is diagonal, and an equation for the endogenous predetermined variables, and a subset of the non-predetermined variables (i.e. those needed to construct the measurement equation), X_2 (n_{X_2}),

$$X_{2t} = \hat{g}_2(X_{1,t}, X_{2,t-1}; \theta). \tag{3.4}$$

Defining the state equation in this way, the measurement equation is linear,

$$Y_t = d + ZX_{2t} + v_t, \tag{3.5}$$

where Z is a selector matrix, which relates the observed variables to the state variables. The vector of innovations, ε_t, and the vector of measurement errors, v_t, are assumed to be independent and normally distributed, with diagonal covariance matrices, $\sigma^2\Sigma_\varepsilon$, and Σ_v, respectively. In a linear framework the normality assumption is crucial, since the Kalman filter, and the prediction error decomposition, is used to calculate the likelihood. In the nonlinear framework the assumption could be relaxed, but it is retained here for convenience.

A row of [3.4] is of the form

$$X^i_{2t} = \hat{g}^i_2(X_{1,t}, X_{2,t-1}) = b^T_{x,i}\hat{x}_t + \frac{1}{2}\hat{x}^T_t C_{xx,i}\hat{x}_t + \frac{1}{2}a_{\sigma\sigma,i}\sigma^2,\ i = 1, \ldots, n_{X_2}, \tag{3.6}$$

where $C_{xx,i}$ is a symmetric $n_X \times n_X$ matrix. Expressed in 'vech form' the equation is

$$X^i_{2t} = \hat{g}^i_2(X_{1,t}, X_{2,t-1}) = b^T_{x,i}\hat{x}_t + \frac{1}{2}c^T_{xx,i}\hat{x}^*_t + \frac{1}{2}a_{\sigma\sigma,i}\sigma^2,\ i = 1, \ldots, n_{X_2}, \tag{3.7}$$

where

$$\hat{x}_t = (X^T_{1t}\ X^T_{2t-1})^T,$$

$$\hat{x}^*_t = vech(\hat{x}_t\hat{x}^T_t), \tag{3.8}$$

and

$$c_{xx,i} = vech[2C_{xx,i} - diag(C_{xx,i})],\ i = 1, \ldots, n_{X_2}. \tag{3.9}$$

The vectors $b_{x,i}$ and $c_{xx,i}$, the scalar $a_{\sigma\sigma,i}$, and the symmetric matrix $C_{xx,i}$, are obtained directly, or can easily be constructed, from the output of the solution algorithm of Schmitt-Grohe and Uribe (2004). The elements of the coefficient vectors and matrices are nonlinear functions of the parameters, θ, of the economic model.

Defining the matrices

$$B = [b_{x,1}, \ldots, b_{x,n_{X_2}}]^T, \tag{3.10}$$

$$C = [c_{xx,1}, \ldots, c_{xx,n_{X_2}}]^T, \tag{3.11}$$

and the vector

$$a_{\sigma\sigma} = [a_{\sigma\sigma,1}, \ldots, a_{\sigma\sigma,n_{X_2}}]^T, \tag{3.12}$$

the equation for the endogenous state variables, [3.4], may be written as

$$X_{2t} = B\hat{x}_t + \frac{1}{2}C\hat{x}_t^* + \frac{1}{2}a_{\sigma\sigma}\sigma^2. \tag{3.13}$$

The state-space model of interest is then given by the state equation, [3.3] and [3.13], and the observation equation, [3.5]. The matrix C is of dimension $n_{X_2} \times m$, where $m = n_X(n_X + 1)/2$. The vech form is provided here, since it shows explicitly how we implement the particle filter in the computer code. The linear and Gaussian state space (LGSS) model, corresponding to a (log-)linear approximation to the policy function, is obtained by letting $C = 0$ and $a_{\sigma\sigma} = 0$.

3.3 Particle filtering

In this section particle filtering methods are reviewed, with an emphasis on resampling, since this is the key part in the parallelisation of the particle filter. More comprehensive introductions to particle filtering are provided by Doucet, de Freitas and Gordon (2001), Arulampalam, Maskell, Gordon and Clapp (2002) and Doucet, Godsill and Andrieu (2000), and a nice recent survey, intended for economists/econometricians, is provided by Creal (2009).

3.3.1 The particle filter

The objective in a Bayesian setting is to obtain a sample, $\{\theta_i\}_{i=1}^H$, from the posterior density of the parameters, θ, of the economic model,

$$p(\theta|Y_{1:T}) \propto L(Y_{1:T};\theta)\varphi(\theta), \tag{3.14}$$

where φ is the prior density, L is the likelihood function, and the data is collected in the vector $Y_{1:T} = (Y_1^T, \ldots, Y_T^T)^T$. Alternatively, in a classical framework, the objective is to obtain the maximum likelihood (ML) estimate, $\hat{\theta}_{ML}$, of θ. In both situations the evaluation of the likelihood function, for a given parameter vector, θ, is the key operation, in the state space model context.

The likelihood function of the non-linear state space (NLSS) model, [3.1] and [3.2], is given by

$$L(Y_{1:T};\theta) = P(Y_{1:T}|\theta) = \prod_{t=1}^{T} L_t, \tag{3.15}$$

where the likelihood at time t is

$$L_t = P(Y_t|Y_{1:t-1};\theta) = \int P(Y_t|X_t;\theta)P(X_t|Y_{1:t-1};\theta)dX_t. \tag{3.16}$$

Since $\hat{g}$ is an approximation to the true policy function, g, the likelihood L is an approximation to the 'true' likelihood function of the economic model[2]. In this article, however, the consequences of the choice of policy function approximation method, for inference, are not investigated.

In order to evaluate the likelihood function, [3.15] and [3.16], the one-step ahead prediction density for the state vector, $P(X_t|Y_{1:t-1};\theta)$, is needed. The conceptual solution to the filtering problem, for the NLSS model, [3.1] and [3.2], is provided by two relations. The *prediction equation*,

$$P(X_t|Y_{1:t-1};\theta) = \int P(X_t|X_{t-1};\theta)P(X_{t-1}|Y_{1:t-1};\theta)dX_{t-1}, \tag{3.17}$$

uses the state equation, [3.1], to obtain the prediction density of the state vector, and the *updating equation*,

$$P(X_t|Y_{1:t};\theta) = \frac{P(Y_t|X_t;\theta)P(X_t|Y_{1:t-1};\theta)}{\int P(Y_t|X_t;\theta)P(X_t|Y_{1:t-1};\theta)dX_t}, \tag{3.18}$$

incorporates the most recent observation, Y_t, via Bayes rule, to obtain the filtering density of the state vector. For a linear and Gaussian state-space (LGSS) model, the sequence of prediction and filtering densities are Gaussian, and they can be computed recursively using the Kalman filter, see e.g. Harvey (1989). For an NLSS model, e.g. for the model described by [3.3], [3.13], and [3.5], the densities are not analytically tractable and approximate methods are required.

The idea of particle filters is to recursively propagate discrete weighted approximations,

$$\chi_{t|t} = \{X^i_{t|t}, w^i_{t|t}\}_{i=1}^N,\ t = 1,\ldots,T, \tag{3.19}$$

to the filtering densities, $P(X_t|Y_{1:t};\theta)$, via sequential application of importance sampling. The $n_X \times 1$ vectors X^i are called particles (or samples), and N is the number of particles used for the approximation. The associated weights, w^i, are assumed to be normalised, i.e. $\sum_{i=1}^{N} w^i_{t|t} = 1$. (In this paper we use the convention that tilde, $\tilde{}$, is used as a superscript for quantities which have not been normalised.)

As a 'byproduct' of filtering, an estimate of the likelihood function in [3.15], $\hat{L}$, is obtained. The particle filter likelihood estimate can be used inside a Metropolis-Hastings algorithm to sample from the posterior density, [3.14], or in an optimisation routine, to obtain the ML estimate, $\hat{\theta}_{ML}$, see e.g. Pichler (2008) for an example of the latter approach.

Assuming that the approximation $\chi_{t-1|t-1}$, to $P(X_{t-1}|Y_{1:t-1};\theta)$, is available, particles are generated from an importance density, π,

$$X^i_{t|t-1} \sim \pi(.|X^i_{t-1|t-1}, Y_t;\theta),\ i = 1,\ldots,N. \tag{3.20}$$

[2]Fernández-Villaverde, Rubio-Ramirez and Santos (2004) provide convergence theorems for the likelihood. First, they establish conditions under which the likelihood converges to the true likelihood as the approximated policy function converges to the exact policy. Second they show that the error in the approximated likelihood gets compounded with the sample size.

It can be shown that the un-normalised weights of the filter, $\tilde{w}^i$, evolve recursively as

$$\tilde{w}^i_{t|t-1} \propto \tilde{w}^i_{t-1|t-1} \frac{P(Y_t|X^i_{t|t-1};\theta)P(X^i_{t|t-1}|X^i_{t-1|t-1};\theta)}{\pi(X^i_{t|t-1}|X^i_{t-1|t-1},Y_t;\theta)},\ i=1,\ldots,N, \tag{3.21}$$

and

$$\chi_{t|t} = \{X^i_{t|t-1}, w^i_{t|t-1}\}_{i=1}^N, \tag{3.22}$$

is the resulting approximation of the filtering density at time t, where

$$w^i_{t|t-1} = \frac{\tilde{w}^i_{t|t-1}}{\sum_{i=1}^N \tilde{w}^i_{t|t-1}}. \tag{3.23}$$

This yields the Sequential Importance Sampling (SIS) filter, comprising of a sampling step, [3.20], and an importance step, [3.21]. In the absence of resampling, which is discussed below, we have $X^i_{t|t} = X^i_{t|t-1}$ and $w^i_{t|t} = w^i_{t|t-1}$. The filter thus requires the following conditions to be fulfilled: the ability to draw from, and to evaluate, the importance density (which is satisfied in an obvious way by an appropriate choice of density), and the ability to evaluate the state density, $p(X_t|X_{t-1})$, and the observation density, $p(Y_t|X_t)$. An estimate of the likelihood at time t is provided by

$$\hat{L}_t = \sum_{i=1}^N w^i_{t-1|t-1}\tilde{w}^i_{t|t-1}, \tag{3.24}$$

and the likelihood estimate is

$$\hat{L} = \prod_{t=1}^T \hat{L}_t. \tag{3.25}$$

The simplest, and therefore most common, choice of importance density is the state density

$$\pi\left(X_t|X_{t-1},Y_t\right) = P\left(X_t|X_{t-1}\right), \tag{3.26}$$

and in this case the weight recursion, [3.21], simplifies to

$$\tilde{w}^i_{t|t-1} \propto \tilde{w}^i_{t-1|t-1} P(Y_t|X^i_{t|t-1};\theta). \tag{3.27}$$

A draw from the importance density, [3.26], is obtained by simulating from the state equation, [3.3] and [3.13]. The resulting filter requires two conditions to hold: the ability to simulate from the state equation, and the ability to evaluate the observation density, $P(Y_t|X_t;\theta)$. The dominant part of computational time in the PF is spent on sampling from [3.26], and in particular the operations described by expressions [3.8] and [3.13].

The choice of the importance density, π, is a crucial aspect in the design of a particle filter. The proposals obtained with [3.26] do not incorporate information from the current observation, Y_t, and can therefore be inefficient, e.g. since it is sensitive to

outlier observations. Improved particle filters result from a more elaborate choice of importance density, see Amisano and Tristani (2007) and Andreasen (2010) for novel approaches, in the context of DSGE model estimation.

Irrespective of the choice of importance density, the major problem of the SIS filter is that the distribution of the particle weights becomes more skewed as time proceeds, which produces an increasingly less accurate approximation of the state filtering density. In practice, after a few iterations all weight will be put on one particle and the filter collapses. (A formal statement of the degeneracy of the Sampling and Importance Sampling, SIS, filter is provided by Doucet et al. (2000), proposition 1).

In order to prevent weight degeneracy, it is necessary to perform resampling. The idea of resampling is to obtain a new approximation to the filtering density, by sampling from the existing population of particles. Resampling prunes away 'unrepresentative' particles, i.e. those with small weights, and multiply particles with large weights, thereby preventing future weight degeneracy, see e.g. Liu, Chen and Logvinenko (2001). The unequally weighted approximation to the filtering density, in [3.22],

$$\chi_{t|t}^{in} = \{X_{t|t-1}^{i}, w_{t|t-1}^{i}\}_{i=1}^{N}, \tag{3.28}$$

is replaced by the equally weighted sample,

$$\chi_{t|t}^{out} = \{X_{t|t}^{i}, w_{t|t}^{i} = N^{-1}\}_{i=1}^{N} = \{X_{t|t}^{i}\}_{i=1}^{N}, \tag{3.29}$$

whereas in the SIS filter above $\chi_{t|t}^{out} = \chi_{t|t}^{in} = \chi_{t|t}$, where $\chi_{t|t}$ is given in [3.22]. Further, we note that for an arbitrary continuous state space, the particles contained in $\chi_{t|t}^{in}$, [3.28], are all unique, i.e.

$$X^{i} = X^{j} \Longleftrightarrow i = j, \tag{3.30}$$

while, in general, this does not hold for the output sample from resampling, $\chi_{t|t}^{out}$ in [3.29].

The drawback of resampling is that the particle support is degraded, which is commonly referred to as the particle attrition problem. It has therefore been suggested to perform resampling only when degeneracy is observed, i.e. when the variance of the weights becomes substantial. Henceforth we refer to this practise as *conditional resampling*. A common strategy is to perform resampling when the estimate of the effective sample size,

$$N_{eff,t} = \frac{\tilde{S}_{t|t-1}^{2}}{\tilde{Q}_{t}} = \frac{1}{\sum_{i=1}^{N}(w_{t|t-1}^{i})^{2}}, \; 1 \leq N_{eff,t} \leq N, \tag{3.31}$$

falls below some arbitrarily chosen threshold, N_{thres}, where

$$\tilde{S}_{t|t-1} = \sum_{i=1}^{N} \tilde{w}_{t|t-1}^{i}, \tag{3.32}$$

is the sum of un-normalised weights and

$$\tilde{Q}_t = \sum_{i=1}^{N} (\tilde{w}^i_{t|t-1})^2, \tag{3.33}$$

is the sum of squared un-normalised weights. The effective sample size is interpreted as the effective number of unique particles, used for the density approximation. The effective sample size, as a fraction of the number of particles, N, is constant, when N is increased. The set of time indices, $\psi_c = \{t,\, t = 1, \ldots, T : N_{eff,t} < N_{thres}\}$, are referred to as the set of *dynamic checkpoints* (Liu et al. (2001)). More generally, a *resampling schedule* is a rule, which determines when to perform resampling.

A particle filter is defined by three characteristics: a) the importance density, b) the resampling schedule, and c) the resampling method. The particle filter used in this paper is defined by a) the usage of the transition density as the importance density, [3.26], b) the application of resampling at each time step of the filter, henceforth *full unconditional resampling*, $\psi_u = \{t,\, t = 1, \ldots, T\}$, and c) the usage of the resampling with proportional allocation (RPA) method for resampling, which is a non-standard choice of resampling method (this is discussed in the next section). In this setup the weight recursion, [3.27], simplifies further to

$$\tilde{w}^i_{t|t-1} \propto P(Y_t | X^i_{t|t-1}; \theta), \tag{3.34}$$

and the likelihood at time t is estimated by

$$\hat{L}_t = N^{-1} \tilde{S}_{t|t-1}, \tag{3.35}$$

such that the likelihood estimate, [3.25], becomes

$$\hat{L} = \prod_{t=1}^{T} N^{-1} \tilde{S}_{t|t-1}. \tag{3.36}$$

The choice to perform full unconditional resampling in our particle filter implementations follows from the purpose of the article. It implies that we consider the most extreme situation, in terms of the communication requirement associated with resampling in a parallel particle filter. The objective is also to simplify the interpretation of parallel efficiency results. By contrast, conditional resampling implies that the number of calls to the resampling procedure, per likelihood evaluation, is a random variable, $T_c = \#(\psi_c)$, $0 \leq T_c \leq T$. This makes it harder to interpret parallel efficiency results, a) since T_c depends on the difficulty of the filtering problem at hand, and b) since operations associated with resampling account for a large share of computational time, when the number of processors used with the parallel particle filter increases. A third alternative is to use *partial unconditional resampling*, which means that the resampling algorithm is called only at pre-specified, i.e. deterministically determined, iterations of the filter, e.g. every 5^{th} iteration.

The standard particle filter is summarised below:

Algorithm 1 *Standard particle filter*

1. Enter the t^{th} time step with the discrete approximation, $\chi^{out}_{t-1|t-1} = \{X^i_{t-1|t-1}, \frac{1}{N}\}^N_{i=1}$, to the time $t-1$ filtering distribution, $p(X_{t-1}|Y_{1:t-1})$.
2. (Sampling) Generate the innovations, $\{\varepsilon^i_{t|t-1}\}^N_{i=1}$, as $\varepsilon^i_{t|t-1} \backsim N(0, \sigma^2\Sigma_\varepsilon)$. Obtain the sample, $\chi_{t|t-1} = \{X^i_{t|t-1}, 1/N\}^N_{i=1}$, which approximates the prediction density, $p(X_t|Y_{1:t-1})$, by propagation of particles through the state equation, [3.3] and [3.4].
3. (Importance step) Calculate $v^i_{t|t-1} = Y_t - d - ZX^i_{t|t-1}$ and obtain the weights, $\tilde{w}^i_{t|t-1}$, by evaluating the observation density, $\tilde{w}^i_{t|t-1} = N\left(v^i_{t|t-1}; 0, \Sigma_v\right)$, $i = 1, \ldots, N$.
4. (Normalisation) Calculate the standardised weights,

$$w^i_{t|t-1} = \tilde{w}^i_{t|t-1} \diagup \tilde{S}_{t|t-1},\ i = 1, \ldots, N,$$

   where the sum of weights, $\tilde{S}_{t|t-1}$, is given in [3.32].
5. (Resampling) Sample from $\chi^{in}_{t|t} = \{X^i_{t|t-1}, w^i_{t|t-1}\}^N_{i=1}$, to produce an equally weighted sample, $\chi^{out}_{t|t} = \{X^i_{t|t}, w^i_{t|t} = \frac{1}{N}\}^N_{i=1}$, which approximates $p(X_t|Y_{1:t})$.
6. Go back to 1 if $t < T$. End if $t = T$.⊡

It is assumed that the sampling and importance steps of the PF, i.e. steps 2 and 3 of algorithm 1, are performed using one loop, i.e. loop fusion is applied. From now on we therefore refer to these parts, in combination, as the sampling and importance (SIS) step, when discussing purely computational aspects of the filter.

3.3.2 Resampling algorithms

In a parallel particle filter (PPF) resampling is the only part of the algorithm, which creates a need for frequent inter-processor communication. Further, the chosen format for the resampling output determines the amount of communication in the PPF, as described in detail in section 3.4.3. The implementation of resampling also determines the particle storage memory requirement, and it shapes the implementation of the sampling and importance (SIS) step, as described in section 3.3.3. The resampling operation is therefore a natural point of entry for a discussion on the, serial and/or parallel, implementation of the particle filter. In this section the three resampling algorithms, which are applied in the PPF algorithms, are presented.

Common methods to perform resampling include multinomial sampling (MN), residual resampling (RR), stratified sampling and systematic resampling (SR). Brief, and partial, comparisons of the statistical properties of the different resampling methods are provided by Douc and Cappe (2005) and Hol, Schn and Gustafsson (2006). Systematic resampling (SR) was introduced in the particle filtering context

by Carpenter, Clifford and Fearnhead (1999). It has been a popular choice in particle filter implementations, mainly because it is fast, while the statistical properties of the method are hard to assess formally, see also Kuensch (2005) for some properties of SR.

Bolić et al. (2003) suggests a computationally efficient variant of SR, called the residual systematic resampling (RSR) algorithm. The resampling method which is used in the parallel particle filter is named resampling with proportional allocation (RPA), see Bolić (2004). The RPA algorithm is a stratified resampling algorithm, explicitly designed for parallel implementation, and it is based on the SR and/or RSR algorithms. Here we first present the algorithm as a serial resampling algorithm, in order to contrast it with systematic resampling.

The input to the SR and RSR resampling algorithms is an $N_{in} \times 1$ vector of normalised weights,

$$q = (q^1, q^2, \ldots, q^{N_{in}})^T, \tag{3.37}$$

the input number of particles, N_{in}, the output number of particles, N_{out}, and a uniform random number, $u \sim U(0,1)$. In a particle filter with a fixed number of particles, N, resampling is always applied with $q = w = (w^1, w^2, \ldots, w^N)^T$, and $N_{in} = N_{out} = N$. Here the separation of the input and output number of particles is motivated by the use of SR and/or RSR, with $N_{in} \neq N_{out}$, in the RPA algorithm.

We let the input sample of particles to resampling,

$$\chi^{in} = \{X_{in}^m, q^m\}_{m=1}^{N_{in}}, \tag{3.38}$$

be contained in a matrix M_{in} ($n_X \times N_{in}$), where each column corresponds to a particle. The output of the RSR algorithm is an $N_{in} \times 1$ vector of replication factors,

$$r = (r^1, r^2, \ldots, r^{N_{in}})^T, \tag{3.39}$$

which satisfies $r^T \iota_{N_{in}} = N_{out}$. The notation ι_L is used throughout the paper to denote an $L \times 1$ vector of ones. The *replication factor*, or *duplication count* in the terminology of Douc and Cappe (2005), r^m, of a unique particle, is the number of times it occurs in a sample. The sample χ^{in} is assumed to contain N_{in} unique particles, see [3.30], and r^m is equal to the number of occurrences of the particle X_{in}^m (the particle in column m of M_1) in the equally weighted output sample of particles,

$$\chi^{out} = \{X_{out}^m\}_{m=1}^{N_{out}} = \{X^m\}_{m=1}^{N_{out}}. \tag{3.40}$$

The number of surviving particles, i.e. the number of particles in χ^{in} with a non-zero replication factor, is

$$N_U = \#(r^m > 0) = \sum_{m=1}^{N_{in}} I(r^m > 0) \leq N_{in}, \tag{3.41}$$

where I is the indicator function. The output sample, [3.40], may, alternatively, be represented in *compact form*:

$$\chi_c^{out} = \{(X_c^m, r_c^m) : r_c^m > 0\}_{m=1}^{N_U}, \tag{3.42}$$

i.e. it is expressed as N_U unique particles, with weights proportional to their replication factors. Particle X_c^m in χ_c^{out} appears $r_c^m > 0$ times in χ^{out}. Our main motive for using the replication factors, r, and the associated compact form, [3.42], is that it allows for a reduction in the inter-processor communication in the parallel particle filter, as further discussed in section 3.4.3.

When the systematic resampling (SR) algorithm is used, the output may alternatively be an $N_{out} \times 1$ index vector, i, which contains the indices of the resampled particles, i.e. the M_{in}-column indices of the resampled particles. The number of unique indices in i is equal to N_U. Examples of the vectors r and i are provided in section 3.3.3.

The output of the SR and RSR algorithms is identical for identical input arguments and we say that the algorithms are *strictly statistically equivalent*, and write this as

$$SR \Leftrightarrow RSR.$$

It simply means that both algorithms perform the systematic resampling (SR) method, while they differ computationally. The replication factor of SR/RSR, r^m, is either $\lfloor q^m N_{out} \rfloor$ or $\lceil q^m N_{out} \rceil$, and the expected replication factor, conditional on the vector of weights, q, is

$$E\left(r^m | q\right) = N_{out} q^m, \tag{3.43}$$

i.e. the resampling method satisfies the 'proper weighting', or 'unbiasedness', condition, see e.g. Douc and Cappe (2005).

Next, the SR and RSR algorithms are presented.

Algorithm 2 *Systematic resampling:* $r/i = SR(q, N_{in}, N_{out}, u)$

$u = u/N_{out}$
$[j = 0]$
for $m = 1 : N_{in}$
    $k = 0$
    $s = s + q^m$
    while $(s > u)$
        $k = k + 1$
        $u = u + N_{out}^{-1}$
        $[j = j + 1]$
        $[i^j = m]$ // [Store index]
    end
    $r^m = k$ // Store replication factor
end $\boxdot$

The SR algorithm uses two loops and has complexity $O(\max(N_{in}, N_{out}))$. The operations needed to produce the index vector, i, are placed within square brackets. As shown by Bolić et al. (2004), if the desired output from resampling is the vector of replication factors, r, the SR algorithm can be modified. The residual systematic resampling (RSR) algorithm only uses one loop and has complexity $O(N_{in})$. In this paper the SR algorithm is used whenever the desired output from resampling is the

index vector, i, while the RSR algorithm is applied when the vector of replication factors, r, is wanted.

Algorithm 3 *Residual systematic resampling:* $r = RSR(q, N_{in}, N_{out}, u)$

$u = u/N_{out}$
for $m = 1 : N_{in}$
  $r^m = \lfloor (q^m - u)N_{out} \rfloor + 1$ // Store replication factor
  $u = u + r^m/N_{out} - q^m$
end $\boxdot$

Systematic sampling (SS), see e.g. Tillé (2006), may be viewed as the special case of systematic resampling (SR), where the weights satisfy the restriction

$$0 < N_{out}\ q^m < 1,\ m = 1, \ldots, N_{in}, \tag{3.44}$$

such that r^m equals zero or one. In this case the SR algorithm can be simplified further, e.g. the *while* statement in algorithm 2 is not needed. It can be shown that the residual resampling (RR) algorithm, with systematic sampling (SS) applied for remainder resampling, (RR-SS) is equivalent to the SR algorithm, i.e.

$$SR \Leftrightarrow RSR \Leftrightarrow RR - SS. \tag{3.45}$$

The main implication of [3.45] is that many of the properties of systematic sampling (SS), described in the survey sampling literature, carry over directly to SR, see appendix A for some examples. It also implies that the RR algorithm, with the SR, RSR or SS algorithms applied for remainder resampling, should not be used, since it is computationally inferior to RSR[3].

Resampling with proportional allocation (RPA) is a method based on stratified sampling with proportional allocation, which is explicitly designed for parallel implementation. The particles in χ^{in} are divided into $\tilde{P}$ groups, with $\tilde{M} = N_{in}/\tilde{P}$ particles per group, where we need to assume $\mathrm{mod}(N_{in}, \tilde{P}) = 0$. The vector of weights is partitioned as

$$q = (q_1^T q_2^T \ldots q_{\tilde{P}}^T)^T, \tag{3.46}$$

where q_p is the $\tilde{M} \times 1$ sub-vector of weights of group p, and the elements of q and q_p are related through

$$q^{(p-1)\tilde{M}+k} = q_p^k,\ k = 1, \ldots, \tilde{M},\ p = 1, \ldots, \tilde{P}. \tag{3.47}$$

The sum of the normalised particle weights for the groups is

$$S_p = q_p^T \iota_{\tilde{M}},\ p = 1, \ldots, \tilde{P}. \tag{3.48}$$

[3]We mention this equivalence because it does not appear to be obvious in the particle filtering literature. Hol et al. (2006) and Bolić (2004) suggest SR for remainder resampling in the RR algorithm. Further, the proof of Lemma 2 in Kuensch (2005) follows directly from the equivalence and results available in the survey sampling literature.

In the first stage of the RPA algorithm, *inter-resampling*, the group replication factors, N_p, are calculated using the RSR algorithm, where the $\tilde{P} \times 1$ input vector of group weights is given by

$$s = (S_1, S_2, \ldots, S_{\tilde{P}})^T \tag{3.49}$$

Intuitively, each group is treated as a single 'aggregated' particle. Proportional allocation means that the group replication factor, N_p, is roughly proportional to the group weight, S_p, and

$$\sum_{p=1}^{\tilde{P}} N_p = N_{out}. \tag{3.50}$$

In inter-resampling the desired output is a vector of replication factors and since $N_{in} = \tilde{P} << N = N_{out}$ the RSR algorithm is much faster than the SR algorithm here.

In the second stage of the algorithm, *intra-resampling*, resampling is performed within each group to produce the $\tilde{M} \times 1$ vector of replication factors of the particles in the group, r_p, where $r_p^T \iota_{\tilde{M}} = N_p$, or the $N_p \times 1$ index vector, i_p. The output vectors of replication factors or indices are given by

$$r = \left(r_1^T \; r_2^T \ldots r_{\tilde{P}}^T\right)^T, \tag{3.51}$$

and

$$i = \left(i_1^T \; i_2^T \ldots i_{\tilde{P}}^T\right)^T. \tag{3.52}$$

The uniform random numbers needed with the algorithm are collected in the vector $U = (u^*, u^1, \ldots, u^{\tilde{P}})^T$, $u^* \sim U(0,1)$, $u^p \sim U(0,1)$, $p = 1, \ldots, \tilde{P}$.

Algorithm 4 *Proportional allocation (RPA): $r/i = RPA_{\tilde{P}}(q, N_{in}, N_{out}, U)$*

1. Partition the vector of weights, $q = (q_1^T \; q_2^T \; \ldots q_{\tilde{P}}^T)$, and calculate $S_p = q_p^T \iota_{\tilde{M}}$, for $p = 1, \ldots, \tilde{P}$, where $\tilde{M} = N_{in}/\tilde{P}$. Collect the group weights in the vector s, in [3.49].

2. *Inter-resampling.* Obtain the group replication factors

$$n = (N_1, N_2, \ldots, N_{\tilde{P}})^T, \tag{3.53}$$

$$n = RSR(s, \tilde{P}, N_{out}, u^*). \tag{3.54}$$

3. *Intra-resampling.* Resample within each group to produce the vector of replication factors, r_p, for the particles in the group.

for $p = 1 : \tilde{P}$

$$r_p = RSR(S_p^{-1} q_p, \tilde{M}, N_p, u^p) \tag{3.55}$$

$$[\text{or } i_p = SR(S_p^{-1} q_p, \tilde{M}, N_p, u^p)] \tag{3.56}$$

end ⊡

The expected replication factor for the $\text{RPA}_{\tilde{P}}$ algorithm, conditional on the vector of weights, is

$$E\left(r^{(p-1)\tilde{M}+k}|q\right) = E\left(N_p S_p^{-1} q^{(p-1)\tilde{M}+k}|q\right) = N_{out} q^{(p-1)\tilde{M}+k}, \tag{3.57}$$

i.e. the RPA algorithm satisfies the proper weighting condition, [3.43].

In section 3.4.2 a parallel version of the RPA algorithm is presented, where the number of groups, $\tilde{P}$, is chosen to be equal to the number of processors applied with the algorithm, P, i.e. $\tilde{P} = P$. The resampling method, i.e. the number of strata used with the RPA algorithm, is therefore defined by the number of processors, and

$$RPA_{\tilde{P}'} \nLeftrightarrow RPA_{\tilde{P}''} \text{ if } \tilde{P}' \neq \tilde{P}''. \tag{3.58}$$

It is easy to realise this by noting that the number of uniform random numbers used with the $\text{RPA}_{\tilde{P}}$ algorithm is $\tilde{P}+1$. If $\tilde{P}=1$ the RPA algorithm is equivalent to SR,

$$SR \Leftrightarrow RPA_1, \tag{3.59}$$

i.e. the algorithm consists only of the intra-resampling step (with $u_1 = u$). In appendix A we further illustrate the difference between the SR/RSR and $\text{RPA}_{\tilde{P}}$ algorithms by obtaining the size of the resampling support of these methods.

Strict equivalence of the sequence of resampling algorithms used in a parallel particle filter, with different numbers of processors P, is the full proof way of assuring that the statistical properties of the particle filter does not depend on P. It means that we apply the same resampling method, independent of P. Using the RPA method, with $\tilde{P} = P$, this is not satisfied, [3.58]. In a serial computing framework it would seem to be an odd choice to use the RPA resampling method with $\tilde{P} > 1$, instead of one of the methods mentioned at the beginning of this section, in the particle filter.

In section 3.5.2 we therefore perform simulations to verify that the variance of the log likelihood estimate, $\hat{l} = \log(\hat{L})$, which is of ultimate concern, does not depend systematically on $\tilde{P} = P$, when the parallel particle filter uses the parallel version of the $\text{RPA}_{\tilde{P}}$ algorithm for resampling. While the sequence of $\text{RPA}_{\tilde{P}}$ algorithms, for some set of values for $\tilde{P}$, are not strictly statistically equivalent, the number of strata used with RPA does not appear to have any major impact on particle filtering performance.

Finally, the following result shows how to apply the SR and RSR algorithms without prior normalisation of the vector of weights, i.e. the algorithms can be applied directly for an arbitrarily scaled vector of weights, $\tilde{q} \propto q$.

Result 5 *Let q be an $N_{in} \times 1$ vector of normalised weights, $q^T \iota_{N_{in}} = 1$, and let $\lambda \neq 0$ be an arbitrary real scalar. The output vector of replication factors, r, of the SR (or RSR) algorithm is invariant to the following scaling of the weight vector, q, and the output number of particles, N_{out}:*

$$r = SR(\lambda q, N_{in}, \lambda^{-1} N_{out}, u)$$

When $\lambda = 1$ the third input argument is the output number of particles.⊡

The result, which formalises and generalises a property noted by Bolić (2004), is easily verified once the RSR algorithm is 'solved' and expressed in matrix form. This is described in appendix A. For example, in the serial particle filter, prior normalisation of the weights, step 4 of algorithm 1, is avoided by calling the SR/RSR algorithm with arguments

$$r = SR(\tilde{w}, N, \tilde{S}^{-1}N, u), \tag{3.60}$$

where $\tilde{S} = \tilde{w}^T \iota_N$ is the sum of un-normalised particle weights, see [3.32]. In practise the un-normalised weights, $\tilde{w}$, may become arbitrarily small or large, which means that one has to be careful in order to avoid numerical problems in implementations.

3.3.3 Resampling output and the SIS step

The output from resampling is a vector of replication factors, r, and/or an index vector, i, as described in the previous section. In this section we describe how the resampling output format determines (i) the particle storage requirement, and (ii) the implementation of the sampling and importance (SIS) step, in the particle filter. In section 3.4.3 it is shown that the communication requirement of the parallel particle filter is, to a large extent, implied by the resampling output format.

Consider the particle filter with a fixed number of particles, N. The pre-resampling approximation, $\chi^{in}_{t|t}$ in [3.28], to the filtering density, $p(X_t|Y_{1:t})$, contains $N_U = N$ distinct particles, see [3.30]. The storage requirement for the particles is therefore at least an $n_X \times N$ matrix, M_{in}.

Bolić (2004) distinguishes between three types of output from resampling: (i) indices, (ii) replication factors and (iii) replication factors and arranged indices. It is shown that in order to reach the minimum particle storage requirement, which is a single $n_X \times N$ storage matrix, the output from the resampling algorithm must consist of both the replication factors and the indices, arranged in a suitable way.

Consider a case where $N_{in} = N_{out} = N = 8$, and let the particles in $\chi^{in}_{t|t}$, the input to resampling, be stored in the $n_X \times 8$ matrix

$$M_{in} = \begin{bmatrix} X^1_{in} & X^2_{in} & X^3_{in} & X^4_{in} & X^5_{in} & X^6_{in} & X^7_{in} & X^8_{in} \end{bmatrix}. \tag{3.61}$$

The systematic resampling (SR) algorithm (algorithm 2) is applied to produce the $N_{out} \times 1 = 8 \times 1$ index vector, i. Assume, for example, that the resulting index vector is

$$i = \begin{pmatrix} 1 & 3 & 3 & 3 & 3 & 5 & 5 & 8 \end{pmatrix}^T. \tag{3.62}$$

This means that particle X^1_{in} survives and is replicated once, X^2_{in} is not replicated (since the index 2 does not appear in i), X^3_{in} is replicated four times, X^4_{in} is not replicated, etc. The corresponding $N_{in} \times 1 = 8 \times 1$ vector of replication factors is given by

$$r = \begin{pmatrix} 1 & 0 & 4 & 0 & 2 & 0 & 0 & 1 \end{pmatrix}^T. \tag{3.63}$$

Index addressing, which is described by [3.62], implies that an additional matrix, M_{out} ($n_X \times N$), is needed to store the output sample of particles from resampling, $\chi^{out}_{t|t}$, since otherwise particles risk being overwritten before they can be read in the subsequent SIS step. Particles must be moved between the two matrices: reading from M_{out} ($\chi^{out}_{t-1|t-1}$) and writing to M_{in} ($\chi^{in}_{t|t}$) in the SIS step and then writing from M_{in} ($\chi^{in}_{t|t}$) to M_{out} ($\chi^{out}_{t|t}$) *some time* before the next iteration SIS step. The latter write operation can be performed immediately after the index vector, i, has been obtained from the resampling algorithm. Alternatively, *addressing without indices or replication factors* can be used, which simply means that neither i or r are constructed. Instead particles are moved directly from M_{in} to M_{out}, while performing resampling, i.e. as a part of the SR algorithm.

In the example the $n_X \times 8$ matrix containing the output sample of particles, after resampling, is

$$M_{out} = \left[\begin{array}{cccccccc} X^1 & X^2 & X^3 & X^4 & X^5 & X^6 & X^7 & X^8 \end{array} \right] = \tag{3.64}$$

$$= \left[\begin{array}{cccccccc} X^1_{in} & X^3_{in} & X^3_{in} & X^3_{in} & X^3_{in} & X^5_{in} & X^5_{in} & X^8_{in} \end{array} \right], \tag{3.65}$$

and, using programming notation, M_{out} and M_{in} are related through

$$M_{out} = M_{in}(:, i). \tag{3.66}$$

The number of unique particles in $\chi^{out}_{t|t}$, i.e. the number of unique columns of M_{out}, is equal to $N_U = 4$. The number of iterations in the *for* loop of the subsequent SIS step, i.e. the loop over the particles which are stored in the columns of M_{out}, is equal to N.

The double storage requirement, associated with index addressing, can be avoided if the output from resampling instead are the *replication factors and arranged indices*. The output of the resampling algorithm, corresponding to the example above, is now the $N_{in} \times 2 = 8 \times 2$ matrix

$$a = \left(\begin{array}{cccccccc} 1 & 4 & 2 & 1 & 0 & 0 & 0 & 0 \\ 1 & 3 & 5 & 8 & 7 & 6 & 4 & 2 \end{array} \right)^T, \tag{3.67}$$

where the first row of a^T contains the replication factors, and the second row the indices of the particles. Surviving ($r^m > 0$) particles/indices enter from left, and discarded ($r^m = 0$) particles/indices enter from right, in a^T. It is easy to modify the RSR algorithm to produce this output, see Bolić (2004) for details.

In this case a single storage matrix, M_*, for the particles is sufficient. After resampling the sample of surviving particles, χ^{out}_c in [3.42], is implicitly described by

$$M_* = \left[\begin{array}{cccccccc} X^1_{in} & * & X^3_{in} & * & X^5_{in} & * & * & X^8_{in} \end{array} \right], \tag{3.68}$$

together with the vector of replication factors, [3.63], e.g. $(X^1_c, r^1_c) = (X^1_{in}, r^1 = 1)$, $(X^2_c, r^2_c) = (X^3_{in}, r^3 = 4)$ etc. In the sampling step the updates of surviving particles can be stored into the places of the previously discarded particles, i.e. those with zero replication factor, as their addresses are now known beforehand (marked with

* in [3.68]). In the subsequent sampling step the $r_c^2 = r^3 = 4$ distinct 'prediction particles' arising from the 'filtering particle' X_{in}^3 are stored into columns 2, 3, 4 and 6 of M_*, such that no filtering particle with a nonzero replication factor is overwritten. As a consequence, we read from M_* ($\chi_{c,t-1|t-1}^{out}$), and write to M_* ($\chi_{t|t}^{in}$), only in the sampling step.

The SIS step is now implemented using a nested loop. The number of iterations in the outer *for* loop, the loop over surviving particles, is $N_U = \#(r^m > 0)$, and the number of iterations of the inner loop is $r_c^m > 0$. The outer loop runs over the indices $a_{2,1} = 1, \ldots, a_{2,N_U} = 8$, in [3.67], which point to the relevant columns of M_* in [3.68]. The total number of iterations is $r^T \iota_N = N$.

Four theoretical benefits of addressing with *replication factors and arranged indices* follow, see Bolić (2004). First, the memory requirement of particle storage is halved. Second, in a serial particle filter the particle memory, M_*, is only accessed during the sampling step. Third, since the number of unique replicated particles, N_U, is less than or equal to the total number of particles, N, the number of particle memory accesses in the sampling step is reduced. Fourth, there is a potential gain, depending on the specific model, in that some repeated calculations can be avoided in the sampling step. In our context, an example is the matrix multiplication in [3.3], which only needs to be performed once for X_c^m while it is performed r_c^m times for X_c^m when index addressing is used.

In our implementations, the operations in the SIS step proceed particle-by-particle. This is motivated by the complex structure of the SIS step loop when replication factors and arranged indices are used. Alternatively, when *addressing without indices or replication factors* or *index addressing* is used, it may be possible, again depending on the particular state-space model, to speedup the SIS step, through loop fission, and (manually implemented) loop blocking. However, to make the two variants of implementation more readily comparable in the computational evaluation this possibility is not considered here, while we return to discuss the issue in the final remarks of the paper.

3.4 Parallel particle filters

3.4.1 Overview

In this section parallel particle filter (PPF) algorithms are presented and discussed, in the context of distributed memory computers (clusters) and the message passing model of parallel programming. The PPF algorithms are implemented using Fortran and Message Passing Interface (MPI) library routines. The intended hardware architecture for the implementations is a homogeneous cluster of workstations, with a reasonably fast interconnection network.

Our work draws on the signal processing literature on parallel particle filters. Brun et al. (2002) and Teulière and Brun (2003) propose parallel standard particle filter, and Gaussian particle filter (GPF), algorithms, and test MPI implementations of these algorithms on different parallel computers. Bolić (2004) (see also Bolić et al.

(2003), Bolić et al. (2005) and Bolić et al. (2004)) develop algorithms and hardware architectures jointly, with the purpose of producing high-speed particle filters for embedded devices.

The parallel particle filters presented here are based on two assumptions:

- The statistical properties of the particle filter, e.g. the variance of the likelihood estimate, does not depend on the number of processors applied with the parallel particle filtering algorithm. (A1)
- Full unconditional resampling is applied, i.e. resampling is applied at each iteration of the particle filter. (A2)

Our objective is, simply stated, to develop computationally efficient parallel particle filters subject to these 'restrictions'.

The discussion proceed mainly in the form of a comparison of two parallel particle filter algorithm/implementations, which are named PPF1 and PPF2. The main difference between these two algorithms concerns the resampling output format. PPF1, which uses indices, is, in our view, the simple and most immediate way of implementing a parallel particle filter. PPF2, which uses replication factors (and arranged indices), is a more complex algorithm. PPF2 is primarily designed with the aim of reducing inter-processor communication in comparison with PPF1, while at the same time it requires substantially more effort to implement. It incorporates many of the ideas presented by Bolić (2004). In section 3.5, these algorithms are taken to three parallel computers, and are compared in a computational experiment.

The SIS step of the particle filter is trivially parallelisable, since there are no data dependencies between particles. This part of the PF algorithm can be characterised as *data parallel*, or *single instruction multiple data* (SIMD). The N particles are distributed to P processors, indexed as $p = 1, \ldots, P$, such that each processor, p, performs the SIS step on $M = N/P$ particles, as described for the one processor ($P = 1$) case in section 3.3.3. It is assumed here that a perfect load balance, i.e. M particles per process, is desirable in the SIS step.

Parallelising merely the SIS step is simple and should yield quick gains in execution time. The relative speedup of a parallel program is defined as

$$S(P) = T(1)/T(P), \tag{3.69}$$

where $T(P)$ is the wall-clock time of running the parallel program on P processors. The theoretical speedup of the SIS step, when viewed in isolation, is $S(P) = N/M = P$.

In order to develop efficient parallel particle filters, it is, however, necessary to focus on the parallelisation of resampling. As we discuss in detail below, resampling is the only part of the PF algorithm which creates frequent inter-processor communication, i.e. communication at each iteration of filter. Further, the simpler algorithm, consisting of a parallelised SIS step and a serial (or centralised) resampling step, is unsuitable in the distributed memory context. It requires that the full, $N \times 1$, vector of weights, $\tilde{w}$, is made available to a master process, which induces very costly communication.

In the parallel framework the resampling operation is divided into three parts. *Parallel resampling (1)* is performed using resampling algorithms which are designed to run in parallel on several processors. *Particle routing (2)* refers to the communication, i.e. sending/receiving, of particles between processes, which is necessary to restore load balance in the subsequent SIS step, since the particles are unevenly distributed across processors after parallel resampling. In order to avoid an unnecessarily large number of transfers in the routing step, particles and replication factors are grouped into messages, *packaging*, an activity which is integrated with the parallel resampling algorithm. The pattern of exchange between processors in particle routing is determined by a *scheduling (3)* algorithm. The output from scheduling is an *exchange matrix,* which carries the necessary information about exchange to the processes, i.e. information on the sender, receiver and the number of particles transferred.

Overall inter-processor communication in the parallel particle filter consists of two parts: particle routing, which is implemented using MPI point-to-point communication routines, and *collective communication*. In our framework scheduling and parallel resampling requires collective communication, which involves all P processes. The computation of estimates via aggregation of local results, e.g. the likelihood (and also the state filtering mean and covariance if these are of interest), also requires collective communication.

Under our assumptions, A1 and A2 above, it is shown that the minimal collective communication equals one *allgather* operation per iteration of the particle filter. Further reduction, or even complete elimination, of collective communication, unavoidably implies that the statistical performance of resampling, and hence particle filtering, is compromised (i.e. assumption A1 above is violated). Importantly, the computation of estimates does not increase the collective communication cost, beyond the communication cost already introduced by parallel resampling. Intuitively, conditional on resampling being applied, the computation of estimates is obtained largely 'for free', if messages are appropriately grouped.

The main contributions to the general literature on parallel particle filtering algorithms are the following. First, *routing of arbitrary particles* is suggested as the natural particle routing method, when the output from resampling are the replication factors, r, and the associated compact form, χ_c^{out}. This approach, which is used in PPF2, i) significantly lowers the total variable particle routing requirement, and ii) largely disconnects the routing requirement from the performance of the particle filter, as measured e.g. by the effective sample size, in [3.31]. Second, a parallel residual systematic resampling (RSR) algorithm is suggested, and it is contrasted with the parallel version of the RPA algorithm, proposed by Bolić et al. (2005). The former algorithm is equivalent to the SR/RSR algorithm, independent of the number of processors applied, whereas the latter is not. Third, the importance of the choice of resampling method for parallel efficiency is illustrated, by interpreting the PPF algorithms of Brun et al. (2002) and Teulière and Brun (2003) within our framework.

As a point of reference for the subsequent discussion a generic parallel particle filter algorithm is presented.

Algorithm 6 *Generic parallel particle filter*

1. **Sampling and importance steps**: Do steps 2-4 in Algorithm 1 in parallel. Each processor treats a block of $M = N/P$ particles.

2. **Interresampling**: Each process sends its partial sum of un-normalised weights,

$$\tilde{S}_{p,t|t-1}(\tilde{w}_p) = \iota_M^T \tilde{w}_{p,t|t-1} = \sum_{j=1}^{M} \tilde{w}_{t|t-1}^{j,p},\ p = 1, \ldots, P \tag{3.70}$$

   to the master process (*gather*). The master process performs interresampling to determine the process replication factor, $N_{p,t}$, of each process.

3. **Scheduling**: The master process calculates the exchange matrix, which determines how particles should be transferred between processes in the routing step (step 5), and sends it to the processes (*scatter*).

4. **Intraresampling**: The processes perform resampling in parallel, and each process resamples $N_{p,t}$ particles.

5. **Routing**: Particles are transferred between processes, such that load balance, i.e. M particles per process, is restored before the next iteration sampling step.⊡

We describe the two algorithms briefly, while noting that many of the concepts are explained later in this section. PPF1 is characterised by addressing without indices or replication factors, simple particle routing (SPR) and a distributed resampling with proportional allocation (DRPA) algorithm which produces indices, i (DRPA1). PPF2 features addressing with replication factors (and arranged indices), particle routing of arbitrary particles (AP), and a DRPA algorithm which produces replication factors, r (DRPA2). The implementation of interresampling, scheduling and collective communication is identical in the two algorithms. The algorithms are summarised in table 3.1, at the end of this section.

3.4.2 Parallel resampling

Parallel resampling algorithms

The parallel resampling algorithms presented in the parallel particle filtering literature can be divided into two broad categories, based on the implied statistical properties of the particle filter, and the communication properties of the parallel particle filter. The resampling algorithms in the first category imply that the statistical properties of the particle filter are independent of the number of processors applied with the parallel particle filter (PPF) algorithm, at the expense of introducing substantial inter-processor communication in the PPF. Distributed resampling with proportional allocation (DRPA), which is the method we use in this article, belongs to this category.

The DRPA algorithm is the direct parallel analogue of the RPA algorithm, algorithm 4, in section 3.3.2. In order to maximise the parallel efficiency of the

algorithm on P processors, it is natural to let $\tilde{P} = P$, i.e. to let each process handle one strata. As a consequence, the resampling method used in the parallel particle filter is determined by the number of processors, P, applied, see [3.58]. The $\text{RPA}_{\tilde{P}}$ algorithm applied on $\tilde{P} = P$ processors is denoted DRPA_P and,

$$RPA_P \Leftrightarrow DRPA_P, \tag{3.71}$$

i.e. the statistical properties of the RPA_P algorithm carry over directly to the DRPA_P algorithm. In principle, however, nothing prevents us from applying the $RPA_{\tilde{P}}$ using P processors, with, for example, $\tilde{P} = kP$ for some integer $k > 1$.

The resampling algorithms in the second category instead explicitly attempt to minimise inter-processor communication in the PPF. The drawback is that, in general, the statistical properties of resampling, and hence the statistical properties of the particle filter, are no longer independent of the number of processors applied with the filter. Examples in this category are the (i) distributed resampling with non-proportional allocation (DRNA), and (ii) local selection (LS), algorithms. An analysis of the statistical properties of the particle filter, when these two algorithms are used for resampling, is performed by Miguez (2007). The local parallel resampling algorithm of Brun et al. (2002), which may be interpreted as the limiting (no communication) case of the DRNA algorithm, yields an embarrassingly parallel PF, which is described and discussed briefly below.

The DRPA algorithm and the distributed residual systematic resampling (DRSR) algorithm

The DRPA_P algorithm is presented in the context of a parallel particle filter with a fixed number of particles, N, such that resampling is applied with $N_{in} = N_{out} = N$ particles. At the outset of resampling, each process, p, holds an $M \times 1$ vector of un-normalised weights,

$$\tilde{w}_p = \left(\tilde{w}_p^1 \ \tilde{w}_p^2, \ldots, \tilde{w}_p^M\right)^T, \ p = 1, \ldots, P, \tag{3.72}$$

which has been calculated in the importance step of the particle filter, and the full, $N \times 1$, vector of un-normalised weights is

$$\tilde{w} = \left(\tilde{w}_1^T \ \tilde{w}_2^T \ \ldots \tilde{w}_P^T\right)^T. \tag{3.73}$$

Prior to resampling, each process, p, calculates its local sum of un-normalised weights, $\tilde{S}_p = \tilde{w}_p^T \iota_M$. A master process collects the $P \times 1$ vector, containing the un-normalised process weights,

$$\tilde{s} = (\tilde{S}_1, \tilde{S}_2, \ldots, \tilde{S}_P)^T \tag{3.74}$$

(using message passing tokens a *gather* is performed). In the first stage of the DRPA algorithm, the master process performs *inter-resampling*, to obtain the $P \times 1$ vector of process replication factors, as in [3.53]. The process replication factors are returned to the processes (a *scatter*) and, in the second stage of the DRPA_P algorithm, the processes perform *intra-resampling* on their local set of particles.

The $M \times 1$ vector of *locally* normalised particle weights, for process p, is

$$w_p = \tilde{S}_p^{-1} \tilde{w}_p, \; p = 1, \ldots, P. \tag{3.75}$$

The output from intra-resampling is either an $M \times 1$ process vector of replication factors, r_p, or an $N_p \times 1$ process vector of particle indices, i_p. The DRPA$_P$ algorithm is summarised below. The uniform random numbers, used in the algorithm, are collected in the input vector $U = (u^*, u_1, \ldots, u_P)^T$.

Algorithm 7 *Distributed RPA:* $r/i = DRPA_P(\tilde{w}, N, N, U)$

1. At the outset of resampling each process holds a vector of un-normalised weights, [3.72].

2. The master process collects the process weights, $\tilde{s}$ in [3.74] (*gather*), and calculates $s = \tilde{S}^{-1}\tilde{s}$, where $\tilde{S} = \tilde{s}^T \iota_P$.

3. The master process performs interresampling to obtain the vector of process replication factors,
$$n = (N_1, N_2, \ldots, N_P)^T, \tag{3.76}$$
$$n = RSR(s, P, N, u^*), \text{ where } u^* \sim U(0,1). \tag{3.77}$$

4. The process replication factors, collected in n, are distributed to the processes (*scatter*).

5. Intra-resampling is performed in parallel by the processes to produce the process vector of replication factors,
$$r_p = RSR(w_p, M, N_p, u_p), \; p = 1, \ldots, P \; , \tag{3.78}$$
or the process index vector,
$$i_p = SR(w_p, M, N_p, u_p), \; p = 1, \ldots, P \; , \tag{3.79}$$
where $M = N/P$ is the number of input particles per process, N_p is the process replication factor, w_p is the vector of locally normalised weights in [3.75] and $u_p \sim U(0,1)$, $p = 1, \ldots, P$. $\boxdot$

The collective communication, in steps 2 and 4, may be arranged differently, when the DRPA$_P$ algorithm is employed within a parallel particle filter, see section 3.4.4. The normalisation of the input weight vectors, prior to the calls in [3.77], [3.78] and [3.79], can be avoided using result 5.

Two variants of the DRPA$_P$ algorithm, which differ only with respect to the final output format of resampling, are used in the parallel particle filters. *DRPA1*, which is used in PPF1, is described by [3.77] and [3.79]. *DRPA2*, which is used in PPF2, is described by [3.77] and [3.78]. The complexity of intraresampling is $O(\max_p N_p)$ and $O(M)$, respectively, and the theoretical speedup of the algorithms is $N/(\max_p N_p/P + P)$ and $N/(N/P + P)$, respectively.

Distributed RSR

A parallel resampling algorithm, *distributed RSR (DRSR)*, which produces replication factors, and which is strictly statistically equivalent to the SR and RSR algorithms, for all values of the processors, P,

$$DRSR_P \Leftrightarrow SR, \tag{3.80}$$

is constructed in appendix A. To our knowledge the DRSR algorithm is the first parallel resampling algorithm with the property that the resampling output does not depend on the number of processors, P, applied with the algorithm. The computational cost of the DRSR algorithm is largely equivalent to the cost of the DRPA2 algorithm, and either of these algorithms could be used with PPF2.

Local resampling and multiple particle filters

The *local parallel resampling algorithm*, presented by Brun et al. (2002), is interpreted in our framework as consisting only of the intra-resampling step, i.e.

$$N_p = M,\ p = 1, \ldots, P, \tag{3.81}$$

is imposed, and the replication factors or indices are obtained through

$$r_p/i_p = SR(w_p, M, M, u_p),\ \ p = 1, \ldots, P. \tag{3.82}$$

This yields a parallel particle filter with no communication requirement, except for post-filtering communication, when local results need to be collected and coalesced, e.g. for the purpose of calculating the likelihood. Applying the PPF with local resampling is equivalent to running P separate particle filters, with M particles each, in parallel, and aggregating the results. This is an embarrassingly parallel approach, and we refer to it as the approach of *multiple particle filters* (MPF), to distinguish it from a 'truly' parallel particle filter, which requires inter-processor communication at each iteration of the filter. The theoretical speedup of the local parallel resampling algorithm, in [3.81] and [3.82], as well as of the overall multiple particle filters algorithm, is equal to P. Each process independently produces an estimate of the likelihood, in [3.15], $\hat{L}^p$, and the P estimates can be combined to a single, unbiased, estimate via averaging

$$\hat{L} = P^{-1} \sum_{i=1}^{P} \hat{L}^p. \tag{3.83}$$

Combining the output from P particle filters, with M particles each, in this way, is thus equivalent to applying a single particle filter, with $N = MP$ particles, while using the particular parallel resampling algorithm, described by [3.81] and [3.82]. Again, this is a resampling method which would appear un-natural in a serial computing context.

It is beyond the scope of this article to perform a full comparison of the various parallel resampling algorithms and their effects on the statistical properties of the

particle filter. Using simulations we have, however, verified that the variance of the averaged likelihood estimate, in [3.83], obtained using multiple particle filters, is significantly larger than the variance of the likelihood estimate resulting from the PPF based on the DRPA algorithm. The simulations have been performed using the DSGE model presented later, in section 3.5, and with $N = 40k$ particles, $P = 40$ processors, and hence $M = 1k$ particles per particle filter, and process.

3.4.3 Particle routing

Particle imbalance

After resampling the surviving particles are unevenly distributed across processes; each process is left with N_p particles, of equal weight. *Particle routing*, the transfer (i.e. sending and receiving) of particles between processes, is necessary to restore load balance, $M = N/P$ particles per process, before the subsequent (next iteration) sampling and importance (SIS) step. In the absence of routing the process weights, $\tilde{S}_p$, would degenerate and eventually one process would perform the SIS step on the full set of particles, while the other processes would be idle. We emphasize that, in our context, particle routing is performed solely to increase the computational efficiency of the PPF, i.e. to obtain load balance in the SIS step. By contrast, a PPF which uses the DRNA algorithm for resampling applies routing to increase the statistical efficiency of filtering, see e.g. Miguez (2007).

In this section three particle routing methods are presented. The characteristics of particle routing derives from the output format of resampling, which is either the indices, i, and the standard representation, χ^{out} in [3.29], or replication factors, r, and the compact representation, χ_c^{out}, in [3.42]. The main idea here is that usage of the latter representation, r and χ_c^{out}, (PPF2) makes it possible to reduce the particle routing communication requirement, i.e. the total number of particles (or bytes), transferred through the network, in comparison with a PPF which uses the former representation, i and χ^{out} (PPF1). A key issue in the comparison of the two algorithms is to assess whether this reduction of the communication requirement also pays off in terms of computational time.

When the *DRPA1* algorithm is used for resampling, to produce the process index vector, i_p, in [3.79], the sample of particles on a processor after resampling is described by

$$\chi^{p,out} = \{X^{p,i}\}_{i=1}^{N_p},\ p = 1, \ldots, P, \tag{3.84}$$

and the complete sample of particles, first given in [3.29], is written as

$$\chi^{out} = \cup_{p=1}^{P} \chi^{p,out}, \tag{3.85}$$

where we use the set notation introduced by Miguez (2007). The number of unique particles in $\chi^{p,out}$ is $N_{U,p} \leq N_p$, such that the total number of unique particles in χ^{out} is

$$N_U = \sum_{p=1}^{P} N_{U,p} \leq N.$$

If, instead, the *DRPA2* algorithm is used for resampling, to produce the process vector of replication factors, r_p, in [3.78], the output sample on a processor is described in the compact form

$$\chi_c^{p,out} = \{(X_c^{p,i}, r_c^{p,i})\}_{i=1}^{N_{U,p,}},\ p = 1, \ldots, P, \tag{3.86}$$

where $r_c^{p,i} \geq 1$, and the complete output sample of particles, first given in [3.42], is

$$\chi_c^{out} = \cup_{p=1}^{P} \chi_c^{p,out}. \tag{3.87}$$

The total number of particles which need to be exchanged in a given iteration of the particle filter, with reference to the representation [3.84] and [3.85], is given by

$$K = \sum_{p:N_p > M} K_p = -\sum_{p:N_p < M} K_p \leq \frac{(P-1)}{P} N = N - M, \tag{3.88}$$

where

$$K_p = N_p - M, \tag{3.89}$$

is the *particle surplus* of process p. A process is called a *sender* if $K_p > 0$, or a *receiver*, if it has a deficit of particles, $K_p \leq 0$. The worst case occurs when $N_p = N$ for some process, p, such that $K = N - M$. The particle surplus, K_p, and the total surplus of sending processes (or alternatively, the total deficit of receiving processes), K, are functions of the vector of particle weights, w, the number of particles, N, and the number of processors, P, i.e.

$$K = K(w, N, P).$$

The relationship between K, N and P is characterised below.

Scheduling

The pattern and quantities of particle exchange between the processes, in routing, is determined by a *scheduling* algorithm. An algorithm is presented by Teulière and Brun (2003) (algorithm 1, p.1080) and a similar method is used in our implementations. The input to scheduling are the processor replication factors, collected in the vector n, which are available after inter-resampling, [3.77]. In particle routing a process, p, must know its counterpart, $\tilde{p}$, in the transfer, and the number of particles to be exchanged between the two processes, $K_{p,\tilde{p}}$, again with reference to the representation [3.84] and [3.85]. The output from scheduling specifies the pattern and quantities of exchange in particle routing, e.g. in the form of a $P \times P$, skew-symmetric, *exchange matrix*: $E = \{K_{p,\tilde{p}}\}$ where $K_{p,\tilde{p}} = -K_{\tilde{p},p}$. The fixed cost, or latency, of particle routing is determined by the non-zero pattern of E.

The scheduling algorithm used here has three key properties. First, the obvious 'load balancing property',

$$\sum_{\tilde{p}} K_{p,\tilde{p}} = K_p,\ p = 1, \ldots, P, \tag{3.90}$$

or,

$$E\iota_P = n - M\iota_P. \tag{3.91}$$

Second, it is assumed that a process cannot both send and receive particles in the routing step, e.g. for sending processes,

$$K_p \geq 0 \Longrightarrow K_{p,\tilde{p}} \geq 0, \text{ for all } \tilde{p}, \tag{3.92}$$

Third, after a transfer has been completed, one of the involved processes is left with the desired number of particles, M. Therefore, the maximum number of transfers in routing equals $P-1$, and the maximum number of non-zeros in E is equal to $2(P-1)$,

$$\sum_p \sum_{\tilde{p}} I(K_{p,\tilde{p}} \neq 0) \leq 2(P-1). \tag{3.93}$$

The number of transfers is lower than $P-1$ whenever $K_p = N_p - M = 0$ for some process, p, an event which occurs with a very low probability in practise. The worst case, in terms of the overall latency of routing, occurs when there is a single sender (or receiver), which needs to be involved in all $P-1$ transfers, i.e. when all elements in a row of E are non-zeros.

To summarise, our scheduling algorithm defines a mapping,

$$E = f(n), \tag{3.94}$$

which satisfies the restrictions [3.90], [3.92] and [3.93].

The randomness of the process sum of weights, $\tilde{S}_p(\tilde{w}_p)$, implies that N_p, and hence also $K_{p,\tilde{p}}$, K_p and K, are stochastic variables. As a consequence the sender/receiver status of a process, and hence the communication pattern in routing, changes between iterations of the PF.

Particle routing methods

There are two ways to transfer a single unique particle, X_c^m, with replication factor, $r_c^m > 0$, between a pair of processes:

- If particles are represented using indices (χ^{out}), r_c^m copies of the unique particle are transferred. (PPF1)
- If particles are represented using replication factors and arranged indices (χ_c^{out}), the unique particle and its replication factor are transferred. (PPF2)

The number of *particles routed*, which is a property of the routing method, is r_c^m and 1, respectively. Alternatively, the number of elements routed is $r_c^m \times n_X$ and $n_X + 1$, respectively, where a particle is represented by an $n_X \times 1$ vector. The total number of bytes routed is obviously a linear function of the number of particles routed.

We first consider the case where the output particles from resampling are represented as in [3.84] and [3.85], which leads to *simple particle routing* (SPR),

the method used in PPF1. The sample of particles of a *sending process*, p, may be decomposed as

$$\chi^{p,out} = \{X^{p,i}\}_{i=1}^{M} \cup \{X^{p,i}\}_{i=M+1}^{M+K_p} = \chi_M^{p,out} \cup \chi_{K_p}^{p,out}, \tag{3.95}$$

where $\chi_{K_p}^{p,out}$ contains the $K_p > 0$ surplus particles. In our implementation of PPF1 the $N_p = M + K_p$ output particles are stored in an $n_X \times (M + K_p)$ matrix

$$M_{out}^p = \left[\begin{array}{cc} M_{out,M}^p & M_{out,K_p}^p \end{array} \right], \tag{3.96}$$

corresponding to [3.95]. The $n_X \times K_p$ right matrix, M_{out,K_p}^p, is sliced into submatrices, which are sent to the counterpart receiving processes, in accordance with the information contained in row p of the exchange matrix, E. The number of particles routed per message is $R_{p,\tilde{p}}^{SPR} = K_{p,\tilde{p}}$ and the total number of particles routed by the sending process, p, is $R_p^{SPR} = K_p$. The SPR approach is apparently wasteful, since a message will typically consist of a matrix with several identical columns, corresponding to copies of a single unique particle.

Next, assume that the resampling algorithm outputs replication factors, such that the output sample of particles is represented as in [3.86] and [3.87]. Consider again a sending process, p, which is to send a total of K_p particles to its set of counterpart processes. The unique particles selected for transfer, from $\chi_c^{p,out}$, to the receiving process, $\tilde{p}$, has the property that their sum of replication factors equals $K_{p,\tilde{p}}$, i.e. particles are selected subject to a stopping rule. The number of particles routed between the sender p and the receiver $\tilde{p}$, $R_{p,\tilde{p}} > 0$, satisfies $R_{p,\tilde{p}} \leq K_{p,\tilde{p}}$. The total number of particles routed is

$$R = \sum_{p:R_p>0} R_p \leq K, \text{ where } 0 \leq R_p = \sum_{\tilde{p}\neq p} R_{p,\tilde{p}} \leq K_p, \tag{3.97}$$

is the number of particles routed by a sending process, p.

In this case, and in contrast to SPR, the selection of the particles for transfer in $\chi_c^{p,out}$ affects the number of particles routed, $R_{p,\tilde{p}}$, R_p and R. One possibility is to order the particles in $\chi_c^{p,out}$ according to their replication factors and transfer the particles with the largest replication factors. By construction this approach minimises the number of particles routed, i.e. $R^{LWP} = R_{\min}$, and we call it *routing of largest weight particles* (LWP).

The computational cost of sorting the particles in $\chi_c^{p,out}$, according to their replication factors, however, is prohibitive. The natural approach is instead to select the particles without consideration of their replication factors, e.g. in list order, such that the messages can be constructed quickly. In constructing its first message, which is received by process $\tilde{p}$, the sender p collects $R_{p,\tilde{p}}^{AP}$ unique particles from $\chi_c^{p,out}$, in [3.86], where $R_{p,\tilde{p}}^{AP}$ is the lowest value satisfying

$$\sum_{m=1}^{R_{p,\tilde{p}}^{AP}} r_c^{p,m} \geq K_{p,\tilde{p}}. \tag{3.98}$$

The construction of the second message/matrix begins where the first ended, and so on. We call this approach *routing of arbitrary particles* (AP), and it is the method used in PPF2.

The index of a particle in $\chi_c^{p,out}$, m, is independent of its replication factor, $r_c^{p,m}$. This implies that the expected average replication factor for the R^{AP} unique particles, which are selected for transfer, is equal to the overall mean of the replication factors in χ_c^{out}, which is

$$\bar{r}_c = \frac{1}{N_U}\sum_{m=1}^{N_U} r_c^m = \frac{N}{N_U} \geq 1. \tag{3.99}$$

To summarise, the total number of particles routed in an iteration of the PF, R, with the three methods presented above, thus satisfies

$$R_{\min} = R^{LWP} \leq R^{AP} \leq R^{SPR} = K \leq N - M, \tag{3.100}$$

where the methods are equivalent when $r^m = r_c^m = 1$ for all particles.

An example is provided to illustrate the three methods.

Example 8 *Let $P = 2$, $N_1 = 12$, $N_2 = 4$, $N = N_1 + N_2 = 16$, $M = N/P = 8$ and assume the following output sets of particles on the two processors*

$$\chi^{1,out} = \left\{X^1, X^3, X^3, X^3, X^3, X^5, X^5, X^8, X^9, X^9, X^{\,9}, X^{11}\right\},$$

and

$$\chi^{2,out} = \left\{X^{12}, X^{12}, X^{15}, X^{15}\right\},$$

such that $N_U = 8$. This may, alternatively, be represented as

$$\chi_c^{1,out} = \left\{(X^1,1), (X^3,4), (X^5,2), (X^8,1), (X^9,3), (X^{11},1)\right\},$$

and

$$\chi_c^{2,out} = \left\{(X^{12},2), (X^{15},2)\right\}.$$

The exchange matrix is

$$E = \begin{pmatrix} 0 & K_{1,2} \\ K_{2,1} & 0 \end{pmatrix} = \begin{pmatrix} 0 & 4 \\ -4 & 0 \end{pmatrix}.$$

In simple particle routing, SPR, $\left\{X^9, X^9, X^{\,9}, X^{11}\right\}$ is transferred from the sending processor, 1, to the receiving processor, 2. Using routing of arbitrary particles, AP, $\left\{(X^1,1), (X^3,3)\right\}$ is transferred. While the replication factor of $X_c^{1,2} = X^3$ is equal to $r_c^{1,2} = 4$, the transferred replication factor is adjusted to 3, such that the transferred replication factors add to $N_1 - M = 12 - 8 = 4$. In these two cases, SPR and AP, it is clearly possible to select the particles for transfer in other, equally simple, ways. Routing of largest weight particles implies that $\{(X^3,4)\}$ is transferred. The number of particles routed in these three cases are $R_{1,2}^{SPR} = 4$, $R_{1,2}^{AP} = 2$ and $R_{1,2}^{LWP} = 1$, respectively.⊡

The implementation of AP (and LWP) raises additional, practical, issues, which are just briefly mentioned here. The construction of the routing messages on sending processors, an activity we call *packaging*, is integrated with the RSR intra-resampling procedure. The $R_{p,\tilde{p}}$ unique particles (reals) and their replication factors (integers), which are transferred from p to $\tilde{p}$, must be contained in a single message/matrix, such that the total number of messages in particle routing remains equal to $P-1$. By contrast, in the serial ($P=1$) case, the particle memory is accessed only in the SIS step (see section 3.3.3). Further, the receiving processes do not know the size, $R_{p,\tilde{p}}$, of an incoming message, and this must be handled by the *receive* call. We note that these issues entail additional costs of AP, which must be weighted against the gains of reduced communication.

Particle routing time

The overall communication cost, i.e. the wall-clock time of communication, of a parallel algorithm may, in general, be divided into two components: a fixed cost, or *latency*, term and a variable cost, or *bandwith*, term. The latency of particle routing is determined by the communication pattern, which is captured by the non-zero structure of the exchange matrix, E. This cost is therefore identical for the three particle routing methods, since the exchange matrix is identical in all three cases. Assuming a fully connected network, and neglecting complex queuing effects associated with the scheduling algorithm, the latency of particle routing is taken to be proportional to the maximum number of transfers of any process, i.e. it is equal to the maximum number of non-zeros in a row of the exchange matrix, E.

The difference in variable routing cost between the methods, on the other hand, depends on the method of routing, and on the type of interconnection network. In appendix B a stylised parallel communication model is presented, in order to discuss relevant proxies for particle routing time, using two assumptions on the interconnection network. The relevant proxies, for a single bus type of network and a fully connected network, are R and $R_{p,\max}$, respectively, where $R_{p,\max}$ is the maximum number of particles transferred (sent or received) by any process in routing. Intuitively, in the $R_{p,\max}-$model of the variable cost of particle routing, the time is determined by the slowest process, i.e. the process which sends or receives the largest number of particles.

In appendix B we include two simulation exercises to illustrate the relations between N, N_{eff}, P, $R^{SPR}=K$, R^{AP}, $R^{SPR}_{p,\max}$ and $R^{AP}_{p,\max}$. Three results/predictions from these simulations are worth highlighting here:

1. The particle routing time with AP should be largely independent of the statistical performance of the PF, as measured by the effective sample size, N_{eff}, while holding N and P constant. The particle routing time with SPR should be decreasing in N_{eff}.

2. The particle surplus, K, is linear in N, when $M=N/P$ and N_{eff}/N are kept constant.

3. Assuming a fully connected network the simulated latency of routing is very similar to the latency of collective communication (see section 3.4.4), i.e. it is roughly proportional to $\log_2 P$.

3.4.4 Collective communication

Implementation of collective communication

Particle routing is performed using point-to-point communication between processes. The remaining communication in the parallel particle filter is implemented using MPI collective communication routines (collectives). The collectives, which are discussed in this section, are briefly explained in appendix C.

The collective communication in a parallel particle filter, which uses the DRPA algorithm for resampling, and where resampling is performed at each iteration of the filter, consists of three parts. First, the process weights, $\tilde{S}_p$, in [3.74], are collected by a master process (*gather*), which then performs inter-resampling, [3.77], followed by scheduling, [3.94]. Second, the results from these serial activities, i.e. the process replication factors, N_p, in [3.76], and the rows of the exchange matrix, E, must be transmitted to the processes (*scatter*), in order for these to perform intra-resampling, [3.78], and particle routing. This is described in the generic PPF algorithm (algorithm 6). Note that it is not necessary to separately transfer the process replication factor, N_p, of process p, since it can be inferred from row p of the exchange matrix, using the relation [3.91]. Finally, the calculation of estimates, e.g. of the likelihood, requires that local results are collected (*gather*). The parallel computation of estimates of the state filtering mean and covariance, and the likelihood, is described in appendix D.

The *gather-scatter* is, perhaps, the most natural implementation of collective communication in the PPF. In appendix C two other variants of implementation are described: *gather-broadcast* and *allgather*. The latter implementation essentially implies that all processes obtain the status of master process, which, in our view, simplifies the implementation of the PPF. The minimum latency of collective communication, using the *allgather*, is proportional to $\log_2 P$, while the variable cost may be disregarded, see appendix C.

Collective communication with conditional resampling

The use of a conditional resampling scheme (see section 3.3), in conjunction with the DRPA algorithm, does not alter the collective communication cost. In this case, the process sum of weights, $\tilde{S}_p$, and the process sum of squared un-normalised weights,

$$\tilde{Q}_p = \tilde{w}_p^T \tilde{w}_p,\ p = 1, \ldots, P, \tag{3.101}$$

are jointly collected (*gather*), since the effective sample size, N_{eff}, in [3.31], needs to be computed. The comparison of N_{eff} with the threshold value, N_{thres}, yields a decision on whether to resample or not, which is transmitted to the processes (*broadcast*). The computational gains from conditional resampling occurs whenever

Table 3.1 Description of the parallel particle filter algorithms.

Serial/parallel(S/P)	S/P	PPF1	PPF2	Section
Sampling/Importance	P	'Simple' loop.	'Complex' loop.	3.3.3, 3.4.1
Resampling method	-	Prop. alloc. (RPA)	RPA	3.3.2
Parallel resampling alg.	S/P	DRPA1	DRPA2	3.4.2
Interresampling	S	Res. sys. res. (RSR)	RSR	3.3.2, 3.4.2
Intraresampling	P	Sys res. (SR)	RSR	3.3.2, 3.4.2
Resampling output	-	Indices, i, χ^{out}	Rep. fac., r, χ_c^{out}	3.3.2, 3.3.3
Scheduling	S	-	-	3.4.3
Particle routing	P	Simple (SPR)	Arb. part. (AP)	3.4.3
Collective comm.	P	Allgather	Allgather	3.4.4

$N_{eff} > N_{thres}$, since resampling, including scheduling and particle routing, is avoided. Again, the *allgather* can be used to implement the collective communication.

The collective communication cost in the PPF with conditional resampling, and with the 'proportional redistribution' method used for resampling, presented by Teulière and Brun (2003), is proportional to $4 \log_2 P$ (two *gather* and two *scatter*), in iterations where it is decided that resampling needs to be performed, and $2 \log_2 P$ (one *gather* and one *scatter*), when resampling is not performed. In order to reach the minimum collective communication, the parallel resampling method must have the property that the vector of un-normalised process weights, $\tilde{s}$, is sufficient for calculation of the vector of process replication factors, n. The proportional redistribution method for resampling does not satisfy this requirement, which explains the excessive collective communication cost in their algorithm. Furthermore this resampling method violates the proper weighting condition, [3.43].

In table 3.1 we summarise the main features of the two parallel particle filter algorithms, PPF1 and PPF2.

3.5 Computational experiment

3.5.1 Setup

DSGE model

The model used in the experiments is a standard small-scale New Keynesian (NK) model (An (2005)). The main feature of the model is price stickiness, modelled through quadratic adjustment costs in firm's price-setting. The second order approximate policy function of the model is represented in the form [3.3] and [3.13]. The model contains $n_{x_1} = 3$ shocks, and in total there are $n_X = 7$ state variables. There are $n_y = 3$ observed variables, and data for $T = 100$ time periods is simulated from the nonlinear model using a 'standard' calibration of the model. The vector of estimated parameters, θ, has dimension 13×1. The model and the calibration, used

to generate data, is described in appendix E.[4]

In estimating DSGE models using the particle filter, it is common to calibrate the measurement error variances, i.e. the elements of the diagonal covariance matrix, Σ_v (see section 3.2). The performance of the particle filter is quite sensitive to this calibration. A peaked observation error density, i.e. small variances of the measurement errors, implies that the efficiency of the standard importance density, in [3.26], is reduced, which is manifested as a small effective sample size, [3.31]. Here we exploit this feature and scale Σ_v to tune the particle routing requirement, R, in the parallel particle filter. In the *benchmark model*, Σ_v is calibrated, such that on average the effective sample size, as a fraction of the number of particles, N_{eff}/N, is roughly 10% (where the average is calculated over the H draws of the sampler, and T iterations of the filter).

In addition, an *extreme model* case is constructed, where N_{eff}/N is roughly 1.5% on average. This signals a poor performance of the particle filter: the variance of the likelihood estimate, in [3.36], is large, and the acceptance rate of the random walk Metropolis algorithm is lower than in the benchmark case, roughly 13% versus 24%. The purpose of the latter calibration is to study the performance of the PPF, and in particular simple particle routing (SPR) with PPF1, in the worst possible situation, from a computational point of view. Estimating the extreme model with a large number of particles, $N = 640k$, while using up to $P = 128$ processors, implies that the PPFs are tested in a setting which is much more demanding than what we expect in 'normal' applied work with a model of this size.

Hardware

The parallel particle filters are tested on three computing clusters, at the Center for High Performance Computing (PDC), at the Royal Institute of Technology in Stockholm. The *Lucidor* cluster has two HP Itanium 2, 900 Mhz, CPUs per computing node and the nodes are connected with a Myrinet 248 Mb network. The *Lenngren* cluster has two Intel Xeon, 3.4 Gb, CPUs per node, and a 1 Gb Infiniband network. On these two clusters we always use both processors on a node, in our experiments. Finally, the *Ferlin* cluster has two quad-core Intel Harpertown, 2.66 GHz, CPUs per node, and a complex 'blend' network, which is not described here since we will only consider one-node runs, with $P \leq 8$ processor cores, on this cluster. More detailed information about the clusters is available at http://www.pdc.kth.se[5].

Most of the results presented below are based on runs at the Lenngren cluster. The processor/network balance on Lucidor and Lenngren appears to be quite similar, while the latter cluster has a better absolute performance. The parallel efficiency results on these two clusters are therefore fairly similar for our implementations. The Ferlin multi-core cluster, which is the newest of the clusters, is explicitly intended

[4] We have also experimented with the model presented by Erceg, Henderson and Levin (2000), but since it is of roughly the same size as the NK model the results are very similar, and they are therefore not reported here.

[5] PPF1 was one of the benchmark parallel algorithms in the process that led to the purchase of the Ferlin cluster in 2008. The Lucidor and Lenngren clusters retired in 2009.

Table 3.2 Dimensions of the experiment.

Algorithms	PPF1 and PPF2
Particles	$N = 40k$, $160k$, $640k$
Processors	$P = 1, 2, 4, 8, 16, 32, 64, 128$
Clusters	Lenngren, Lucidor, Ferlin
Model	Benchmark and extreme parameterisation

for one-node jobs. It is mainly suitable for applications which require modest inter-processor communication, e.g. embarrassingly parallel applications, and the PPFs are not highly scalable on this type of cluster. The multi-core computer, however, is becoming a popular personal high performance computing (PHPC) desktop machine, which, in our view, makes it interesting to include this computer (i.e. a single node on the cluster) in the comparison.

Software

The PPFs are implemented using Fortran and the Message Passing Interface (MPI). The MPI implementations used are Mpich, on the Lucidor and Ferlin clusters, and ScaliMPI, on Lenngren. The programs are compiled with the Intel Visual Fortran (IVF) Compiler, versions 10 or 11, and use the Intel Math Kernel Library (IMKL) versions 9 or 10. Fine-level optimisation of the code, including the selection of compiler options, has been performed mainly via experimentation on the Lenngren cluster.

Measurement

The scalability of the parallel particle filters is measured using the *relative efficiency*,

$$\tau(P) = \frac{T(1)}{PT(P)} = \frac{S(P)}{P}, \tag{3.102}$$

where $T(P)$ is the wall-clock time of running the parallel program using P processors, and $S(P)$ is the relative speedup, defined in [3.69].

The PPFs are timed when they are applied inside the random walk Metropolis algorithm, to estimate the DSGE model. Typically a fairly small number of draws, $H = 1k - 5k$, are obtained and occasionally we use longer chains, $H = 100k$ or $200k$, to verify that the timing results from the shorter chains are not 'unrepresentative' in any sense. This strategy is motivated by the large number of runs, where the dimensions of the experiment are summarised in table 3.2. Each run is repeated at least twice.

The time of random number generation (RNG) is not included in the reported timing results. Common random numbers (CRN) are used for the particle filter, as suggested by Fernández-Villaverde (2009)[6].

[6]In our setting, using standard random number generators to generate a unique sequence of random numbers, for each PPF likelihood evaluation, the execution time is almost doubled. As a

Table 3.3 Accuracy of the log likelihood estimate for the NK model. Mean and standard deviation for 1,000 likelihood evaluations.

	Number of processors							
	1	2	4	8	16	32	64	128
Log likelihood eval. using PPF1								
M	-458.68	-458.70	-458.70	-458.68	-458.68	-458.68	-458.69	-458.68
Sd	0.25	0.26	0.24	0.26	0.24	0.27	0.26	0.26
Log Likelihood eval. using PPF2								
M	-458.67	-458.71	-458.70	-458.70	-458.69	-458.70	-458.70	-458.69
Sd	0.25	0.25	0.25	0.25	0.25	0.27	0.25	0.26

3.5.2 Likelihood evaluation

In this subsection the PPFs are first applied for likelihood evaluation at a single parameter point, which is the parameter vector used to generate the data, θ_{dgp}. The PPFs are applied with $N = 40k$ particles and $P = 1, 2, \ldots, 128$ processors and in each case the likelihood is evaluated $1k$ times and the mean and standard deviation of the log likelihood is calculated. The purpose of this exercise is to validate that the DRPA$_P$ resampling procedure, which depends on P, does not affect the performance of the particle filter, in any important sense. The results, presented in table 3.3, verify that the standard deviation of the log likelihood estimate does not depend on the number of processors applied.

3.5.3 Execution time and scalability

In table 3.4 execution times and relative efficiencies, for the two parallel particle filters, PPF1 and PPF2, are presented, for the benchmark model, on the Lenngren cluster. The execution times are relevant for a comparison of the two PPFs, whereas the relative efficiency, defined in [3.102], measures the scalability of each algorithm, in isolation. Results are presented for $N = 40k$, $160k$ and $640k$ particles, and $P = 1, 2, 4, \ldots, 128$ processors.

Applying a small number of processors, PPF2 is faster than PPF1. This is mainly explained by faster intra-resampling with the former algorithm. The speed difference vanishes as P increases. With $N = 40k$ particles, the speed gap is roughly closed when $P = 16$ processors are applied, and with $160k$ and $640k$ particles, the gap is closed when $P = 32$ and $P = 128$, respectively. These results suggest that, as P increases, the execution time is dominated by parts of the PPF algorithms, which are common to the two PPFs.

In table 3.5 execution times and relative efficiencies of the PPFs, are presented, for the extreme model, on the Lenngren cluster. In this case the speed gap between the two filters is significant also for large P, which suggests that particle routing accounts for a larger share of the execution time in PPF1, in comparison with the benchmark

rough rule-of-thumb, if RNG time is included, the relative efficiency reported for P processors below instead holds for $2P$ processors.

case. Comparing the results in tables 3.4 and 3.5, it is seen that the performance of PPF2 is roughly similar in the benchmark and extreme cases.

A lowest acceptable relative efficiency in 'production runs' is, perhaps, somewhere in the interval $70 - 80\%$, which can be motivated by considering an alternative parallelisation scheme for the RWM sampler (Strid (2010)). The implied maximum numbers of processors to use with $40k$, $160k$ and $640k$ particles, respectively, then are 32, 64 and 128. The range of absolute execution times per likelihood evaluation in these cases are $56 - 67$ ms, $115 - 133$ ms and $242 - 311$ ms, where the high value is the time of PPF1 in the extreme case. These numbers summarise the absolute computational performance that can be achieved using parallel methods on this estimation problem.

To put these numbers into perspective, we have implemented the serial particle filter using Matlab, a programming language which is widely used by macroeconomists. On a standard desktop computer, the efficient Matlab implementation of the particle filter, which uses two Fortran mex routines, executes in roughly 2 seconds, with $N = 40k$ particles.[7]

The results on the Lucidor cluster, for the benchmark case, are quite similar to those on Lenngren, i.e. the speed difference between the two filters are only modestly in favour of PPF2, and these are therefore not reported here. On the Ferlin cluster, however, the results are different, which is seen in table 3.6. Results are presented for $N = 40k$ and $160k$ particles, and $P = 1, 2, 4$, and $P = 8$ processor cores, on a single node. On this cluster, PPF1 generally performs better than PPF2, but in the relevant case, i.e. when all 8 processor cores are utilised, the performance difference is negligible. The execution time with $N = 40k$ particles is roughly $0.14s$ in all cases and the relative efficiency is above 80%. We have verified that the execution time with $P = 16$ processor cores, i.e. the full usage of two nodes, is actually *much larger* than in the $P = 8$ case, which simply illustrates that the Ferlin cluster is not designed for the application of communication intensive programs, on several nodes.

3.5.4 Profiling

The purpose of code profiling is to further understand how the PPF algorithms behave in practise, and, in particular, to identify the parts of the algorithm, which are the main contributors to the drop in parallel efficiency, as P increases[8]. In the previous section we already provided some remarks based on profiling to aid intuition. The PPF algorithm is divided into five sections: a) sampling and importance sampling (SIS),

[7]The Matlab implementation uses a block implementation of the sampling and importance (SIS) step. Mex functions are used for systematic resampling, algorithm 2, and the vech operator, in [3.8].

The computer is an Intel Quad Core, 2.67Ghz, 8Gb RAM. We note that the parallel particle filter cannot be successfully applied, using Matlab's parallel functionality, since it is a communication intensive application.

[8]Profiling of the programs is performed using instrumentation and the insertion of artificial barriers in the code, which increases the overall execution time to some extent. Therefore, separate runs are performed to profile the programs.

Table 3.4 Execution times and relative efficiency for NK model likelihood evaluation, on Lenngren. Benchmark case.

	Proc. P	1	2	4	8	16	32	64	128
N=40k									
PPF1	Time (ms)	1540	805	389	191	96	56	50	64
	RE (%)	100.0	95.7	99.0	100.8	100.3	85.9	48.1	18.8
PPF2	Time (ms)	1455	735	370	185	99	58	51	64
	RE (%)	100.0	99.0	98.3	98.3	91.9	78.4	44.6	17.8
N=160k									
PPF1	Time (ms)	6,242	3,119	1,611	805	405	201	124	97
	RE (%)	100.0	100.1	96.9	96.9	96.3	97.0	78.7	50.3
PPF2	Time (ms)	5,959	2,971	1,487	792	383	196	121	101
	RE (%)	100.0	100.3	100.2	94.0	97.2	95.0	76.9	46.1
N=640k									
PPF1	Time (ms)	25,375	12,659	6,586	3,146	1,634	811	450	247
	RE (%)	100.0	100.2	96.3	100.8	97.1	97.8	88.1	80.3
PPF2	Time (ms)	24,823	12,463	6,186	2,991	1,555	779	418	244
	RE (%)	100.0	99.6	100.3	103.7	99.8	99.6	92.8	79.5

Table 3.5 Execution times and relative efficiency for NK model likelihood evaluation, on Lenngren. Extreme case.

	Proc. P	1	2	4	8	16	32	64	128
N=40k									
PPF1	Time (ms)	1,512	777	394	196	111	67	58	80
	RE (%)	100.0	97.3	95.9	96.4	85.1	70.5	40.7	14.8
PPF2	Time (ms)	1,389	750	378	189	93	61	49	65
	RE (%)	100.0	92.6	91.9	91.9	93.3	71.2	44.3	16.7
N=160k									
PPF1	Time (ms)	6,235	3,138	1,516	791	426	232	133	124
	RE (%)	100.0	99.3	102.8	98.5	91.5	84.0	73.2	39.3
PPF2	Time (ms)	5,676	3,021	1,432	770	392	205	115	100
	RE (%)	100.0	93.9	99.1	92.1	90.5	86.5	77.3	44.3
N=640k									
PPF1	Time (ms)	24,461	12,661	6,139	3,152	1,636	858	497	311
	RE (%)	100.0	96.6	99.6	97.0	93.4	89.1	76.9	61.4
PPF2	Time (ms)	22,850	12,031	6,152	2,974	1,561	782	424	242
	RE (%)	100.0	95.0	92.9	96.0	91.5	91.3	84.2	73.8

Table 3.6 Execution times and relative efficiency for NK model likelihood evaluation, on the Ferlin cluster. Benchmark case.

		Benchmark				Extreme			
	Proc.	1	2	4	8	1	2	4	8
N=40k									
PPF1	Time (ms)	890	454	233	136	899	454	237	139
	RE (%)	100.0	98.0	95.5	81.8	100.0	99.0	94.8	80.8
PPF2	Time (ms)	996	495	256	143	974	492	254	142
	RE (%)	100.0	100.6	97.3	87.1	100.0	99.0	95.9	85.7
N=160k									
PPF1	Time (ms)	3835	1968	910	493	3889	2013	1020	519
	RE (%)	100.0	97.4	105.4	97.2	100.0	96.6	95.3	93.7
PPF2	Time (ms)	4190	2071	994	514	4064	2045	988	505
	RE (%)	100.0	101.2	105.4	101.9	100.0	99.4	102.8	100.6

b) inter-resampling and scheduling, c) collective communication, d) intra-resampling and e) particle routing.

Profiling shows that the speedup of the SIS step is roughly linear, which is expected. Inter-resampling and scheduling, together account for a negligible share of computing time, maximally 9% of time, which occurs in the very fine-grained case, when $P = 64$ and $N = 40k$. The time of intra-resampling converges for the two PPFs, as P increases, since packaging of particles for transfer in PPF2 accounts for a larger share of intra-resampling. Our detailed analysis in this section focuses exclusively on the behaviour of the inter-processor communication parts of the PPFs, c and e, since these are the most important parts for understanding parallel efficiency.

In table 3.7, the percentage of execution time spent on particle routing and collective communication, for the two PPFs, on the Lenngren cluster, are presented. The most important observations are the following. First, collective communication is the main reason for the efficiency loss, as the number of processors, P, increase. In the benchmark case, with $N = 40k$ particles, and $P = 64$ processors, almost half of the execution time is spent on collective communication, whereas particle routing accounts for 12% of the time for PPF1, and 10% for PPF2. Second, the fraction of time spent on particle routing is significantly larger for PPF1 in the extreme case, in comparison with the benchmark case, which is due to the increased variable particle routing requirement in the extreme case. Particle routing performance of PPF2, on the other hand, is largely disconnected from statistical performance, as measured by the effective sample size, N_{eff}.

In figure 3.1 the absolute times of collective communication and particle routing are displayed, for all the three clusters. In figures 3.1a-c the time of collective communication, i.e. the time of a call to the *allgather* routine, is presented. In figures 3.1a and 3.1b, a curve, $t = \alpha + \beta log_2 P$, has been fitted to the time observations, to aid the visual interpretation. On the Lenngren cluster, collective communication time is roughly linear in P for the relevant range of processors, i.e. large P. On the Lucidor

Table 3.7 Profile of communication on the Lenngren cluster. Percentage of execution time spent on collective communication and particle routing.

	PPF1						
Particles, N	40				160		
Processors, P	8	16	32	64	16	32	64
Collective							
Benchmark	2	6	23	46	2	7	22
Extreme	2	6	19	42	2	6	18
Particle routing							
Benchmark	4	8	12	12	4	8	13
Extreme	9	18	21	24	11	21	32
	PPF2						
Particles, N	40				160		
Processors, P	8	16	32	64	16	32	64
Collective							
Benchmark	2	6	22	46	2	7	22
Extreme	2	7	21	47	2	7	23
Particle routing							
Benchmark	2	5	8	10	2	4	8
Extreme	2	5	8	10	2	4	8

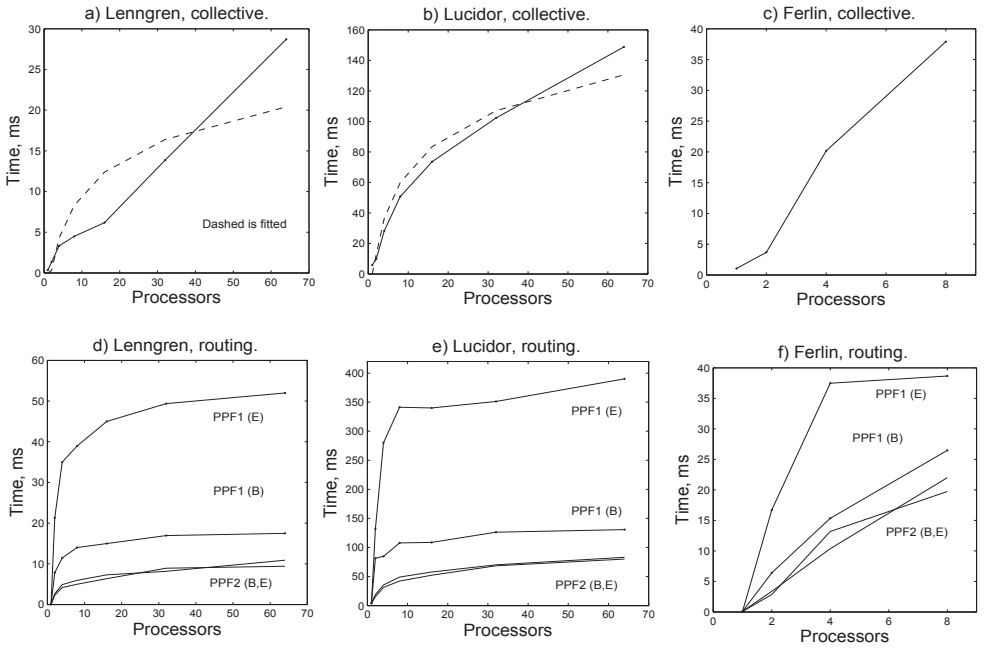
cluster, the timings appear to be somewhat closer to the $log_2 P$-shape, associated with an optimal implementation of the *allgather*, on a fully connected network.

Two things are worth to note about the timing of PPF collective communication. First, the measurements are very accurate in this case. Collective communication time does not depend on N, on the PPF algorithm used, or on the scaling of measurement errors in the model. Consequently, for each value of the processors, P, we average the timings from a large number of runs, while observing that the variance in the time observations, for a given P, is very small. Second, we have tested the different variants of collective communication implementation, outlined in appendix C, and the results are fairly insensitive to the usage of other collective communication routines, e.g. *gather-scatter*.

In figures 3.1d-f, the time of particle routing is presented for PPF1 and PPF2, the benchmark (B) and extreme (E) routing cases, and with $N = 160k$ particles. First, the particle routing time of PPF2 does not depend on the performance of the particle filter. For all three clusters, the timings for the benchmark and extreme cases are virtually indistinguishable, which confirms one of the predictions of the routing simulation experiments (section 3.4.3, and appendix B). For PPF1 there is a significant difference in particle routing time, between the benchmark and extreme cases.

Second, on both the Lenngren and Lucidor clusters, the time of routing increases very slowly for $P \geq 8$, after a steep initial increase, see figures 3.1d and 3.1e. We may compare the qualitative features of routing time in the benchmark case to the

Figure 3.1 Time of collective communication and particle routing on the Lenngren, Lucidor and Ferlin clusters.



'predictions' in figures 3.2e (R) and 3.2f ($R_{p,\max}$), in appendix B. A comparison of the qualitative features of these figures says that the total number of particles routed, R, is not a good proxy for the variable part of particle routing time. According to the 'R-model' of routing time the absolute time difference between SPR (PPF1) and AP (PPF2) should grow in P, while this difference is roughly constant in figures 3.1d and 3.1e. The behaviour of actual particle routing time is instead located somewhere between the two extreme 'predictions' provided by figures 3.2e and 3.2f, and perhaps it is closer to the '$R_{p,\max}$-model', once the latency of routing is also taken into account.

Finally, we expect figures 3.1d and 3.1e, i.e. the displayed very slow increase of particle routing time for fixed N and increasing P, to be a useful empirical characterisation of particle routing, which should be of more general interest. These pictures, we believe, provide important clues for understanding the parallel efficiency properties of the PPFs when executed on 'standard' HPC clusters.

3.5.5 Discussion

In this section we summarise the main lessons from the experiments.

Choice of algorithm

In the particular application considered above, the nonlinear estimation of a small scale DSGE model, our recommendation would be to use the simpler PPF1 algorithm. First, our experiments show that the difference in particle routing speed between the two algorithms becomes important only in 'extreme' situations, i.e. when a large number of particles are used and/or when the effective sample size, N_{eff}, is very low. In the benchmark case, which we expect to be more relevant in practise, the differences are fairly small. The robustness of the routing of arbitrary particles approach (AP) is, however, an attractive feature of PPF2, which could prove useful in problems of a larger size.

Second, efficient Fortran code is largely based on writing the code such that Basic Linear Algebra Subroutines (BLAS) can be efficiently utilised. This is a general guideline for efficient coding and the extent to which it is applicable in the particle filter context depends on the particular non-linear state space model (NLSS) under consideration. In both PPF implementations the sampling and importance (SIS) step is implemented as a loop over particles, i.e. particle-by-particle, which is the most natural implementation. The smaller particle routing requirement in PPF2 is, in a sense, bought at the expense of introducing a more complex loop in the SIS step, see section 3.3.3. The complex structure of this loop implies that a particle-by-particle implementation must be used.

The simpler structure of the PPF1 SIS step loop, however, is amenable to manually implemented loop blocking for the second order approximated DSGE model. Blocking simply means that the operations of the SIS step are performed on blocks of particles, i.e. matrices. It is expected to improve performance further since it facilitates a more efficient utilisation of BLAS routines. Further, it appears that the most modern, and fastest, processor type used in our experiments (Ferlin) favours the PPF1 'clean' code implementation, presumably because it is better suited for compiler-level optimisation of the code.

Finally, PPF1 is considerably simpler to implement. PPF2 introduces several difficulties at a detailed level of implementation, which was partly described at the end of section 3.4.3. The modest overall performance differences imply that simplicity of implementation becomes a major argument in favour of PPF1.

Communication

To our knowledge, our paper is the first in which message passing parallel particle filters, which require extensive communication, are empirically evaluated on clusters. A key result in our experiments is that, at least for a moderate number of particles, collective communication, rather than particle routing, is the bottleneck of the parallel particle filter algorithm. The collectives do not reach the performance one would expect under the ideal conditions, described in section 3.4.4 and in appendix C. Our experience is certainly not unique and optimisation of MPI collectives performance remains an active research area (Pjesivac-Grbovic, Angskun, Bosilca, Fagg, Gabriel and Dongarra (2007)). Conversely, the large fraction of time spent on the single

allgather call for large P *may* suggest that the other parts of the PPFs are reasonably efficiently implemented.

Particle routing time increases very slowly in P, for fixed N. This should be explained by a large degree of parallelism in the interconnection networks on the clusters. The *total* number of particles routed, R, is therefore a poor predictor of routing time. In order to experience large time gains with the arbitrary particles approach (AP, PPF2), the number of particles, N, must be large and/or the performance of the particle filter, as measured by the effective sample size, N_{eff}, must be poor.

The results imply that the application of a conditional resampling scheme, instead of resampling at each iteration, would hardly pay off computationally, since collective communication in the particle filter is identical for these two resampling schemes, see section 3.4.4. In order to reduce the usage of collectives in the PPF, one must either use (i) partial unconditional resampling (section 3.3) or (ii) apply deterministic resampling algorithms, where the latter approach may eliminate usage of collectives completely, e.g. when the DRNA algorithm is used for resampling. Finally, (counterfactually) improved performance of collective communication, if possible, would improve the relative performance of PPF2, since particle routing would account for a larger fraction of computational time for large P.

3.6 Conclusion

Particle filter based maximum likelihood, or Bayesian, estimation of non-linearly approximated DSGE models is computationally demanding. In a recent paper at the frontier of this research area, a DSGE model with $n_\theta = 30$ parameters is estimated nonlinearly. The reported likelihood evaluation time is 22 seconds, when a particle filter (implemented using Fortran and OpenMP) is applied with $P = 8$ processors on a Dell server. Obtaining a mere $R = 5,000$ draws from the posterior, using the random walk Metropolis algorithm, takes 38 hours (Fernández-Villaverde et al. (2010)). This example vividly illustrates the demand for fast particle filter algorithms, and/or improved particle filters and sampling algorithms.

In this paper we propose the use of parallel particle filters (PPF) to speed up the particle filter based evaluation of the likelihood. Importantly, our suggested PPF algorithms do not compromise the statistical performance of filtering. In our view this is a desirable feature of a PPF, while it implies a larger cost of communication in comparison with PPF algorithms which put a larger weight on computational performance, see e.g. Miguez (2007). Given this 'restriction', we expect our algorithms to be close to optimal, computationally. In future implementations of PPF2 the DRPA2 algorithm for resampling could be replaced by the parallel residual systematic resampling (RSR) algorithm, such that the systematic resampling method is always used, independent of the number of processors applied.

The presented algorithms should be able to successfully tackle computationally challenging problems, like the one just mentioned, while maintaining a good parallel efficiency. The absolute execution times reported in the paper illustrate the power

of the brute force approach. Further, our recommended algorithm, PPF1, is fairly simple to implement. The modest reductions in PPF execution time, using the more elaborate PPF2 algorithm, is intuitively explained by the high degree of parallelism in communication, when applied on clusters with modern interconnection networks. The robust computational performance of the PPF2 algorithm, when statistical performance is poor, could, however, prove valuable when applied to larger problems. Finally, further improvement of the performance of both algorithms mainly relies on better performance of collective communication routines.

Bibliography

Amisano, G. and Tristani, O. (2007), 'Euro Area Inflation Persistence in an Estimated nonlinear DSGE model', *Journal of Economic Dynamics and Control* . Forthcoming.

An, S. (2005), 'Bayesian Estimation of DSGE Models: Lessons from Second-order Approximations'. Working Paper, University of Pennsylvania.

An, S. and Schorfeide, F. (2007), 'Bayesian Analysis of DSGE Models', *Econometric Reviews* **26(2-4)**, 113–172.

Andreasen, M. (2010), 'Nonlinear DSGE Models and The Optimized Particle Filter'. CREATES Research Paper 2010-5.

Arulampalam, S., Maskell, S., Gordon, N. and Clapp, T. (2002), 'A Tutorial on Particle Filters for On-Line Non-Linear/Non-Gaussian Bayesian Tracking', *IEEE Transactions on Signal Processing* **50(2)**, 174–188.

Aruoba, S. B., Fernández-Villaverde, J. and Rubio-Ramírez, J. F. (2006), 'Comparing Solution Methods for Dynamic Equilibrium Economies', *Journal of Economic Dynamics and Control* **30(12)**, 2477–2508.

Bolić, M. (2004), Architectures for Efficient Implementation of Particle Filters, PhD thesis, Stony Brook University.

Bolić, M., Djurić, P. M. and Hong, S. (2003), 'New Resampling Algorithms for Particle Filters', *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing* **2**, 589–592.

Bolić, M., Djurić, P. M. and Hong, S. (2004), 'Resampling Algorithms for Particle Filters: A Computational Complexity Perspective', *EURASIP Journal of Applied Signal Processing* **15**, 2267–2277.

Bolić, M., Djurić, P. M. and Hong, S. (2005), 'Resampling Algorithms and Architectures for Distributed Particle Filters', *IEEE Transactions on Signal Processing* **53**(7), 2442– 2450.

Brun, O., Teulière, V. and Garcia, J.-M. (2002), 'Parallel Particle Filtering', *Journal of Parallel and Distributed Computing* **62**, 1186–1202.

Carpenter, J., Clifford, P. and Fearnhead, P. (1999), 'An Improved Particle Filter for Non-linear Problems', *IEE Proceedings. Radar and Sonar Navigation* **146(1)**, 2–7.

Chan, E., Heimlich, M., Purkayastha, A. and van de Geijn, R. (2004), 'On Optimizing Collective Communication', *Proceedings of the 2004 IEEE International Conference on Cluster Computing.* pp. 145–155.

Creal, D. (2009), 'A Survey of Sequential Monte Carlo Methods for Economics and Finance', *Technical report, University of Amsterdam* .

Douc, R. and Cappe, O. (2005), 'Comparison of Resampling Schemes for Particle Filtering', *Image and Signal Processing and Analysis, 2005. ISPA 2005. Proceedings of the 4th International Symposium In Image and Signal Processing and Analysis* pp. 64–69.

Doucet, A., de Freitas, N. and Gordon, N. (2001), *Sequential Monte Carlo Methods in Practice*, Springer.

Doucet, A., Godsill, S. and Andrieu, C. (2000), 'On Sequential Monte Carlo Sampling Methods for Bayesian Filtering', *Statistics and Computing* **10**, 197–208.

Erceg, C. J., Henderson, D. W. and Levin, A. T. (2000), 'Optimal Monetary Policy with Staggered Wage and Price Contracts', *Journal of Monetary Economics* **46(2)**, 281–313.

Fernández-Villaverde, J. (2009), 'The Econometrics of DSGE Models', *NBER Working Paper, No. w14677* .

Fernández-Villaverde, J., Guerron-Quintana, P. and Rubio-Ramírez, J. F. (2010), 'Fortune or Virtue: Time-Variant Volatilities Versus Parameter Drifting in US Data', *NBER Working Paper No. 15928* .

Fernández-Villaverde, J. and Rubio-Ramírez, J. F. (2007), 'Estimating Macroeconomic Models: A Likelihood Approach', *Review of Economic Studies* **74(4)**, 1059–1087.

Fernández-Villaverde, J., Rubio-Ramirez, J. F. and Santos, M. (2004), 'Convergence Properties of the Likelihood of Computed Dynamic Models', *Econometrica* **74**, 93–119.

Flury, T. and Shephard, N. (2008), 'Bayesian Inference Based Only on Simulated Likelihood: Particle Filter Analysis of Dynamic Economic Models'. Economics working paper 413, Oxford University.

Harvey, A. (1989), *Forecasting, Structural Time Series Models and the Kalman Filter*, Cambridge University Press.

Hol, J. D., Schn, T. B. and Gustafsson, F. (2006), 'On Resampling Algorithms for Particle Filters'. Conference paper. Nonlinear Statistical Signal Processing Workshop, Cambridge.

Judd, K. (1998), *Numerical Methods in Economics*, MIT Press, Cambridge.

Kuensch, H. (2005), 'Recursive Monte Carlo Filters: Algorithms and Theoretical Analysis', *Annals of Statistics* **33**, 1983–2021.

Larsson Trff, J., Gropp, W. D. and Thakur, R. (2009), 'Self-Consistent MPI Performance Guidelines', *IEEE Transactions on Parallel and Distributed Systems.* **99**, 698–709. RapidPosts.

Liu, J. S., Chen, R. and Logvinenko, T. (2001), A Theoretical Framework for Sequential Importance Sampling with Resampling, *in* A. Doucet, N. de Freitas and N. Gordon, eds, 'Sequential Monte Carlo Methods in Practice', Springer, chapter 11, pp. 224–246.

Miguez, J. (2007), 'Analysis of Parallelizable Resampling Algorithms for Particle Filtering', *Signal Processing* **87**, 3155–3174.

Pacheco, P. (1997), *Parallel Programming with MPI*, Morgan Kaufmann Publishers, San Franscisco.

Pea, J., Qualité, L. and Tillé, Y. (2007), 'Systematic Sampling is a Minimum Support Design', *Computational Statistics and Data Analysis* **51**, 5591–5602.

Pichler, P. (2008), 'Forecasting with DSGE Models: The Role of Nonlinearities', *The B.E. Journal of Macroeconomics (Topics)* **8(1)**.

Pjesivac-Grbovic, J., Angskun, T., Bosilca, G., Fagg, G. E., Gabriel, E. and Dongarra, J. J. (2007), 'Performance Analysis of MPI Collective Operations', *Cluster Computing* **10(2)**, 127–143.

Schmitt-Grohe, S. and Uribe, M. (2004), 'Solving Dynamic General Equilibrium Models Using a Second-Order Approximation to the Policy Function', *Journal of Economic Dynamics and Control* **28**.

Strid, I. (2010), 'Efficient Parallelisation of Metropolis-Hastings Algorithms Using a Prefetching Approach', *Computational Statistics and Data Analysis* **54(11)**, 2814–2835.

Teulière, V. and Brun, O. (2003), 'Parallelisation of the Particle Filtering Technique and Application to Doppler-Bearing Tracking of Manouvering Sources', *Parallel Computing* **29**, 1069–1090.

Tillé, Y. (2006), *Sampling Algorithms*, Springer.

Winschel, V. (2005), 'Solving, Estimating and Selecting Nonlinear Dynamic Economic Models Without the Curse of Dimensionality'. Manuscript, University of Mannheim.

Woodford, M. (2003), *Interest and Prices*, Princeton University Press.

Appendix

Appendix A. Distributed systematic resampling

The main objective in this appendix is to construct a parallel residual systematic resampling (RSR) algorithm, i.e. an algorithm which is statistically equivalent to the RSR algorithm (algorithm 3) for any value of the number of processors, P, applied with the algorithm. We also provide some simple properties of the SR and RSR algorithms.

The RSR algorithm in matrix form

The RSR algorithm can be written compactly in matrix form as

$$r = r_{RSR}(q, N_{in}, N_{out}, u) = A^{-1}\left(\lceil A(\hat{q} - \upsilon)\rceil\right), \tag{3.103}$$

where r is the vector of replication factors. The N_{out}-scaled vector of weights is

$$\hat{q} = N_{out}q = (N_{out}/\tilde{S})\tilde{q}, \tag{3.104}$$

where q is the vector of normalised weights and $\tilde{q}$ is a vector of un-normalised, i.e. arbitrarily scaled, weights, with the sum of weights, $\tilde{S} = \tilde{q}^T\iota_{N_{in}}$. Further,

$$\upsilon = (u\ 0\ \ldots\ 0)^T, \tag{3.105}$$

where $u \sim U(0,1)$. The vectors r, $\tilde{q}$, $\hat{q}$, q, and υ have dimension $N_{in} \times 1$ and the ceiling operator applied to a vector v of length L is

$$\lceil v^T\rceil = \left(\lceil v^1\rceil\ \ \lceil v^2\rceil\ \ \ldots \lceil v^L\rceil\right)^T,$$

and similarly for the floor operator. The lower triangular $N_{in} \times N_{in}$ matrix A is the cumulative sum operator

$$A_{i,j} = 1 \text{ if } i \geq j \text{ and } A_{i,j} = 0 \text{ otherwise.} \tag{3.106}$$

and the inverse, A^{-1}, is the difference operator

$$A^{-1}_{i,i} = 1,\ A^{-1}_{i,i-1} = -1 \text{ and } A^{-1}_{i,j} = 0 \text{ otherwise.} \tag{3.107}$$

We also note that

$$A\upsilon = u\iota_{N_{in}}.$$

Expression [3.103] can be verified by writing the replication factor of particle m as

$$\begin{aligned} r^m &= \left\lfloor (\sum_{i=1}^{m}\hat{q}^i - \sum_{i=1}^{m-1} r^i - u)\right\rfloor + 1 = \\ &= \left\lfloor \sum_{i=1}^{m}\hat{q}^i - u\right\rfloor + 1 - \sum_{i=1}^{m-1} r^i, \end{aligned}$$

by 'solving' the loop in algorithm 3. The latter step follows since any sum of replication factors is an integer, and $\lfloor a + b\rfloor = a + \lfloor b\rfloor$, where a is an integer and b is a real number.

Some properties of the SR and RSR algorithms

It is immediately seen from the matrix form of the RSR algorithm, [3.103], and [3.104], that, for an arbitrary real scalar $\lambda \neq 0$, we have

$$r_{RSR}(q, N_{in}, N_{out}, u) = r_{RSR}(\lambda q, N_{in}, \lambda^{-1} N_{out}, u),$$

which is stated as result 5 in the main text. For example, in the particle filter with $q = w$ and $N_{in} = N_{out} = N$, if we choose $\lambda = \tilde{S} = \tilde{w}^T \iota_N$ the RSR algorithm can be applied without prior normalisation of the weights, i.e.

$$r = r_{RSR}(\tilde{w}, N, \tilde{S}^{-1} N, u),$$

which implies that N divisions are replaced by a single division prior to resampling.

The following result relates the SR and RSR algorithms with the residual resampling (RR) algorithm:

Result 9 *The vector of replication factors obtained using the SR algorithm (or equivalently using the RSR algorithm)*

$$r = SR(q, N_{in}, N_{out}, u), \tag{3.108}$$

can be expressed as

$$r = \lfloor N_{out} q \rfloor + r_r, \tag{3.109}$$

where the vector of residual replication factors, r_r, is given by

$$r_r = SR(q_r, N_{in}, N_r, u), \tag{3.110}$$

where

$$q_r = N_r^{-1} \left(N_{out} q - \lfloor N_{out} q \rfloor \right)$$

are the normalised residual weights and

$$N_r = N_{out} - \sum_{m=1}^{N_{in}} \lfloor N_{out} q^m \rfloor = \iota_{N_{in}}^T r_r$$

is the number of residually resampled particles. In other words, the residual resampling (RR) algorithm with SR (or RSR) applied for remainder resampling, which is described by [3.109] and [3.110], is strictly statistically equivalent to the SR (and hence also the RSR) algorithm, [3.108]. The residual replication factor, r_r^m, is either equal to zero or one.⊡

Sampling r_r using SR (or RSR), [3.110], is equivalent to applying *systematic sampling* to sample N_r objects without replacement from a population of size N_{in} where the objects are sampled with unequal probabilities and where the inclusion probabilities,

$$\hat{q}_r^m = \Pr\{r_r^m = 1\}, \tag{3.111}$$

are given by

$$\hat{q}_r^m = N_r q_r^m = N_{out} q^m - \lfloor N_{out} q^m \rfloor =$$

$$= \mathrm{mod}(N_{out} q^m, 1),\ 0 < \hat{q}_r^m < 1,\ m = 1, \ldots, N_{in}, \tag{3.112}$$

see e.g. Tillé (2006), algorithm 7.1, p. 124.

Result 9 is verified through the following steps:

$$\begin{aligned} r_r = RSR(q_r, N_{in}, N_r, u) &= RSR(N_r q_r, N_{in}, 1, u) = \\ &= RSR(N_{out} q - \lfloor N_{out} q \rfloor, N_{in}, 1, u) = \\ &= A^{-1}\left(\lceil A\left(N_{out} q - \lfloor N_{out} q \rfloor\right) - A\upsilon \rceil\right) = \\ &= A^{-1}\left(\lceil A N_{out} q - A\upsilon - A\lfloor N_{out} q \rfloor \rceil\right) = \\ &= A^{-1}(\lceil A N_{out} q - A\upsilon \rceil - \lceil A\lfloor N_{out} q \rfloor \rceil) = \\ &= A^{-1}\left(\lceil A N_{out} q - A\upsilon \rceil\right) - \lfloor N_{out} q \rfloor = \\ &= r - \lfloor N_{out} q \rfloor, \end{aligned}$$

where we use result 5 and simple properties of the floor and ceil operators.

In fact, we note that

$$RSR(N_{out} q, N_{in}, 1, u) = r^* + RSR(N_{out} q - r^*, N_{in}, 1, u),$$

where r^* is an arbitrary vector of integers. The RR algorithm, with systematic sampling applied for remainder resampling, (RR-SS) uses $r^* = \lfloor q N_{out} \rfloor$, while it is computationally most efficient to use $r^* = 0$. It is possible to implement the RR-SS algorithm using one loop, i.e. it has complexity $O(N_{in})$. The RR-SS algorithm, however, requires $2N_{in}$ instances of the floor operator, while the RSR algorithm (algorithm 3) only applies the floor operator N_{in} times. In summary, the RR algorithm, with SS applied for remainder resampling, is statistically equivalent to SR/RSR but computationally inferior.

Conditional on the vector of weights, q, the vector of replication factors produced by the SR/RSR algorithm is a function of the uniform random number u, $r = r(u)$. The support,

$$\mathcal{R} = \{r : r = r(u|q), u \in (0,1)\},$$

of the algorithm is the set of vectors of replication factors, conditional on q, where each vector corresponds to a realisation of u in an interval, $u \in (u_L, u_H)$. The size of the support of SR and RSR, when applied in the particle filter outlined in algorithm 1, is given by the following result which follows immediately from result 9 and a result on systematic sampling in Pea, Qualité and Tillé (2007).

Result 10 *The size of the support of the SR (and RSR) algorithm is equal to the number of distinct* $c^i = mod(\sum_{m=1}^{i} \hat{q}_r^m, 1) = mod(N_{out} \sum_{m=1}^{i} q_r^m, 1)$, $i = 1, 2, \ldots, N_{in}$. *The size of the support is therefore not larger than* N_{in}. *When the SR algorithm is applied in a particle filter, where all* c^i *are typically distinct, the size of the support is exactly* N_{in}.⊡

The size of the support of the $\text{RPA}_{\tilde{P}}$ algorithm follows from the previous result. The size of the support of the RSR interresampling step of the $\text{RPA}_{\tilde{P}}$ algorithm is equal to $\tilde{P}$. For each of the $\tilde{P}$ possible realisations of interresampling the size of the support in intra-resampling for a process p is $\tilde{M}$. The size of the support of the $\text{RPA}_{\tilde{P}}$ algorithm is therefore $\tilde{P}\tilde{M}^{\tilde{P}}$.

Result 11 *The size of the support of the $RPA_{\tilde{P}}$ algorithm is equal to $\tilde{P}\tilde{M}^{\tilde{P}}$, where $N_{in} = \tilde{M}\tilde{P}$, and where $\tilde{P}$ is the number of groups, or strata, and $\tilde{M}$ is the number of particles per group.*$\boxdot$

The parallel RSR algorithm

In a parallel particle filter with a fixed number of particles, N, the parallel residual systematic resampling (RSR) algorithm would be applied with $N_{in} = N_{out} = N$, while here we keep these arguments separated for the purposes of clarity and generality. At the outset of resampling each process p is assumed to hold an arbitrarily scaled $(N_{in}/P) \times 1 = M \times 1$ vector of weights, $\tilde{q}_p$, e.g. in the particle filter $\tilde{q}_p = \tilde{w}_p$. The vector of N_{out}-scaled weights, in [3.104], is partitioned as

$$\hat{q} = \left(\hat{q}_1^T \ \hat{q}_2^T \ \ldots \hat{q}_P^T\right)^T,$$

where the N_{out}-scaled sub-vector of weights is obtained as

$$\hat{q}_p = (N_{out}/\tilde{S})\tilde{q}_p,$$

such that the processes must have access to the global sum of un-normalised weights, $\tilde{S} = \tilde{q}^T \iota_{N_{in}}$, in order to obtain $\hat{q}_p$.

Analogously to [3.105], we define the $M \times 1$ vector,

$$v = (u \ 0 \ \ldots \ 0)^T, \tag{3.113}$$

and the $M \times 1$ vectors $\hat{v}_p$, $p = 1, \ldots, P$, where all elements are zero except the first element which is

$$\hat{v}_p^1 = u - \hat{C}_p, \ p = 1, \ldots, P \tag{3.114}$$

where

$$\hat{C}_p = \sum_{k=1}^{p-1} \hat{S}_k, \ p = 1, \ldots, P \tag{3.115}$$

is the cumulative sum of N_{out}-scaled process weights (and $\hat{C}_1 = 0$). The scaled process weight is $\hat{S}_k = N_{out} S_k$, where $S_k = \tilde{S}_k/\tilde{S}$ is the globally normalised process sum of weights.

Further, the cumulative N_{out}-scaled weights,

$$\hat{c}^k = \sum_{i=1}^{k} \hat{q}^i, \ k = 1, \ldots, N_{in}$$

are related to the cumulative scaled process weights, $\hat{C}_p$, through

$$\hat{c}^{(p-1)M+j} = \hat{C}_p + \sum_{i=1}^{j} \hat{q}^{(p-1)M+i} = \hat{C}_p + \sum_{i=1}^{j} \hat{q}_p^i.$$

In particular,

$$\hat{v}_p^1 = u - \hat{c}^{(p-1)M} = u - \hat{C}_p.$$

In the parallel RSR algorithm the full vector of replication factors is

$$r = \left(r_1^T \; r_2^T \ldots r_P^T\right)^T, \tag{3.116}$$

where the processes obtain their vectors of replication factors, r_p, in parallel using

$$r_p = A_M^{-1}\left(\lceil A_M\left(\hat{w}_p - \hat{v}_p\right)\rceil\right) - \lceil -\hat{v}_p \rceil =$$

$$= RSR(\hat{w}_p, M, 1, \hat{v}_p^1) - \lceil -\hat{v}_p \rceil = \; \bar{r}_p \; - \lceil -\hat{v}_p \rceil, \tag{3.117}$$

where A_M is the $M \times M$ cumulative sum matrix defined in [3.106] and where we refer to the vector $\bar{r}_p$ as an auxiliary vector of replication factors. In order for a process p to obtain the vector of replication factors, r_p, for its subset of particles it needs the vector of weights, $\tilde{w}_p$, the uniform random number, u, the global sum of weights, $\tilde{S}$, and the cumulative sum of weights for processes 1 to $p-1$, $\hat{C}_p$.

It is straightforward to verify that the vector of replication factors, r, obtained with [3.116] and [3.117] is equivalent to the vector obtained using [3.103]. Write out the vector of replication factors, r, in [3.103] explicitly in the following way

$$\left(\begin{array}{c}
\left(\begin{array}{c} \lceil \hat{c}^1 - u \rceil \\ \lceil \hat{c}^2 - u \rceil - \lceil \hat{c}^1 - u \rceil \\ \cdots \\ \lceil \hat{c}^M - u \rceil - \lceil \hat{c}^{M-1} - u \rceil \end{array}\right) \\
\left(\begin{array}{c} \lceil \hat{c}^{M+1} - u \rceil \\ \lceil \hat{c}^{M+2} - u \rceil - \lceil \hat{c}^{M+1} - u \rceil \\ \cdots \\ \lceil \hat{c}^{2M} - u \rceil - \lceil \hat{c}^{2M-1} - u \rceil \end{array}\right) \\
\cdots \\
\left(\begin{array}{c} \lceil \hat{c}^{(P-1)M+1} - u \rceil \\ \lceil \hat{c}^{(P-1)M+2} - u \rceil - \lceil \hat{c}^{(P-1)M+1} - u \rceil \\ \cdots \\ \lceil \hat{c}^{PM} - u \rceil - \lceil \hat{c}^{PM-1} - u \rceil \end{array}\right)
\end{array}\right) - \left(\begin{array}{c}
\left(\begin{array}{c} \lceil \hat{c}^0 - u \rceil \\ 0 \\ \cdots \\ 0 \end{array}\right) \\
\left(\begin{array}{c} \lceil \hat{c}^M - u \rceil \\ 0 \\ \cdots \\ 0 \end{array}\right) \\
\cdots \\
\left(\begin{array}{c} \lceil \hat{c}^{(P-1)M} - u \rceil \\ 0 \\ \cdots \\ 0 \end{array}\right)
\end{array}\right)$$

and note that this can be expressed in the partitioned form

$$r = \left(\begin{array}{c}
A_M^{-1}\left(\left\lceil A_M\left(\hat{q}_1 - v\right) + \hat{C}_1 \iota_M \right\rceil\right) \\
A_M^{-1}\left(\left\lceil A_M\left(\hat{q}_2 - v\right) + \hat{C}_2 \iota_M \right\rceil\right) \\
\cdots \\
A_M^{-1}\left(\left\lceil A_M\left(\hat{q}_P - v\right) + \hat{C}_P \iota_M \right\rceil\right)
\end{array}\right) - \left(\begin{array}{c}
\lceil -\hat{v}_1 \rceil \\ \lceil -\hat{v}_2 \rceil \\ \cdots \\ \lceil -\hat{v}_P \rceil
\end{array}\right) =$$

$$= \begin{pmatrix} A_M^{-1}\left(\lceil A_M\left(\hat{q}_1 - \hat{v}_1\right)\rceil\right) \\ A_M^{-1}\left(\lceil A_M\left(\hat{q}_2 - \hat{v}_2\right)\rceil\right) \\ \cdots \\ A_M^{-1}\left(\lceil A_M\left(\hat{q}_P - \hat{v}_P\right)\rceil\right) \end{pmatrix} - \begin{pmatrix} \lceil -\hat{v}_1 \rceil \\ \lceil -\hat{v}_2 \rceil \\ \cdots \\ \lceil -\hat{v}_P \rceil \end{pmatrix} = \begin{pmatrix} \bar{r}_1 \\ \bar{r}_2 \\ \cdots \\ \bar{r}_P \end{pmatrix} - \begin{pmatrix} \lceil -\hat{v}_1 \rceil \\ \lceil -\hat{v}_2 \rceil \\ \cdots \\ \lceil -\hat{v}_P \rceil \end{pmatrix},$$

since

$$A_M\left(\hat{q}_p - v\right) + \hat{C}_p \iota_M = A_M \hat{q}_1 - (\hat{C}_p - u)\iota_M = A_M(\hat{q}_p - \hat{v}_p).$$

The algorithm is summarised below.

Algorithm 12 *Distributed RSR algorithm:* $r = DRSR_P(q, N_{in}, N_{out}, u)$

1. At the outset of resampling each process holds an arbitrarily scaled, or un-normalised, vector of weights, $\tilde{q}_p$, $p = 1, \ldots, P$.
2. The master process collects the vector of un-normalised process weights,

   $$\tilde{s} = (\tilde{S}_1\ \tilde{S}_2 \ldots \tilde{S}_p)^T,$$

   (*gather*), where $\tilde{S}_p = \tilde{q}_p^T \iota_M$.
3. The master process generates $u \sim U(0,1)$ and calculates $\hat{v}_p^1$ and $\hat{C}_p$, in [3.114] and [3.115], using the process sum of non-normalised weights, $\tilde{S}_p$, $p = 1, \ldots, P$.
4. The master process distributes the scalars $\hat{v}_p^1$ and $\tilde{S}$ to the processes (*scatter*).
5. The processes obtain the local vectors of replication factors, r_p, in parallel. First, the auxiliary vector of replication factors is computed

   $$\bar{r}_p = RSR(\lambda \hat{q}_p, M, \lambda^{-1}, \hat{v}_p^1),\ p = 1, \ldots, P, \tag{3.118}$$

   where $r_p^m = \bar{r}_p^m$ for $1 < m \leq M$, and where $\lambda \neq 0$ is an arbitrary real scalar. After obtaining $\bar{r}_p$ the first element is 'corrected'

   $$r_p^1 = \bar{r}_p^1 - \left\lceil -\hat{v}_p^1 \right\rceil. \tag{3.119}$$

   to produce the process vector of replication factors, r_p.⊡

It is important to note that, in the DRSR algorithm, the RSR algorithm, in [3.118], is used merely as a computational device. It is straightforward to implement step 4 such that the replication factors are obtained in order, which is important in the parallel particle filter in order to simplify the packaging of particles for particle routing.

Inter-resampling is not performed in the DRSR algorithm, while the process replication factors, $N_p = r_p^T \iota_M$, may be obtained locally after resampling. In a parallel particle filter, N_p is needed to perform scheduling, and the full vector of process replication factors, n, must be held by the master process. In the PPF we therefore suggest to obtain n in step 3 of the DRSR algorithm. We may use the RSR algorithm to simultaneously calculate N_p and the cumulative process weights, $\hat{C}_p$.

Appendix B. Particle routing

The execution time of particle routing on a parallel computer depends, in a complex way, on the method of particle routing and on the properties of the interconnection network. A very simple, abstract, framework is used to illustrate how the wall-clock time of particle routing depends on these factors. The time of routing is split into a fixed cost, or latency, term and a variable cost, or bandwith, term.

It is assumed that, in the absence of network conflicts, the time to send a message between two, arbitrarily chosen, processors is

$$t_{p,\tilde{p}} = \alpha + \beta k R_{p,\tilde{p}}, \tag{3.120}$$

where α is the latency for a message, β is the reciprocal of the network bandwith, $R_{p,\tilde{p}}$ is the number of particles routed between the sender, p, and the receiver, $\tilde{p}$, and k is a constant which translates particles into bytes (i.e. $k = 8n_X$, with double precision floating point). Further, we define the matrix $\tilde{E} = \{\tilde{E}_{p,\tilde{p}}\}$, where $\tilde{E}_{p,\tilde{p}} = I(K_{p,\tilde{p}} \neq 0)$, i.e. a matrix which describes the non-zero pattern of the exchange matrix, E.

The properties of the interconnection network may, abstractly, be described via an assumption on the number of simultaneous transfers, or messages, through the network. Two assumptions are considered:

- Only one transfer can occur at a given time in the network. (A1)
- A process can be involved in only one transfer at a given point in time, while there are no other restrictions on the number of parallel transfers in the network, i.e. with P processes the maximum number of simultaneous transfers is $P/2$. (A2) This assumption is used e.g. by Chan, Heimlich, Purkayastha and van de Geijn (2004).

In the first case (A1) the total time of particle routing is

$$t_1 = \sum_{p:R_{p,\tilde{p}}>0} \alpha + \beta k R_{p,\tilde{p}} = (P-1)\alpha + \beta k R, \tag{3.121}$$

i.e. particle routing time is a linear function of the total number of particles routed, R. In the second case (A2)

$$t_2 = \alpha_{\max} + \beta k R_{p,\max}, \tag{3.122}$$

where $R_{p,\max}$ is the maximum number of particles transferred (sent or received) by any process. The latency of routing is $\alpha_{\max} = b_{\max}\alpha$, where $b_{\max}$ is the maximum number of transfers of any process in routing. Formally,

$$b_{\max} = \max_p \bar{e}, \text{ where } \bar{e} = \bar{E}\iota_P. \tag{3.123}$$

The first term in the expressions for particle routing time, [3.121] and [3.122], is the latency of routing and the second term is the variable cost, or bandwith, term. The pattern of exchange in routing, which is captured by $\bar{E}$, is shared by the three routing methods. As a consequence, the latency of routing is identical for the three routing methods, regardless of the type of network.

Particle routing: simulation exercise 1

The number of particles routed, R, is a function of the weight vector, w, the number of particles, N, and the number of processors, P, i.e. $R = R(w, N, P)$. In the first exercise, the properties of the weight vector, w, are kept fixed, while N and/or P are varied. The purpose is to understand how the number of particles routed, with simple particle routing (SPR), $R^{SPR} = K$, and with routing of arbitrary particles (AP), R^{AP}, depends on N and P.

For each combination of N and P, $H = 10k$ weight vectors, of size N, are generated. The elements of an un-normalised vector, $\tilde{w}$, are generated as $\tilde{w}^i = u_i^{\delta}$, where $u_i \sim U(0,1)$, $i = 1, \ldots, N$. The parameter $\delta = 18$ is selected to obtain $N_{eff}/N \approx 10\%$, where N_{eff} is the effective sample size, defined in [3.31]. For each weight vector, the number of particles routed, R, and the maximum number of particles routed by a process, $R_{p,\max}$, are obtained. In the figures, the means (solid) and the 5^{th} and 95^{th} percentiles (dashed), for these quantities, are displayed.

In figures 3.2a and 3.2b, the means and percentiles of R and $R_{p,\max}$ are displayed, as N is varied, while the number of processors is fixed, at the value $P = 16$. In figures 3.2c and 3.2d, N is varied, while the number of particles per process is fixed at $M = N/P = 2,500$. Figure 3.2c illustrates that the particle surplus, K, grows linearly in N, when N_{eff}/N and $M = N/P$ are kept fixed. A lower effective sample size, N_{eff}, is related to an increase in the slope of the line, i.e an increase in K. Finally, in figures 3.2e and 3.2f, P is varied, while $N = 160k$ is fixed[9].

The figures may be interpreted as predicting the variable part of particle routing time, under the two extreme assumptions on the properties of the interconnection network, the single bus (upper panels) and the fully connected network (lower panels). In section 3.5 we apply the PPF algorithms on clusters, holding N fixed and varying P, in correspondence with the simulation in figure 3.2e and 3.2f.

Particle routing: simulation exercise 2

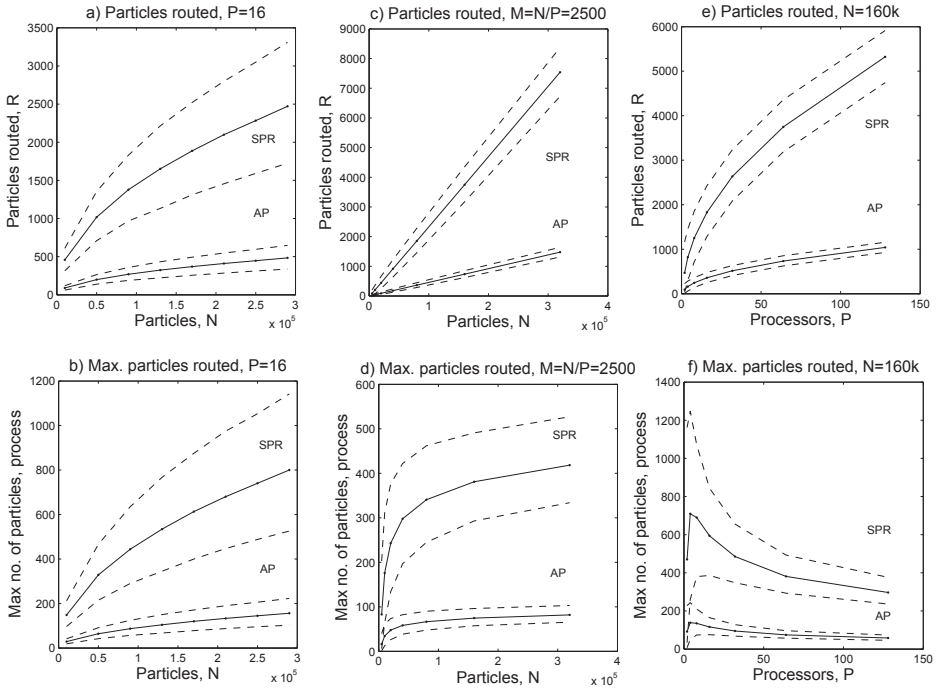
The purpose of the second simulation exercise is to illustrate how the total number of particles routed, R, with each routing method, is related to the effective sample size, N_{eff}, in [3.31]. A higher variance of the weights, w^m, implies a lower effective sample size, and a larger dispersion in the process replication factors, N_p, such that the total particle surplus, $R^{SPR} = K$, is increased.

The relationship between N_{eff} and R is less obvious for the largest weights (LWP) and arbitrary particles (AP) approaches. Intuitively, in the case of AP, the effect of a decrease in N_{eff}, and the accompanied increase in the particle surplus, K, is counteracted by an increase in the average replication factor of the transferred particles, $\bar{r}_c$ in [3.99].

The vectors of particle weights, w, used in the experiment, are generated by running the Metropolis-Hastings algorithm for H iterations, to estimate the DSGE model, as in section 3.5. Here, it is sufficient for our purposes to note that this yields a sample of HT vectors of weights, of length N, $\{w_i\}_{i=1}^{HT}$, where each weight vector

[9]Note that $K = K_{p,\max}$ for $P \leq 3$. This explains the shape of the curves in figure 3.2 f.

Figure 3.2 Particles routed, R, and maximum number of particles routed for a process, $R_{p,\max}$.



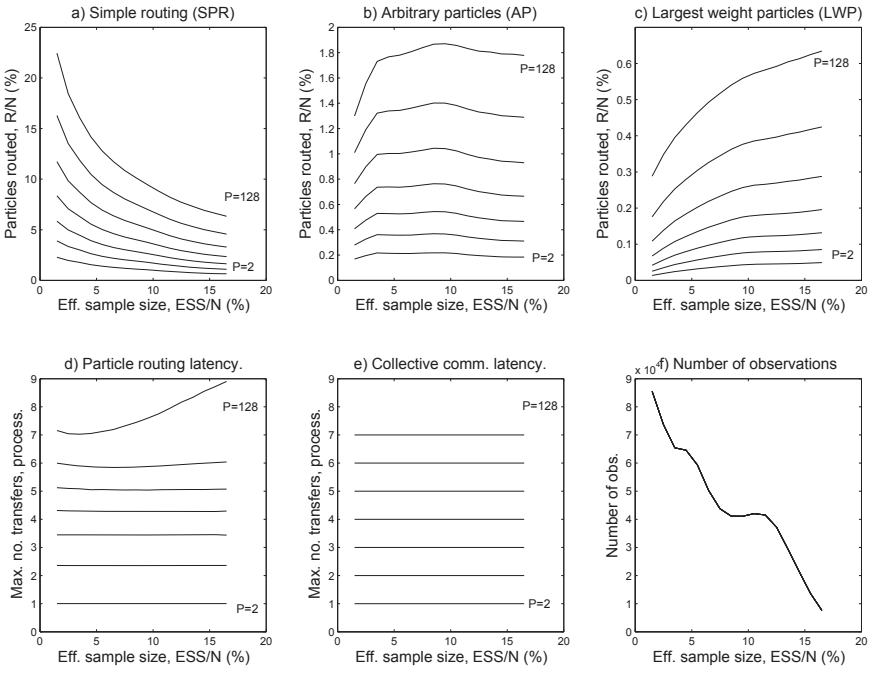
corresponds to an iteration of the particle filter, and where T is the number of time periods of data used to estimate the model. For this sample, the particles routed for the three routing methods, R, and the estimate of the effective sample size, N_{eff}, in [3.31], are calculated, where the number of particles, N, is fixed and the number of processors is varied, $P = 2^k$, $k = 1, \ldots, 7$. In the experiment we let $H = 10k$, $T = 90$ and $N = 32k$.

Next, a grid is constructed for the HT normalised effective sample sizes, N_{eff}/N. These turn out to be in the range 1 to 17%, and this interval is split into subintervals of length 0.5%. The skewed distribution of effective sample sizes is displayed in figure 3.3 f. The average of the number of particles routed, $\bar{R}$, is calculated for each subinterval of the grid and is then normalised by N, to yield $\bar{R}/N$.

In figure 3.3a, it is first verified that the number of particles routed with SPR, $R^{SPR} = K$, decreases in the effective sample size, N_{eff}. When the performance of the particle filter becomes poor, the particle surplus, K, increases significantly. For LWP the relation between N_{eff} and R is the reverse: the routing requirement, R^{LWP}, is increasing in the effective sample size, N_{eff}, as seen in figure 3.3c. Finally, using AP

it is seen that the routing requirement, R^{AP}, is essentially flat in N_{eff}, figure 3.3b.

Figure 3.3 Particles routed and theoretical latency of routing and collective communication.



In figure 3.3d the average simulated latency of particle routing is plotted against the effective sample size. It is seen that the latency, defined as the maximum number of messages of any process in routing, is largely independent of the effective sample size, N_{eff}. An exception occurs in the very fine-grained case, when the number of particles per process, $M = N/P$, is small, e.g. when $M = 32k/128 = 250$.

We conclude with two remarks about the exercise. First, the particular way of generating the weight vectors, i.e. using the DSGE model estimation, does not appear to have any important influence on the results of the experiments. We have tried other, simpler, methods of generating the sample of weight vectors, w, and the results are very similar. Second, changing the number of particles, N, at which the calculations are performed, affects the levels of $\bar{R}/N$, while leaving the qualitative features, displayed in figure 3.3, unchanged.

Appendix C. Collective communication

The purpose of the collectives is to provide a set of efficient routines to perform common communication operations, which involve a group of processes. The availability of these routines also simplifies parallel programming. The implementations in this paper can potentially use four collective communication routines. These are described briefly, and next, three possible implementations of PPF collective communication are described.

The *broadcast* transmits an identical message, e.g. a vector y, from the root process to the other processes, $p = 1, \ldots, P$. Upon completion of the broadcast, all processes have a copy of y. The *scatter* partitions a vector $x = (x_1^T, x_2^T, \ldots, x_P^T)^T$, initially stored by the root process, and transmits the subvector x_p^T, from the root process to process p, for $p = 1, \ldots, P$. The *gather* is the inverse of the scatter. The processes transmit x_p^T to the root process. Upon completion of the gather, the root process is left with the full vector, x. The *allgather* is functionally equivalent to a *gather*, followed by a *broadcast*. Initially process p has the data x_p^T, and upon completion *all* processes have the full vector x.

We identify three ways of arranging interresampling, scheduling and the collective communication associated with these steps. These are described using the collectives involved:

Gather-Scatter: *Gather* the un-normalised process weights, $\left\{\tilde{S}_p\right\}_{p=1}^{P}$, $(P, 1, 1)$, $\Rightarrow$ Interresampling (master) $\Rightarrow$ Scheduling (master) $\Rightarrow$ *Scatter* the output from scheduling, e.g. the rows of the exchange matrix E, to the processes, $(1, P, P)$.

Gather-Broadcast: *Gather* the un-normalised process weights, $\left\{\tilde{S}_p\right\}_{p=1}^{P}$, $(P, 1, 1)$, $\Rightarrow$ Interresampling (master) $\Rightarrow$ *Broadcast* the process replication factors, $\{N_p\}_{p=1}^{P}$, $(1, P, P)$, $\Rightarrow$ Scheduling (all).

Allgather: *Allgather* the un-normalised process weights, $\left\{\tilde{S}_p\right\}_{p=1}^{P}$, $\Rightarrow$ $(P, 1, P)$ $\Rightarrow$ Interresampling (all) $\Rightarrow$ Scheduling (all).

Here 'master' means that only the master process performs the step, while the other processes remain idle, and 'all' means that identical calculations are replicated, in parallel, by all processes. The notation (a, b, c) means that a processes send b elements each to c processes. We also note that the *allgather* implementation requires that the T uniform random numbers, used in inter-resampling, are shared by all processes, which can be arranged using a single *broadcast* at the outset of the particle filter.

The performance of collective communication depend mainly on the PPF collective communication implementation (algorithm), the MPI implementation of the collectives (software) and on the type of interconnection network (hardware). The variable cost of collective communication in the PPF may be disregarded, since the amount of data transferred by the collectives is always small. Assuming tree-based, recursive doubling, algorithms for the *gather*, *broadcast* and *scatter* operations, and a fully connected network, the minimum latency of each of these operations is proportional to $\log_2 P$, see e.g. Chan et al. (2004). Assuming an optimal algorithm for the *allgather*

its minimum latency is also proportional to $\log_2 P$, see e.g. Pacheco (1997), ch. 13. On theoretical grounds, the *allgather* implementation of collective communication in the PPF should therefore be preferred, assuming ideal algorithms for the collectives and a fully connected network. Alternatively, in the absence of assumptions on the type of network it may still be argued that the *allgather* should at least not be slower than the alternatives (Larsson Trff, Gropp and Thakur (2009)). Experimenting with the three variants on the clusters mentioned in the main text, we have, however, found that they are largely equivalent. A comparison of the theoretical, simulated, latency of particle routing and the deterministic latency of collective communication is provided in figure 3.3, parts d and e (Appendix B).

Appendix D. Computation of estimates

In this appendix we briefly describe the computation of estimates in the parallel particle filter. We consider three typical quantities to be estimated using the PF; the sequence of filtering means, $E(X_t|Y_{1:t})$, and covariances, $C(X_t|Y_{1:t})$, $t = 1, \ldots, T$, and the likelihood L. The main point here is that the computation of these estimates, which requires collective communication, need not increase the fixed collective communication cost in the parallel particle filter.

An estimate of the contribution to the likelihood at time t is provided by

$$\hat{L}_t = N^{-1}\tilde{S}_{t|t-1} = N^{-1}\sum_{i=1}^{P}\tilde{S}_{p,t|t-1},\ t = 1, \ldots, T. \tag{3.124}$$

where $\tilde{S}_{p,t|t-1}$ is the local sum of un-normalised weights, in [3.70]. Estimates of the state filtering mean and covariances are provided by

$$\hat{X}_{t|t} = \frac{1}{\tilde{S}_{t|t-1}}\sum_{i=1}^{P}\left(\sum_{j=1}^{M}\tilde{w}_{t|t-1}^{(p-1)M+j}X_{t|t-1}^{(p-1)M+j}\right) = \frac{1}{\tilde{S}_{t|t-1}}\sum_{i=1}^{P}Z_{p,t|t-1},\ t = 1, \ldots, T. \tag{3.125}$$

and

$$\hat{C}_{t|t} = \frac{1}{\tilde{S}_{t|t-1}}\sum_{i=1}^{P}\sum_{j=1}^{M}\tilde{w}_{t|t-1}^{(p-1)M+j}\left(\left(X_{t|t-1}^{(p-1)M+j}\right)\left(X_{t|t-1}^{(p-1)M+j}\right)^T - \hat{X}_{t|t}\hat{X}_{t|t}^T\right) = \tag{3.126}$$

$$= \frac{1}{\tilde{S}_{t|t-1}}\sum_{i=1}^{P}\left(K_{p,t|t-1} - \tilde{S}_{p,t}\hat{X}_{t|t}\hat{X}_{t|t}^T\right),\ t = 1, \ldots, T.$$

where

$$K_{p,t|t-1} = \sum_{j=1}^{M}\tilde{w}_{t|t-1}^{(p-1)M+j}\left(X_{t|t-1}^{(p-1)M+j}\right)\left(X_{t|t-1}^{(p-1)M+j}\right)^T,\ p = 1, \ldots, P,\ t = 1, \ldots, T. \tag{3.127}$$

The quantities which need to be gathered from the processes to calculate $\hat{L}_t$, $\hat{X}_{t|t}$ and $\hat{C}_{t|t}$ are the scalar $\tilde{S}_{p,t}$, the $n_X \times 1$ vector $Z_{p,t|t-1}$, and the $n_X \times n_X$ matrix $K_{p,t|t-1}$. If these quantities are gathered at each iteration, i.e. if estimates need to be computed 'online', they should be grouped into a single message per process, which yields a single *gather*. Alternatively $Z_{p,t}$ and $K_{p,t}$ can be gathered once, at the end of the filter. In either case the fixed collective communication cost is unaffected by the computation of estimates. Note that it is not necessary to collect the vector of process sum of weights, $\tilde{s}_{t|t-1} = (\tilde{S}_{1,t|t-1} \dots \tilde{S}_{P,t|t-1})^T$, at each iteration of the filter, for the purpose of calculating the likelihood estimate, $\hat{L}$. In our context the collection of $\tilde{s}_{t|t-1}$, at each iteration, is performed solely for the purpose of resampling.

Appendix E. The DSGE model

The model used for the experiments is borrowed from An (2005). A brief description is provided here. Details can be found in An's paper, and a general treatment of New Keynesian models is given by e.g. Woodford (2003).

The model consists of a final goods producer, a continuum of producers of intermediate goods, which face quadratic price adjustment costs, and a monetary authority. Aggregate productivity, A, is assumed to follow a random walk with drift,

$$\ln A_t = \ln \gamma + \ln A_{t-1} + \ln z_t, \tag{3.128}$$

where z is a shock to productivity

$$\ln z_t = \rho_z \ln z_{t-1} + \epsilon_{z,t}. \tag{3.129}$$

The productivity process induces a stochastic trend in output and consumption. The output and consumption variables are detrended, such that all variables in the model are stationary, e.g. $c_t = C_t/A_t$ is the detrended consumption.

The stationary dynamic equilibrium is described by eight equations, [3.130]-[3.136], where the variables are expressed in terms of log deviations from the stationary steady state, e.g. $\hat{c}_t = \ln(c_t/c)$, where c is the steady state value of c_t. The parameters of the model are defined in table 3.8 below.

The household's problem yields the Euler equation

$$1 = E_t \left[\exp\left(-\tau \hat{c}_{t+1} + \tau \hat{c}_t + \hat{R}_t - \hat{z}_{t+1} - \hat{\pi}_{t+1} \right) \right], \tag{3.130}$$

where $\hat{c}$ is consumption, $\hat{\pi}$ inflation, $\hat{R}$ is the nominal interest rate, and $\hat{z}$ is the technology shock. The intermediate goods producing firm's profit maximisation yields the condition

$$\frac{1-\nu}{\nu\varphi\pi^2} \left(\exp\left(\tau \hat{c}_t\right) - 1 \right) = \left(\exp\left(\hat{\pi}_t\right) - 1 \right) \left(\left(1 - \frac{1}{2\nu} \right) \exp\left(\hat{\pi}_t\right) + \frac{1}{2\nu} \right) -$$

$$\beta E_t \left(\left(\exp\left(\hat{\pi}_{t+1}\right) - 1 \right) \exp\left(-\tau \hat{c}_{t+1} + \tau \hat{c}_t + \hat{y}_{t+1} - \hat{y}_t + \hat{\pi}_{t+1} \right) \right), \tag{3.131}$$

where $\hat{y}$ is output, and the resource constraint is

$$\exp\left(\hat{c}_t - \hat{y}_t\right) = \exp\left(-\hat{g}_t\right) - \frac{g\varphi\pi^2}{2}\left(\exp\left(\hat{\pi}_t\right) - 1\right)^2. \tag{3.132}$$

The model is closed with a Taylor rule, with interest rate smoothing,

$$\hat{R}_t = \rho_R \hat{R}_{t-1} + (1-\rho_R)\left(\psi_1 \hat{\pi}_t + \psi_2\left(\hat{y}_t - \hat{g}_t\right)\right) + \epsilon_{R,t}, \tag{3.133}$$

where $\epsilon_{R,t} \sim N(0, \sigma_R^2)$ is a monetary policy innovation. The government spending shock, $\hat{g}$, and the technology shock, $\hat{z}$, follow AR(1) processes

$$\hat{g}_t = \rho_g \hat{g}_{t-1} + \epsilon_{g,t},\ \epsilon_{g,t} \sim N(0, \sigma_G^2), \tag{3.134}$$

and

$$\hat{z}_t = \rho_z \hat{z}_{t-1} + \epsilon_{z,t},\ \epsilon_{z,t} \sim N(0, \sigma_Z^2) \tag{3.135}$$

Finally, we have the definition of output growth,

$$\Delta\hat{y}_t = \hat{y}_t - \hat{y}_{t-1}, \tag{3.136}$$

which is included since data on gross domestic product (GDP) growth is used in the estimation.

The vectors of state and control variables are $\mathcal{S}_t = (\epsilon_{R,t}\ \hat{g}_t\ \hat{z}_t\ \hat{y}_{t-1}\ \hat{R}_{t-1})^T$ and $\mathcal{C}_t = (\hat{c}_t\ \Delta\hat{y}_t\ \hat{\pi}_t)^T$. The approximate solution to the model is of the form

$$\mathcal{S}_t = \hat{f}_1(\mathcal{S}_{t-1}, \epsilon_t),\ \mathcal{C}_t = \hat{f}_2(\mathcal{S}_t).$$

The solution is rearranged and the state space representation, [3.3], [3.13], and [3.5], is constructed, i.e. the functions $\hat{g}_1$ and $\hat{g}_2$ in [3.3] and [3.4] are constructed based on $\hat{f}_1$ and $\hat{f}_2$. The state vector in the state space model is defined as $X_t = (X_{1t}^T\ X_{2t}^T)^T$ where the vector of exogenous states is

$$X_{1t} = \begin{pmatrix} \epsilon_{R,t} & \hat{g}_t & \hat{z}_t \end{pmatrix}^T, \tag{3.137}$$

and the remaining state variables are collected in

$$X_{2t} = \begin{pmatrix} \hat{y}_t & \hat{R}_t & \Delta\hat{y}_t & \hat{\pi}_t \end{pmatrix}^T. \tag{3.138}$$

The consumption variable, $\hat{c}_t$, is not a state variable, in the strict sense, and it is not needed in the observation equation (see below). It is therefore excluded from the econometric model, since it is very costly to include 'unnecessary' variables, when estimating the model using the particle filter.

The data used to estimate the model is the quarterly growth rate of GDP, $GROWTH_t$, the annualised quarterly inflation rate, $INFLATION_t$, and the annualised nominal interest rate, $INTEREST_t$. These variables are expressed as percentages, and they are collected in the 3×1 observation vector, Y_t, in the given

order. The observed variables, in Y_t, are related to the variables in the model, X_t, through the observation equation, [3.5], with

$$d = \begin{pmatrix} \gamma^Q \\ \pi^A \\ \pi^A + r^A + 4\gamma^Q \end{pmatrix}, \text{ and } Z = \begin{pmatrix} 0 & 0 & 100 & 0 & 0 & 100 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 400 \\ 0 & 0 & 0 & 0 & 400 & 0 & 0 \end{pmatrix}.$$

The parameters of the model can be divided into three groups: calibrated, estimated, and derived, parameters. The first two groups of parameters are described in table 3.8, along with the values used to generate the data for the experiments (DGP). The parameters in the observation equation, γ^Q, π^A and r^A are related to the model's parameters through:

$$\gamma = \exp(\gamma^Q/100),\ \beta = \exp(-r^A/400),\ \pi = \exp(\pi^A/400),$$

which gives the discount parameter, β, and the steady state values for growth, γ, and inflation, π. The steady state values for the real interest rate, r, the nominal interest rate, R, government spending, g, and output, y, are obtained through:

$$r = \gamma/\beta,\ R = \pi r,\ g = (c/y)^{-1},\ c = \left(\frac{1-\nu}{\chi_H}\right)^{1/\tau},\ y = g\left(\frac{1-\nu}{\chi_H}\right)^{1/\tau}.$$

The price stickiness parameter, φ, in [3.131], is given by

$$\varphi = \frac{\tau(1-\nu)}{\kappa\nu\pi^2}.$$

Table 3.8 DGP parameters in experiment

Estimated parameters		DGP
Inverse intertemporal elasticity of substitution	τ	2.0
Price stickiness	κ	0.33
Inflation parameter, Taylor rule	ψ_1	1.5
Output parameter, Taylor rule	ψ_2	0.125
Interest rate smoothing, Taylor rule	ρ_R	0.75
Ar(1), government spending	ρ_G	0.95
AR(1), technology shock	ρ_Z	0.9
Real interest rate, steady state	r^A	1.0
Inflation, steady state	π^A	3.2
Growth, steady state	γ^Q	0.55
Std., monetary policy shock	σ_R	0.002
Std. government spending innovation	σ_G	0.006
Std. technology innovation	σ_Z	0.003
Calibrated parameters		
Inverse elasticity of demand, intermediate goods	ν	0.10
Consumption-output ratio	c/y	0.85
Labour disutility	χ_H	1.0

Chapter 4

Efficient parallelisation of Metropolis-Hastings using Prefetching

Efficient parallelisation of Metropolis-Hastings algorithms using a prefetching approach[1]

abstract

Prefetching is a simple and general method for single-chain parallelisation of the Metropolis-Hastings algorithm based on the idea of evaluating the posterior in parallel and ahead of time. Improved Metropolis-Hastings prefetching algorithms are presented and evaluated. It is shown how to use available information to make better predictions of the future states of the chain and increase the efficiency of prefetching considerably. The optimal acceptance rate for the prefetching random walk Metropolis-Hastings algorithm is obtained for a special case and it is shown to decrease in the number of processors employed. The performance of the algorithms is illustrated using a well-known macroeconomic model. Bayesian estimation of DSGE models, linearly or nonlinearly approximated, is identified as a potential area of application for prefetching methods. The generality of the proposed method, however, suggests that it could be applied in other contexts as well.

[1]This paper has appeared in *Computational Statistics and Data Analysis* 54, 2814-2835, 2010. I thank Sune Karlsson, John Geweke, Ingelin Steinsland, Karl Walentin, Darren Wilkinson and Mattias Villani for comments and discussions that helped improve this paper. I also thank seminar participants at Sveriges Riksbank, Örebro University and the 14^{th} International Conference on Computing in Economics and Finance in Paris. The Center for Parallel Computers at the Royal Instititute of Technology in Stockholm provided computing time for experiments conducted in the paper.

4.1 Introduction

Efficient single-chain parallelisation of Markov Chain Monte Carlo (MCMC) algorithms is difficult due to the inherently sequential nature of these methods. Exploitation of the conditional independence structure of the underlying model in constructing the parallel algorithm and parallelisation of a computationally demanding likelihood evaluation are examples of problem-specific parallel MCMC approaches (Wilkinson (2006); Strid (2007*b*)). The prefetching approach and 'automatic' parallelisation using parallel matrix routines are more general approaches (Brockwell (2006); Yan, Cowles, Wang and Armstrong (2007)).

In this paper we propose simple improvements to the prefetching Metropolis-Hastings algorithm suggested by Brockwell (2006), which is a parallel processing version of the method originally proposed by Metropolis, Rosenbluth, Rosenbluth, Teller and Teller (1953) and later generalised by Hastings (1970). As the name suggests the idea of prefetching is to obtain several draws from the posterior distribution in parallel via multiple evaluations of the posterior ahead of time. It is assumed that the proposal density depends on the current state of the chain, such that the future states must be predicted. We show how the random walk Metropolis-Hastings (RWMH) prefetching algorithm can be improved by utilising information on the acceptance rate, the posterior and the sequence of realised uniform random numbers in making these predictions. It is also explained how the optimal acceptance rate of the RWMH algorithm depends on the number of processors in a parallel computing setting.

When the proposal density does not depend on the current state the prediction problem vanishes. Parallelisation of the Metropolis-Hastings algorithm is simplified considerably and reminiscent of parallelisation by running multiple chains. The attractiveness of the independence chain Metropolis-Hastings (ICMH) algorithm is therefore obvious from a parallel computing perspective.

The main focus here is on developing efficient prefetching versions of the one-block RWMH algorithm. The one-block case is the most attractive setting from a pure parallel efficiency perspective. Further it allows for a simple exposition of concepts and algorithms. The prefetching method generalises to the multiple blocks case, as we describe briefly. In general, however, we expect it to be less effective in that context, at least in situations where some of the full conditional posteriors can be sampled directly using Gibbs updates.

Prefetching has obvious limitations in terms of parallel efficiency but there are at least three reasons why the approach is still interesting. First is the generality of the one-block prefetching method; it is largely problem independent. Second, prefetching is easy to combine with other parallel approaches and can therefore be used to multiply the effect of a competing parallel algorithm, via construction of a two-layer parallel algorithm. As a result, even if prefetching, by itself, is reasonably efficient only for a small number of processors its contribution to overall performance is potentially large. Finally, the method is easy to implement and provides a cheap way of speeding up already existing serial programs.

The prefetching algorithms are applied exclusively in the context of Bayesian

estimation of Dynamic Stochastic General Equilibrium (DSGE) models. Estimation of large-scale linearised DSGE models or nonlinearly approximated DSGE models of any size are computationally demanding exercises. This class of models is also chosen because the one-block RWMH algorithm has been the predominant choice of sampling method, for reasons explained below.

Despite the focus on macroeconomic models the generality of the prefetching approach suggests that the methods should be useful in other contexts. Estimation of long memory time series models and α-stable distributions using the RWMH algorithm are direct examples (Brockwell (2006); Lombardi (2007)). For latent variable models we argue that prefetching is suitable in conjunction with marginalisation techniques whereas other parallel approaches are required with sampling schemes based on data augmentation and Gibbs updates. To illustrate this further we discuss how to apply prefetching methods to the hierarchical Bayesian models with Gaussian Markov Random Field (GMRF) components encountered, for example, in spatial statistics (Knorr-Held and Rue (2005)).

In section 4.2 a brief overview of parallel MCMC is given. In section 4.3 terminology is introduced and the main ideas are conveyed via some simple examples before the algorithms are presented. In section 4.4 the prefetching algorithms are illustrated using two example problems. First the algorithms are compared using a well-known medium-scale macroeconomic model due to Smets and Wouters (2003) and the potential gains from using prefetching methods in the context of Bayesian estimation of large-scale linearised DSGE models are discussed. Second the algorithms are applied to the nonlinear estimation of a small DSGE model in an easy-to-use personal high performance computing (PHPC) environment, using Matlab on a multi-core computer. Finally, in section 4.5 it is illustrated how prefetching can be combined with lower level parallelism to increase the overall parallel efficiency of an estimation algorithm.

4.2 A brief overview of parallel MCMC

Parallel algorithms require the existence of independent tasks that can be performed concurrently. The granularity, or size, of the tasks determines to what extent an algorithm can be successfully parallelised. Bayesian inferential techniques fit unequally well with the parallel paradigm. Nonsequential methods, e.g. importance sampling, are better suited for parallelisation than sequential methods, such as Gibbs sampling. The relative merits of different sampling algorithms thus change in a parallel setting. This becomes apparent below when the random walk prefetching and parallel independence chain Metropolis-Hastings algorithms are compared. Furthermore, as we demonstrate in this paper the optimal scaling of the proposal density of the RWMH algorithm changes in a parallel computing environment, since statistical efficiency is no longer the sole concern.

The most obvious approach to parallel Metropolis-Hastings is simply to run multiple chains in parallel. This parallel chains approach does not require any parallel programming although a parallel program could be used to automise the procedure

of running the same program with different inputs on several machines (Rosenthal (2000); Azzini et al. (2007)).

In some situations it may also be of interest to parallelise a single chain. Poor mixing and long burn-in times are factors which increase the attractiveness of single-chain parallelisation (Wilkinson (2006)). More generally, MCMC methods are computationally intensive for a wide range of models and can require days or even weeks of execution time.

Parallelisation of a single chain can be divided into *within* and *between draw* parallelisation. The former includes, necessarily problem-specific, parallelisation of the likelihood evaluation, since typically this is the computationally challenging part of the posterior evaluation (Strid (2007*b*)). The blocking strategy based on exploitation of the underlying model's conditional independence structure also falls into this category (Wilkinson (2006); Whiley and Wilson (2004)). In both these approaches several processors collaborate to obtain a draw from the posterior. Between draw parallelism has been given the name prefetching (Brockwell (2006)). In the one-block prefetching approach each processor works independently on a posterior evaluation.

Parallel independence chain Metropolis-Hastings (ICMH) and parallel approaches based on regeneration are important special cases (Brockwell and Kadane (2005)). From a parallel computing perspective both these methods are largely equivalent to the parallel chains approach. Any other single-chain parallel algorithm requires frequent communication between processors. The precise nature of the parallel computer then determines whether an algorithm can be successfully used. Within-draw parallel algorithms are expected to be (typically substantially) more communication intensive than the prefetching algorithm. Hence if a particular parallel computer is deemed unsuitable for the prefetching algorithm, e.g. because the interconnection network is too slow in relation to the capacity of processors, then it should be even more inappropriate for any other, competing, within-draw single-chain parallel algorithm.

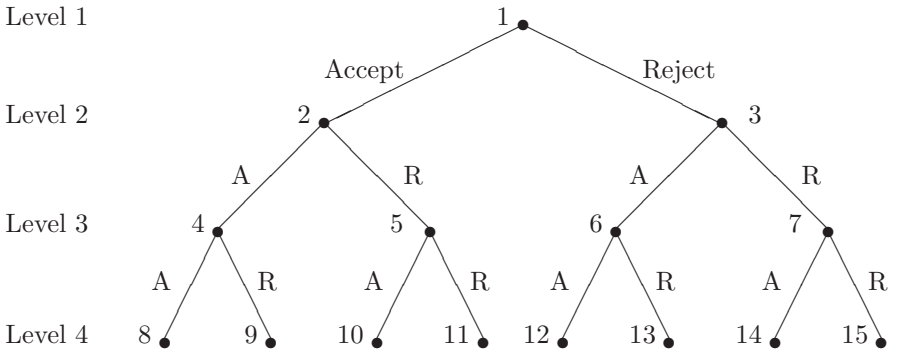
4.3 Prefetching

4.3.1 Terminology

The objective is to generate draws, $\left\{\theta^i\right\}_{i=1}^{R}$ where R is the length of the chain, from a posterior distribution, using the one-block Metropolis-Hastings prefetching algorithm. Throughout the paper we use the convention that parameters with superindices refer to draws whereas parameters with subindices refer to proposals. The assumption that all parameters in the vector θ are updated jointly, i.e. in one block, allows for a simple exposition of concepts.

The prefetching algorithm can be pictured using a tree; here called a Metropolis tree (figure 4.1). The nodes in the tree represent the *possible* future states of the chain. The number of *levels* of the tree, K, is related to the number of nodes, M, through $M = 2^K - 1$. The branches represent the decisions to accept or reject a proposal. Accepts are pictured as down-left movements and rejects as down-right movements in the tree.

Figure 4.1 Four-level Metropolis tree.



A node in the tree is associated with a state of the chain and a proposal based on this state. An evaluation of a proposed parameter will always occur at the start node $i_1 = 1$. This corresponds to the evaluation required for a serial Metropolis-Hastings algorithm. The state of the chain at node i_1 is θ^i, assuming that i draws have been obtained previously, and the proposed parameter is $\theta_1 \backsim f\left(.|\theta^i\right)$. If θ_1 is accepted the chain moves from node 1 to node 2, with state $\theta^{i+1} = \theta_1$, and otherwise the chain moves to node 3 and $\theta^{i+1} = \theta^i$. More generally the states of the chain at nodes i_p and $2i_p + 1$ are the same whereas the proposed parameter is unique to each node.

The number of processes/processors (the terms are used interchangeably in this paper) is given by P. A description of the P nodes and the associated *proposed* parameters, at which the posterior is evaluated in parallel, is called a *tour* of size P

$$T(P) = \{i_1, i_2, \ldots, i_P;\ \theta_{i_1}, \theta_{i_2}, \ldots, \theta_{i_P}\}.$$

Often we refer to a tour merely by the node indices, assuming that it is understood how to determine the parameter points. The number of draws, $\tilde{D}(P,T)$, produced by a tour is a stochastic variable with support $1, \ldots, P$. A chain consists of the draws obtained from a sequence of tours, $\{T_n\}_{n=1}^{N}$, and the last parameter reached by a tour becomes the starting point for the next tour.

For an even node i_p the *parent node* is $\frac{i_p}{2}$ and for an odd node it is $\frac{i_p-1}{2}$. If a node belongs to a tour, then its parent must also belong to the tour. The expected number of draws obtained from a tour of size P is denoted $D(P,T)$ and it is given by

$$D(P,T) = \sum_{p=1}^{P} \Pr(i_p), \tag{4.1}$$

where $\Pr(i_p)$ is the probability of reaching node i_p and trivially $\Pr(1) = 1$. The

expected number of draws of the *optimal tour*, the tour that maximises $D(P,T)$ conditional on the branch probabilities in the Metropolis tree, is denoted $D(P)$.

For some of the algorithms below the expected draws per tour can be obtained exactly whereas for others it is estimated via the average

$$\bar{d} = \frac{1}{N}\sum_{n=1}^{N} d_n, \tag{4.2}$$

using the output of the Metropolis-Hastings prefetching algorithm, where $\{d_n\}_{n=1}^{N}$ are the realised number of draws from the tours $\{T_n\}_{n=1}^{N}$ and

$$\bar{d} \xrightarrow{p} D,$$

by a law of large numbers. To obtain a chain of length R the required number of tours is $N = ceil\left[R/\bar{d}\right]$ and $\tilde{R} = NP > R$ posterior evaluations occur in total. Thus $\tilde{R} - R$ of the posterior evaluations are useless.

Expected draws per tour, $D(P)$, is equivalent to the theoretical speedup of the algorithm, i.e. it is the speedup in the absence of communication costs and other factors which affect parallel efficiency in practice. Draws per tour therefore provides an upper bound on the observed relative speedup

$$S(P) = \frac{T(1)}{T(P)}, \tag{4.3}$$

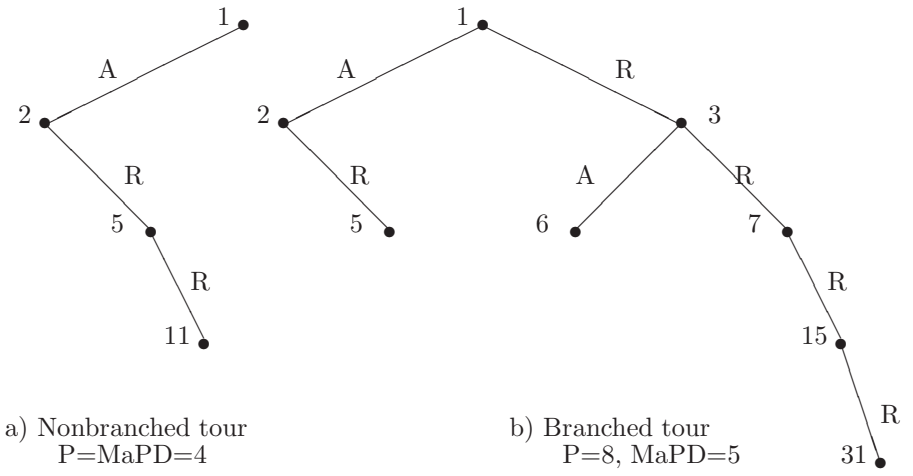
where $T(p)$ is the time of executing the parallel program using p processors in a particular hardware and software environment. The difference $D(P) - S(P) > 0$ thus depends on the precise nature of the parallel computing environment.

The *maximum possible depth* (MaPD) is the maximum number of draws that can be obtained from a tour, such that $MaPD \leq P$. Also, let $L = L(i_p)$ be the function that maps Metropolis tree indices to levels, e.g. $L(7) = 3$. A *branched tour* is a tour for which $MaPD < P$. In other words it is a tour with the property that at least two nodes at the same level of the Metropolis tree are evaluated, i.e. $L(i_p) = L(i_{\tilde{p}})$ for some pair $i_p, i_{\tilde{p}} \in T$. A *nonbranched tour* satisfies $MaPD = P$, or alternatively $L(i_p) = P$ for exactly one $i_p \in T$. The concepts are illustrated in figure 4.2.

A *static prefetching algorithm* is an algorithm where all tours consist of the same nodes, i.e. the indices $i_1, \ldots, i_P$ are constant across tours. For a *dynamic prefetching algorithm*, on the other hand, the nodes of the tour vary as we move through the chain.

Parallel random number generation is not an issue in our context since naturally all random numbers are generated by one process. The random numbers should be generated such that the chain is independent of the number of processors used. If this is the case we say that the random numbers are *draw-specific* and it is assured that prefetching does not affect the statistical properties of the chain. This is an implementational issue. The attachment of random numbers to draws implies that if tour $n-1$ produces $d_{n-1} < P$ draws then the random numbers attached to levels $d_{n-1}+1, \ldots, P$ of the Metropolis tree, i.e. the levels that are not reached by the tour, should instead be used in tour n.

Figure 4.2 Tours.



4.3.2 Examples

EXAMPLE 1 Assume that two processors, $P1$ and $P2$, are available for implementation of the random walk Metropolis-Hastings (RWMH) algorithm. The first processor is necessarily employed for evaluation of the posterior kernel p at

$$\theta_1 = \theta^i + \epsilon^{i+1},$$

where θ^i is the current state of the chain.

The second processor, $P2$, can be used either to prepare for an accept in the first stage, i.e. for evaluation of the posterior at

$$\theta_2 = \theta_1 + \epsilon^{i+2} = \theta^i + \epsilon^{i+1} + \epsilon^{i+2},$$

or to prepare for a reject and evaluate the posterior at

$$\theta_3 = \theta^i + \epsilon^{i+2}.$$

Assume that the acceptance probability can be chosen approximately as α by proper selection of the increment density $g(\epsilon)$, e.g. by scaling the covariance matrix of a Normal proposal density appropriately. Now, if $\alpha < 0.5$ it is obviously optimal to use $P2$ to prepare for a reject at the first stage (R1) and the optimal tour is $T(2) = \{1, 3\}$.

If the first proposal is rejected two draws are obtained in one time unit using the two processors. If, on the other hand, θ_1 is accepted the posterior evaluation at θ_3 is

useless. In this case one draw is obtained in one time unit using two processors. The expected number of draws per time unit, or the theoretical speedup, is

$$D\left(2,\{1,3\}\right)=\alpha+2\left(1-\alpha\right)=2-\alpha.$$

This is an example of static prefetching since the tours that constitute the chain will contain the same node indices. It should also be noted that if instead $\alpha>0.5$ the analysis is symmetric. In the remainder of the paper we restrict attention to the case $\alpha<0.5$. The reason for this is our focus on the random walk variant of the Metropolis-Hastings method and it is further motivated by the discussion in section 4.3.3 below.

EXAMPLE 2 In the case of three processors, $P1$,$P2$ and $P3$, assume $\alpha<0.5$ as before. The optimal allocation of $P1$ and $P2$ is clearly the same as in the example above. For the third processor there are three options. First it can be employed for evaluation of

$$\theta_7=\theta^i+\epsilon^{i+3},$$

i.e. anticipating rejects in the first two stages (R1,R2). A second possibility is to evaluate θ_2 in anticipation of an accept at the first stage. This yields the branched tour $\{1,2,3\}$ and $D\left(3,\{1,2,3\}\right)=2$ draws are obtained with certainty, since $P2$ was used to prepare for a reject at the first stage.

The final possibility is to prepare for a reject in the first stage followed by an accept in the second stage (R1,A2) and evaluate

$$\theta_6=\theta^i+\epsilon^{i+2}+\epsilon^{i+3}.$$

In the first case (R1,R2) the expected number of draws is

$$D\left(3,\{1,3,7\}\right)=\alpha+2\left(1-\alpha\right)\alpha+3\left(1-\alpha\right)^2. \tag{4.4}$$

For example, with $\alpha=0.25$ this yields 2.31. It is optimal to choose the nonbranched tour $\{1,3,7\}$ if

$$D\left(3,\{1,3,7\}\right)>D\left(3,\{1,2,3\}\right)=2,$$

which is if

$$\alpha\leq\frac{3-\sqrt{5}}{2}\approx 0.38.$$

Note that

$$D\left(3,\{1,3,7\}\right)-D\left(3,\{1,3,6\}\right)=\left(1-\alpha\right)\left(1-2\alpha\right)>0,$$

for $\alpha<0.5$ so it is never optimal to prepare for (R1,A2).

Again, if instead $\alpha>0.5$ is assumed the optimal tour for $\alpha\geq\frac{\sqrt{5}-1}{2}\approx 0.62$ would be given by $\{1,2,4\}$ due to the symmetry.

EXAMPLE 3 In the random walk Metropolis-Hastings algorithm a proposal at stage $i+1$ is accepted with probability $\alpha^{i+1} = \min\left\{X^{i+1}, 1\right\}$, i.e. if $u^{i+1} < X^{i+1}$ for some $u^{i+1} \sim U[0,1]$ where

$$X^{i+1} = \frac{p\left(\theta^i + \epsilon^{i+1}\right)}{p\left(\theta^i\right)}.$$

Consider example 1 again and assume that $u^{i+1} = 10^{-5}$ where u^{i+1} is the realised uniform random number associated with draw $i+1$. Clearly, in this hypothetical situation it is wise to use $P2$ to prepare for an accept, assuming no information about the posterior ratio X^{i+1} is available. The sequence of uniform random numbers, $\left\{u^i\right\}_{i=1}^R$ which are fixed from the outset, thus carries some information on how to best structure prefetching.

EXAMPLE 4 The acceptance rate, α, and the sequence of uniform random numbers, $\left\{u^i\right\}_{i=1}^M$, can be used to improve the efficiency of prefetching. Finally, knowledge about the posterior, p, can be used to make better predictions on where the chain is moving. Assume that an approximation to the posterior, p^*, is available and that the evaluation of p^* takes a fraction of the time to evaluate p. Then this approximate distribution can be used to suggest the states which are likely to be visited subsequently.

In the limiting case when $p^* = p$ we say that prefetching is perfect; it is analogous to importance sampling using the posterior as the importance function. In other words it is a situation when neither prefetching or importance sampling is necessary.

4.3.3 Algorithms

One-block random-walk Metropolis-Hastings prefetching

In this section the one-block Metropolis-Hastings prefetching algorithm is presented. This algorithm applies when all the parameters in θ are updated jointly and the proposal density f depends on the current state of the chain, e.g. for the one-block random walk Metropolis-Hastings sampler which we focus on here. It would be possible to use other proposal densities, e.g. a more general autoregressive proposal, but this route is not pursued here. Multiple block prefetching is discussed briefly at the end of this section and in appendix B.

The algorithm assumes that the posterior evaluation time is not parameter dependent and it is intended for application on a homogeneous cluster. The key assumption for application of the algorithm in practice is, loosely speaking, that the time of a posterior evaluation is significant in relation to the other steps of the algorithm.

Algorithm 13 *Metropolis-Hastings prefetching algorithm*

1. Choose a starting value θ^0. Set the draw counter $S_0 = 0$.

2. (Prefetching step) Assume that the chain is in state $\theta^{S_{n-1}}$ when the n^{th} tour begins, where S_{n-1} is the state of the draw counter after $n-1$ completed tours. Construct a tour, $T_n = \{i_1, \ldots, i_P; \theta_{i_1}, \theta_{i_2}, \ldots, \theta_{i_P}\}$ (serial).

3. Distribute $\theta_{i_p}, p = 1, \ldots, P$, to the processes (*scatter*).

4. Evaluate $p\left(\theta_{i_p}\right)$ in parallel.

5. Return $p\left(\theta_{i_p}\right)$ (*gather*).

6. (Metropolis-Hastings step) Run the Metropolis-Hastings algorithm for the tour T to generate D_n draws, where $1 \leq D_n \leq P$. Update the draw counter, $S_n = S_{n-1} + D_n$, and assign the starting state for the next tour, θ^{S_n} (serial).

7. Go to 2. Stop when $S_n \geq R$ draws from the posterior have been obtained. ⊡

The posterior evaluations, step 4 above, are performed in parallel. Communication between processes takes place in steps 3 and 5 and message passing collective communication tokens are used to describe the required operations. Steps 2 and 6 are performed by a master process or are replicated by all processors.

The Metropolis-Hastings step, step 6 in the above algorithm, is implemented in the following way for the random walk algorithm:

Algorithm 14 *Random walk Metropolis-Hastings step*

1. Set $j = 1$. The current state of the chain is $\theta^{S_{n-1}}$, where S_{n-1} is the state of the draw counter after $n-1$ completed tours. Let $\theta_{i_1} = \theta_1$ be the proposal conditional on the current state, i.e. $\theta_1 = f\left(.|\theta^{S_{n-1}}\right) = \theta^{S_{n-1}} + \epsilon^{S_{n-1}+1}$.

2. The current node is i_p, with associated state $\theta^{S_{n-1}+j-1}$, and its level is $L\left(i_p\right) = j$. If
$$u^{S_{n-1}+j} < \alpha^{S_{n-1}+j} = \min\left\{1, \frac{p\left(\theta^{S_{n-1}+j-1} + \epsilon^{S_{n-1}+j}\right)}{p\left(\theta^{S_{n-1}+j-1}\right)}\right\}, \tag{4.5}$$
where $u = U[0,1]$, accept the draw and set $\theta^{S_{n-1}+j} = \theta^{S_{n-1}+j-1} + \epsilon^{S_{n-1}+j}$. Otherwise set $\theta^{S_{n-1}+j} = \theta^{S_{n-1}+j-1}$.

3. If (i) the proposal $\theta_{i_p} = \theta^{S_{n-1}+j-1} + \epsilon^{S_{n-1}+j}$ was accepted and $i_{\tilde{p}} = 2i_p$ or (ii) if θ_{i_p} was rejected and $i_{\tilde{p}} = 2i_p + 1$ for some node $i_{\tilde{p}} \in T$ then move to $i_{\tilde{p}}$. In this case set $j = j + 1$ and the current node to $i_{\tilde{p}}$ and return to 2. Otherwise exit. ⊡

The output from algorithm 14 are the D_n draws from the tour n, $\theta^{S_{n-1}+1}, \ldots, \theta^{S_n}$, and the output from the prefetching algorithm consists of the draws

$$\left\{\theta^{S_{n-1}+1}, \ldots, \theta^{S_n}\right\}_{n=1}^{N} = \left\{\theta^1, \theta^2, \ldots, \theta^{S_N}\right\},$$

where N is the total number of completed tours and

$$\sum_{n=1}^{N} D_n = S_N \geq R.$$

The remaining discussion in this section focuses on the prefetching step (step 2 of algorithm 13). The prefetching, or tour construction, problem consists of two separate parts:

1. Determine a rule for calculating the probabilities of reaching the nodes in the Metropolis tree based on some set of information.
2. Find the tour that maximises the expected number of draws per tour, D, based on these probabilities. We call this the *optimal tour*, with the implicit understanding that optimality is conditioned on the specific rule used.

The second task is easy and an algorithm which constructs the optimal tour given a rule for assigning the probabilities of the Metropolis tree is presented in appendix A. Below a parallel independence chain Metropolis-Hastings algorithm is first presented and then five variants of prefetching are discussed, named as follows: (1) Basic prefetching, (2) Static prefetching, (3) Dynamic prefetching using the sequence of uniform random numbers, (4) Dynamic prefetching using a posterior approximation and (5) Most likely path. Finally we discuss prefetching with multiple blocks and other proposal densities.

Independence chain Metropolis-Hastings

In the special case when the proposal density is fixed

$$f\left(\theta|\theta^i\right) = f\left(\theta\right),$$

parallelisation is simplified further since there are no dependencies between posterior evaluations. Let

$$R = \sum_{p=1}^{P} R_p,$$

where R_p is the number of posterior evaluations performed by process p and R is the length of the chain.

Algorithm 15 *Parallel Independence Chain Metropolis-Hastings (ICMH) algorithm*

1. Each process p generates $\theta_p = \left\{\theta_{1p}, \theta_{2p}, \ldots, \theta_{R_p p}\right\}$ where

   $$\theta_{ip} \sim f, \ i = 1, \ldots, R_p,$$

   and collects the values of the posterior evaluated at these parameters in the vector $p_p = \left\{p\left(\theta_{1p}\right), p\left(\theta_{2p}\right), \ldots, p\left(\theta_{R_p p}\right)\right\}$ (parallel).

2. The master process gathers θ_p and p_p, $p = 1, \ldots, P$ (*gather*).

3. Run the Metropolis-Hastings algorithm with the posterior already evaluated at R parameter values (serial). ⊡

In this algorithm the master process only collects the results from the other processes once. If memory limitation is a concern it can be solved by collecting the local results more often. In the case of inhomogeneous processors load balance is easily restored by allowing R_p to vary across processors.

The parallel ICMH algorithm is *embarrassingly parallel* both in the sense that it is simple to implement and because there is no essential dependency between the parallel tasks. As a consequence this algorithm will yield close to linear speedup, i.e. $S(P) \approx P$, on *any* homogeneous parallel computer. The prefetching RWMH algorithm is also simple to implement but good parallel performance for this algorithm requires a balance between the problem (or problem size) and hardware/network since communication is frequent.

Basic prefetching

The basic prefetching algorithm suggested by Brockwell (2006) has the property that all future states at the same level in the Metropolis tree are treated as being equally likely to be reached. No information is used to structure prefetching. In our framework this algorithm is obtained if every branch in the Metropolis tree is assigned the probability 0.5.

The basic prefetching tour evaluates the proposed parameters of all nodes up to a given level of the Metropolis tree, i.e. $T(P) = \{1, 2, \ldots, P\}$, and thus produces a certain number of draws

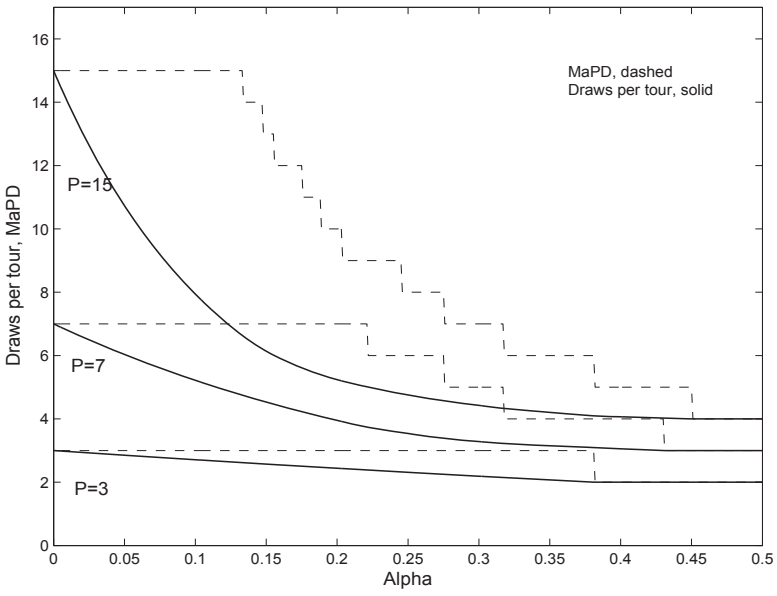
$$D(P) = MaPD = \log_2(P + 1). \tag{4.6}$$

This approach provides a lower bound on the scalability that can be achieved using prefetching. Note that if the number of processors P does not correspond to the values $2^p - 1, p = 1, 2, \ldots$ there is no clear guide on how to select the 'surplus' nodes for evaluation.

Static prefetching

The researcher can typically target an acceptance rate, α, with good precision by selection of the proposal density, f, in the Metropolis-Hastings algorithm. Knowing α it is easy to improve on the basic algorithm. Let $T(P|\alpha, \tilde{\alpha})$ denote the tour when the acceptance rate is α and the 'perceived' acceptance rate used to construct the tour is $\tilde{\alpha}$, thus allowing for imperfect targeting. The static prefetching tour is obtained by attaching the probability $\tilde{\alpha}$ to all accept (down-left) branches of the Metropolis tree and $1 - \tilde{\alpha}$ to reject (down-right) branches. This was explained in some detail in examples 1 and 2 above for the cases $P = 2$ and $P = 3$. The nonbranched tour shown in figure 4.2 is obtained if we choose $P = 8$ and target, for example, $\tilde{\alpha} = 0.25$.

Figure 4.3 Static prefetching performance, $\tilde{\alpha} = \alpha$.

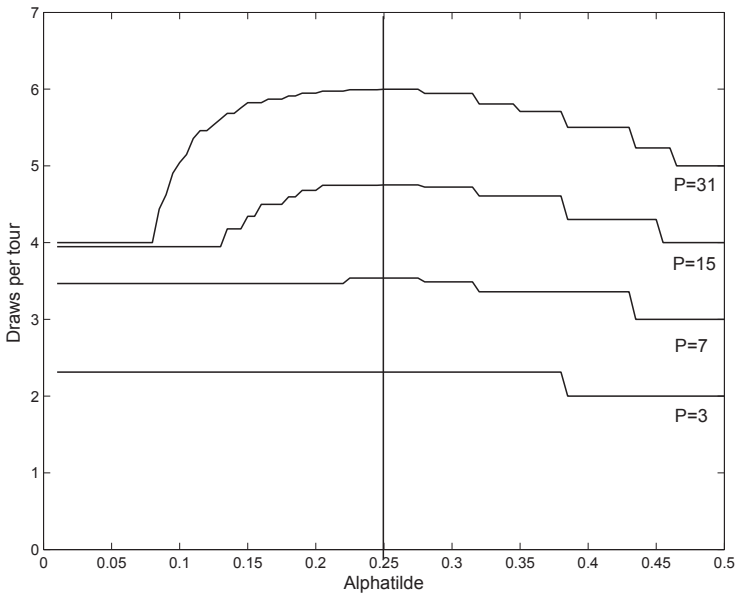


The node indices of the tours only needs to be obtained once per chain and the approach is general, i.e. model independent, since it only depends on the acceptance rate. For large enough α and/or P the tour will be branched.

In figure 4.3 the expected number of draws per tour, $D(\alpha, P)$, and the maximum possible depth (MaPD) of the optimal static prefetching tour is plotted against the acceptance rate $\tilde{\alpha} = \alpha \in (0, 0.5)$ for $P = 3, 7, 15$. The expected number of draws decreases smoothly in α and above some threshold $\alpha_h(P)$ it becomes optimal to choose the basic prefetching tour. If α is below some threshold $\alpha_l(P)$ the optimal tour is nonbranched and an 'always prepare for a reject' strategy is chosen. The MaPD curve jumps at points where the optimal tour changes.

What happens if the tour is constructed based on a perceived acceptance rate $\tilde{\alpha}$ when the true acceptance rate is in fact α? In figure 4.4 the expected number of draws is plotted for $\tilde{\alpha} \in (0, 0.5)$ and $P = 3, 7, 15, 31$ when the true acceptance rate is $\alpha = 0.25$. It is seen that it is optimal to use $\tilde{\alpha} = \alpha$ in constructing the tour but the optimal choice of $\tilde{\alpha}$ is not unique. For example, as was seen in example 2 above, in the case of three processors any $\tilde{\alpha} \in (0, 0.38)$ will suggest the nonbranched tour $T(3|0.25, \tilde{\alpha}) = \{1, 3, 7\}$ which is the optimal tour if $\alpha = 0.25$.

Figure 4.4 Static prefetching performance. The true acceptance rate is $\alpha = 0.25$ and tours are constructed based on $\tilde{\alpha}$.



Optimal static prefetching

A parallel efficiency perspective obviously suggests targeting a low acceptance rate, as seen in figure 4.3. Intuitively it becomes very easy to predict where the chain is moving when α is small.

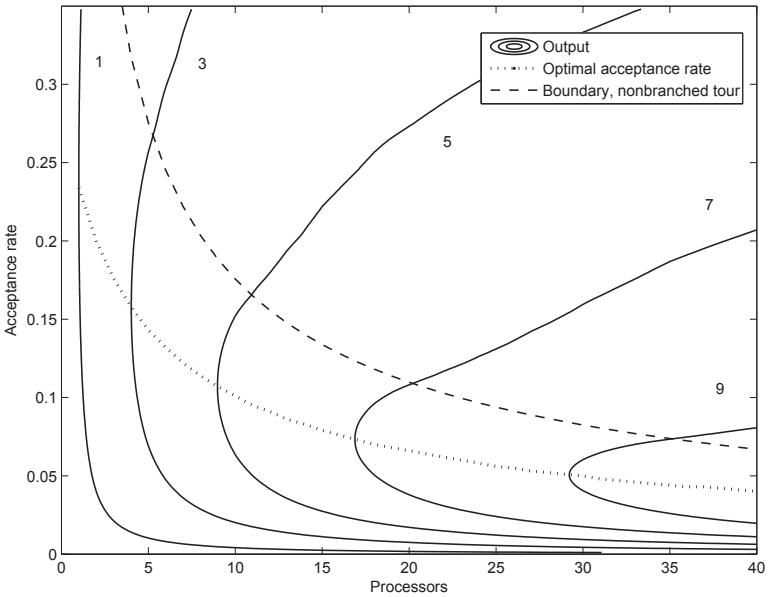
The optimal acceptance rate, $\alpha_{opt,1}$, for the random walk Metropolis-Hastings algorithm has been derived under various assumptions on the target density p (Roberts et al. (1997); Roberts and Rosenthal (2001)). Here we show how the optimal acceptance rate depends on the number of processors in a parallel computing framework, thus considering jointly Markov chain, or statistical, efficiency and parallel efficiency of the static prefetching RWMH algorithm.

We revisit the special case where the posterior density has the product form

$$p(\theta) = \prod_{i=1}^{k} \tilde{p}(\theta_k),$$

and the increment density, g, is of the form $g(\epsilon) = N\left(0, I_k \sigma_k^2\right)$ where $\sigma_k^2 = l^2/k$ and l is the scaling parameter. In the high-dimensional limit, i.e. as $k \longrightarrow \infty$, and under certain regularity conditions on $\tilde{p}$ it can be shown that the Markov chain converges to a diffusion process and the optimal acceptance rate is obtained by maximising the

Figure 4.5 Optimal static prefetching acceptance rates.



'efficiency function'

$$E^1(\alpha) \propto \alpha \times \left[\Phi^{-1}\left(\frac{\alpha}{2}\right)\right]^2, \tag{4.7}$$

where Φ is the standard normal cumulative distribution function. This yields the famous result of an optimal acceptance rate $\alpha_{opt,1} = \arg\max_\alpha E^1(\alpha) \approx 0.234$ (see Theorem 1 in Roberts and Rosenthal (2001) and the subsequent discussion).

In figure 4.5 the joint output measure

$$E(\alpha, P) = E^1(\alpha) \times D(\alpha, P) \times c, \tag{4.8}$$

is plotted as a function of the acceptance rate α and the number of processors P. The constant c is chosen such that $E(\alpha_{opt,1}, 1) = 1$. The figure is interpreted as follows: the *time* needed to achieve a given accuracy in estimating any function $h(\theta)$ of the posterior is inversely related to the output E.

The *optimal acceptance rate in a parallel setting* is the acceptance rate which minimises the time of obtaining a sample of a fixed size and quality, e.g. a fixed number of iid draws, from the posterior, conditional on the number of processors used. It is seen in figure 4.5 that the optimal acceptance rate implies an 'always prepare for a reject' prefetching strategy. The optimal rate is below the nonbranched tour boundary which is defined implicitly by $\alpha = (1-\alpha)^{P-1}$. For example, using $P = 7$ processors the optimal acceptance rate is $\alpha_{opt,7} = 0.120$ and a sample of a given size

and quality from the posterior can be obtained $E(\alpha_{opt,7}, 7) = 4.3$ times faster than when applying one processor and the optimal acceptance rate $\alpha_{opt,1} = 0.234$.

Our choice of c implies that $E(\alpha_{opt,P}, P)$ is interpreted as the *optimal speedup* of the static prefetching RWMH algorithm and it satisfies

$$D(\alpha_{opt,1}, P) \leq E(\alpha_{opt,P}, P) \leq D(\alpha_{opt,P}, P),$$

where equality holds for $P = 1$. For the case of $P = 7$ processors we have $E(\alpha_{opt,1}, 7) = D(\alpha_{opt,1}, 7) = 3.65$. The benefit of lowering the acceptance rate from $\alpha_{opt,1} = 0.234$ to $\alpha_{opt,7} = 0.120$, the gain in parallel efficiency, is larger than the cost, the loss of statistical efficiency.

More generally the optimal acceptance rate $\alpha_{opt,P}$ will be determined by the curvature of the efficiency function $E^1(\alpha)$. In a serial computing setting a flat efficiency curve implies that it is 'of little value to finely tune algorithms to the exact optimal values'(Roberts and Rosenthal (2001)). In a parallel computing setting the implication is that parallel efficiency can be improved, by lowering the acceptance rate, without incurring a large cost in terms of statistical efficiency.

In our applications below the efficiency function $E^1(\alpha)$ is not available analytically and hence the optimal acceptance rate $\alpha_{opt,P}$ cannot be solved for. Instead we target an acceptance rate $\alpha = 0.25$ for all P, a 'conservative' choice, and compare the parallel efficiency of different prefetching approaches while keeping the statistical properties fixed. In the main illustration the empirical static prefetching optimal speedup and the associated empirical optimal acceptance rates are also obtained.

Finally, in the processor limit

$$\lim_{P \longrightarrow \infty} \left\{ \arg\max_{\alpha} E(\alpha, P) \right\} = \lim_{P \longrightarrow \infty} \alpha_{opt,P} = 0.$$

In other words we can afford to make the algorithm arbitrarily poor, in the statistical sense, only if parallel computing technology is arbitrarily cheap.

Dynamic prefetching based on the uniform random numbers

The probability of accepting a proposal θ_z drawn from $f\left(\theta|\theta^i\right)$ depends on the realised value of a uniform random number, u^{i+1}. Let

$$X^{i+1} = \frac{p(\theta_z)}{p(\theta^i)},$$

and consider the conditional acceptance probability

$$\alpha^{i+1}\left(u^{i+1}\right) = \Pr\left(u^{i+1} < X^{i+1} | u^{i+1}\right),$$

where u is treated as fixed and X as stochastic. Since it is possible to characterise the relationship between α and u the static prefetching algorithm can be improved upon.

In the applications below we incorporate information from the sequence of realised uniform random numbers in the following way. First R draws from the Metropolis-Hastings sampler are obtained. Next, the unit interval is divided into K subintervals I_k and an empirical acceptance rate

$$\alpha_k = \frac{\#\left\{\{u < X\} \cap \{u \in I_k\}\right\}}{\#\left\{u \in I_k\right\}}, \tag{4.9}$$

is calculated for each subinterval $k = 1, \ldots, K$ based on approximately R/K draws. The constant acceptance rate used in the static prefetching algorithm above is then replaced by an acceptance probability which depends on a uniform random number

$$\alpha^{i+1}\left(u^{i+1}\right) = \sum_{k=1}^{K} \alpha_k I\left(u^{i+1} \in I_k\right), \tag{4.10}$$

where $I_k = [\frac{(k-1)}{K}, \frac{k}{K}]$ and I is the indicator function. This algorithm has the property that the branch probabilities in the Metropolis tree are level-specific, i.e. all accept branches at the same level in the tree will have the same probability attached to them.

The procedure is illustrated in figure 4.6 where the acceptance rate is plotted against the uniform random number for the linear estimation example presented later in the paper. Here $R = 100,000$ draws are used to obtain estimates of α_k for $K = 20$ equally sized subintervals. For example it is seen that $\alpha_1 = 0.60$ is the estimate of $\Pr\left(u < X | u \in [0, 0.05]\right)$.

Dynamic prefetching based on a posterior approximation

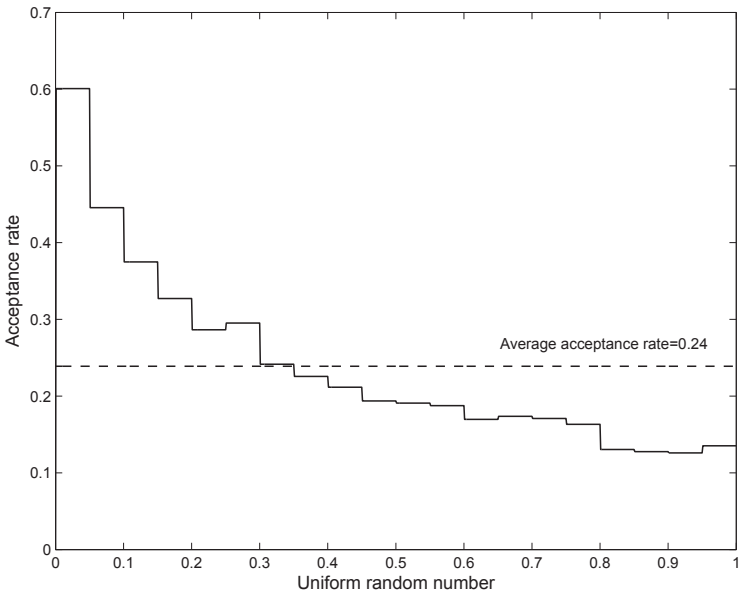
If a posterior approximation, p^*, is available and if evaluation of p^* is fast in comparison with the posterior kernel p a possibility is to use p^* to determine at which parameters to evaluate p. The objective is thus the same as in importance sampling or independence chain Metropolis-Hastings (ICMH) sampling, i.e. to find a good approximation to the posterior. However, a difference is that the quality of the approximating density used for prefetching has no implications for the statistical analysis; it will 'merely' affect computational efficiency.

For the RWMH algorithm one possible strategy is to simply replace the constant α in the construction of the static prefetching tour with the probability

$$\min\left\{\beta, \frac{p^*\left(\theta_{i_p}\right)}{p^*\left(\theta^i\right)}\right\},$$

where $\theta_{i_p} = \theta^i + \epsilon$ and thus incorporate information about the posterior in the construction of the tour. We call this *dynamic prefetching using a posterior approximation*. If $\beta < 1$ then $\Pr\left(i_p\right) > 0$ for all p, i.e. all nodes of the Metropolis tree have positive probability of being visited. In the application below we let $\beta = 1$, thus eliminating parts of the tree. As in the static prefetching case the tour will in general be branched.

Figure 4.6 Acceptance rate as a function of the uniform random number in the estimation of a linear DSGE model.



Another possibility is to run the chain P steps ahead using the approximate density, thereby also incorporating the information contained in the sequence of realised uniform random numbers. The states visited using the approximate posterior yield the parameter points at which the posterior p is subsequently evaluated in parallel. The tour is nonbranched and we call this type of algorithm a *most likely path algorithm*.

4.3.4 Marginalisation and blocking

The Dynamic Stochastic General Equilibrium (DSGE) models used for the illustrations in this paper are represented as state-space models. The coefficients of the DSGE-state-space model are nonlinear functions of the structural parameters, θ. The latent variables, collected in x, are integrated out of the joint posterior $p(\theta, x|y)$ using filtering techniques and attention is restricted to the marginal posterior density, $p(\theta|y)$, where y denotes the data. Sampling from the conditional $p(\theta|x, y)$ is precluded for these models. In the main example, in section 4.4.1, the Kalman filter is applied for integration in a degenerate linear and Gaussian state-space (LGSS) model. If there is interest in the posterior distribution for the state variables, $p(x|y)$, this distribution is obtained via smoothing *after* $p(\theta|y)$ has been sampled, see e.g. Durbin and Koopman (2001). The marginalisation technique has also been used in other contexts, e.g.

Garside and Wilkinson (2004).

For comparison, consider the more typical LGSS model where interest centers on $p(\theta, x|y)$ and where, in contrast to the linearised DSGE model, the classic two-block data augmentation scheme is available. The conditional posterior $p(\theta|x, y)$ is sampled using Gibbs or Metropolis updates and $p(x|\theta, y)$ is sampled using a simulation smoother, e.g. Strickland, Forbes and Martin (2009). A Gibbs step, viewed as a Metropolis-Hastings update with acceptance probability 1, implies perfect prefetching and hence prefetching does not apply. In situations where a Metropolis-Hastings block sampler with reasonably good mixing properties is available we expect prefetching to be of limited use and, if thought necessary, other parallelisation approaches should be considered. Wilkinson (2006) shows how to exploit the conditional independence structure to parallelise the multimove simulation smoother used for estimation of the baseline stochastic volatility (SV) model. This approach can also be applied to extended SV models and the stochastic conditional duration (SCD) model, for which efficient block sampling schemes have been developed (Omori and Watanabe (2008); Takahashi, Omori and Watanabe (2009); Strickland, Forbes and Martin (2006)). A similar 'parallel blocks' approach is also applied by Whiley and Wilson (2004).

There are two features, in addition to the inability to sample from $p(\theta|x, y)$, which increase the attractiveness of the one-block prefetching approach for DSGE models. First, it is typically a nontrivial task, at least *a priori*, to split the parameter vector θ into blocks such that there is weak posterior dependency between blocks. Second, the resulting Metropolis sampler with B blocks is 'penalised' by a B-factor increase in computational time relative to the one-block sampler.

More generally, although prefetching generalises to the case of Metropolis multiple block sampling parallel efficiency is expected to suffer in that context, at least when a subset of the full conditionals can be sampled directly. If high dimensionality of the parameter vector, rather than exploitable structure, is the motive for splitting the parameters into blocks then Metropolis block prefetching *could* be interesting but this is not investigated further here. A brief description of block prefetching is given in appendix B.

Another potentially interesting application for prefetching methods, not fully explored in this paper, is the class of hierarchical Gaussian Markov random field (GMRF) models with a wide range of applications, e.g. in spatial statistics (Knorr-Held and Rue (2005)). Knorr-Held and Rue (2002) develop a 'hyperblock' sampler for this type of model, using the methods for fast sampling from GMRFs presented by Rue (2001). This is another, more elaborate, example of a marginal updating scheme where the hyperparameters θ and the latent field x are updated jointly using a Metropolis-Hastings step.

Knorr-Held and Rue (2002) demonstrate that the hyperblock sampler mixes more rapidly than the traditional single-site updating scheme and various intermediate blocking schemes in three typical disease mapping applications. The results are driven by the strong posterior dependency between θ and x in these applications. A one-block sampler for linear mixed regression models, along similar lines, is suggested by Chib and Carlin (1999). The marginalisation approach is also used by Gamerman, Moreira and Rue (2003) for spatial regression models and in Moreira and Gamerman

(2002) for time series models.

The possibility of prefetching parallelisation may be interpreted as a further argument in favour of the sophisticated one-block sampling scheme for GMRF models, in addition to the documented improvement in chain mixing. In fact, the 'always prepare for a reject' prefetching strategy applies to a more general one-block sampler, suggested by Wilkinson and Yeung (2004) for the related class of linear Gaussian Directed Acyclic Graph (DAG) models. In appendix B we outline how to apply prefetching in the GMRF context and also mention the possibility of combining prefetching with the parallel GMRF sampling approach suggested by Steinsland (2007).

4.4 Illustrations

4.4.1 Linear estimation of a DSGE model

Long burn-in times and poor mixing are factors that can motivate interest in single-chain parallel methods. In the area of Bayesian estimation of Dynamic Stochastic General Equilibrium (DSGE) models the one-block random walk Metropolis-Hastings algorithm is the most commonly used estimation method and our impression is that poor mixing of chains is a typical experience of researchers in this field. For an exception, see Belaygorod and Dueker (2006).

The performance of the five variants of the prefetching algorithm presented above and the parallel ICMH algorithm is compared using one of the core macroeconomic models at the European Central Bank, the Smets and Wouters (SW) model (Smets and Wouters (2003)). The model is chosen because it is well known and since it is the backbone of larger DSGE models currently developed at many central banks. A major determinant of the computational cost of estimating a DSGE model is the number of state variables of the model. Our version of the SW model has 15 state variables, including 8 shocks. Recently developed large-scale microfounded models in use at central banks have as many as 65 state variables and there is at least one very recent example of a model which contains more than 100 state variables (Adolfson, Laséen, Lindé and Villani (2007); Christiano, Trabandt and Walentin (2007)). In a relative sense, and using the jargon of parallel computing, this implies that our estimation problem can be viewed as fine-grained.

The empirical analysis of large-scale DSGE models is restricted by the computational cost of estimating them. Re-estimation is necessary as different model specifications are evaluated or when new data arrives. Furthermore, a central bank is facing real-time constraints on these activities. In our view the development of larger and more complex models and the unavailability of adequately efficient sampling methods increase the attractiveness of applying the parallel methods presented here.

Model, prior, solution and likelihood

The economic content of the model is largely unimportant for our evaluation of the parallel algorithms and therefore it is not presented here. Similar models have been

analysed and/or estimated in many articles (Smets and Wouters (2003); del Negro, Schorfheide, Smets and Wouters (2005)). The model used here is presented in detail in Strid (2007*a*).

The model consists of a set of nonlinear expectational equations describing the equilibrium and a linear approximation to the policy function of the model is obtained (the solution). The linear approximate policy function is represented as a linear state-space model:

$$X_t = T(\theta) X_{t-1} + R(\theta) \epsilon_t \tag{4.11}$$

and

$$Y_t = d(\theta) + Z X_t + v_t, \tag{4.12}$$

where [4.11] is the state equation and [4.12] the measurement equation. Here X_t (dimension n_x) is a vector containing the state variables and Y_t (n_y) is a vector containing the observed variables. The structural parameters of the model are collected in the vector θ (n_θ) and the coefficient matrices, T (which is dense), R and Z, and the vector d are nonlinear functions of θ. The innovations, ϵ_t (n_ϵ), and the measurement errors, v_t (n_v), are assumed to be independent and normally distributed, $\epsilon_t \backsim N(0, \Sigma_\epsilon)$ and $v_t \backsim N(0, \Sigma_v)$.

The likelihood evaluation consists of two parts. First the model is solved using one of the many available methods (Klein (2000)). Second the likelihood function of the model is evaluated using the Kalman filter, see e.g. Durbin and Koopman (2001). The prior distribution for θ is similar to the distributions typically used in the literature on estimation of New Keynesian models.

For the model estimated here we have the dimensions $n_x = 15, n_y = 5, n_\epsilon = 8$ and $n_\theta = 23$. The model is estimated on Euro Area data for the period 1982:1-2003:4 (88 observations). The five data series used for estimation are the short-term interest rate, inflation, output growth, the consumption-to-output ratio and the investment-to-output ratio.

Parallel approaches for linearised DSGE models

In the context of estimation of large-scale linearised DSGE models, what other strategies for single-chain parallelisation are available apart from prefetching? An obvious within-draw approach is to apply parallel matrix algorithms (e.g. using ScaLAPACK or PLAPACK) to the computations involved in Kalman filtering, primarily the matrix multiplications of the Riccati equation which account for a significant part of Kalman filtering time. However, if this approach is attempted it is crucial to apply parallel routines *whenever fruitful*, including in the solution algorithm. The extent to which a few standard matrix operations account for overall Kalman filtering and model solution time determines the parallel efficiency properties of the approach. Furthermore, it is hard to improve on optimal serial DSGE-specific Kalman filter implementations using parallel methods, see Strid and Walentin (2008).

Note again that it would be straightforward, at least in principle, to combine such local computations strategies, which are somewhat more demanding to implement, with prefetching.

Parallel efficiency: draws per tour

The posterior approximation used for prefetching (variants 4 and 5) is a normal approximation $p^* = N(\theta_m, \Sigma_m)$ where θ_m is the posterior mode, H_m the Hessian matrix at the mode and $\Sigma_m = H_m^{-1}$. The scaled inverse Hessian is also used as the covariance matrix for the proposal density in the random walk Metropolis-Hastings algorithm, i.e. proposals are generated with

$$\theta^z = \theta^i + \epsilon^{i+1},$$

where $\epsilon^{i+1} \sim N\left(0, cH_m^{-1}\right)$ and c is a scaling parameter.

Prior to optimisation of the posterior p and estimation using the RWMH algorithm some parameters in θ are reparameterised to make the parameter space unbounded. The reparameterisation serves two purposes: first, it makes optimisation easier and, second, the efficiency of sampling using the RWMH algorithm is improved because the approximation to normality is better. In the context of prefetching the reparameterisation will thus also improve the parallel efficiency of the algorithms which rely on the posterior approximation.

For each combination of prefetching algorithm and number of processors P the RWMH prefetching algorithm is used to sample $R = 500,000$ draws from the posterior distribution. In total the posterior is evaluated $\tilde{R} = NP = RP/\bar{d}$ times where N is the number of tours and $\bar{d}$ is the average number of draws per tour. In all estimations the chain is started at the posterior mode and $50,000$ draws are discarded as burn-in. A target acceptance rate $\tilde{\alpha} = 0.25$ is used to construct the tours in static prefetching (2) and dynamic prefetching based on the uniform random numbers (3) and the average acceptance rate α turns out to be 0.24. Later on we allow $\tilde{\alpha}$ to vary with P for the static prefetching algorithm. Detailed results on prefetching performance are presented for $P = 2^p - 1, p = 1, \ldots, 5$. These values are chosen because they allow for a neat comparison with the benchmark basic prefetching algorithm, which has integer draws per tour for these values of P (see expression [4.6]).

In table 4.1 the average number of draws per tour $\bar{d}$ are presented. It is seen, first, that static prefetching allows us to make easily reaped, although quite small, gains in comparison with the basic algorithm. Second, incorporating knowledge about the sequence of uniform random numbers lead to additional small gains. These efficiency improvements are model independent and can always be obtained.

Third, in our example the inclusion of information about the posterior via the normal approximation yields the largest gains. For example, using the most likely path algorithm (5) with $P = 7$ processors the extra draws, i.e. the draws which are added to the sure draw, is doubled in comparison with the basic algorithm (1). If a reasonable approximation to the posterior is available it thus appears possible to improve the parallel efficiency of prefetching quite substantially.

Finally, we report empirical measures of the static prefetching optimal speedup and optimal acceptance rates for this problem (6). Note that these quantities are adjusted for differences in sampling efficiency when targeting different acceptance rates. Thus direct comparison with the other approaches is possible. The experiments conducted to obtain these numbers are described in appendix A. Also note that we have not

Table 4.1 Performance of prefetching algorithms for the linearised DSGE model. Average number of draws per tour, $\bar{d}$.

	Processors					
Algorithm variant	1	3	7	15	31	∞
1. Basic prefetching	1	2	3	4	5	∞
2. Static prefetching	1	2.31	3.54	4.75	6.00	∞
3. Dynamic prefetching, uniform	1	2.38	3.81	5.04	6.24	∞
4. Dynamic prefetching, post. approx.	1	2.57	4.58	6.45	8.04	-
5. Most likely path	1	2.66	4.99	7.31	8.42	8.70
6. Optimal static prefetching speedup	1	2.44	4.30	6.55	6.55	-
Optimal empirical acceptance rate	0.22	0.16	0.13	0.09	0.05	

been able to increase the quality-adjusted speedup for the most likely path algorithm (5) by targeting lower acceptance rates in this way.

The results reported above must be interpreted with caution. If the parallel performance of the most likely path algorithm is becoming 'too good' it suggests that other approaches, such as independence chain Metropolis-Hastings (ICMH), should be considered. For our estimation problem we have verified that the sampling efficiency of ICMH using standard choices of proposal densities, e.g. the multivariate t, is inferior to the efficiency of RWMH. The acceptance rate of the best ICMH sampler is 13%. If the thinning factor for ICMH is roughly eight times the factor used with the RWMH sampler it is found that the relative numerical efficiencies, or the inefficiency factors, of the two approaches are roughly similar.

Parallel efficiency on an HPC cluster

Draws per tour, $D(P)$, is an abstract measure of scalability since the cost of constructing tours and communication cost is not taken into account. In order to assess the magnitude of the difference between theoretical and actual speedup, $D(P) - S(P)$, the algorithms are taken to a parallel computer. The prefetching algorithm is implemented using Fortran and the Message Passing Interface (MPI) and tested on the Lenngren cluster at the Center for Parallel Computers (PDC) at the Royal Institute of Technology (KTH) in Stockholm, a high performance computing (HPC) environment. The cluster uses Intel Xeon 3.4GHz processors connected by an Infiniband 1GB network. The MPI implementation on the cluster is Scali MPI. Further information on the performance characteristics of the cluster is available at *www.pdc.kth.se*.

For each combination of P and algorithm an acceptance rate $\tilde{\alpha} = 0.25$ is targeted and $R = 500,000$ draws from the posterior are obtained. In table 4.2 the draws per tour D and relative speedup S, as defined in [4.3], are reported for $P = 2^p, p = 0, \ldots, 5$ for the static prefetching algorithm (2) and the best performing algorithm according to table 4.1, the most likely path algorithm (5). Using one processor $R = 500,000$ draws are obtained in 80 minutes and using $P = 8$ processors the static prefetching algorithm (2) executes in 23 minutes and the most likely path algorithm (5) in 15

Table 4.2 Performance of prefetching algorithms for the linearised DSGE model on the Lenngren cluster.

Static prefetching						
Processors, P	1	2	4	8	16	32
Draws per tour, D	1	1.76	2.77	3.77	4.92	6.16
Relative speedup, S	1.00	1.72	2.70	3.46	4.61	5.18
Most likely path						
Processors, P	1	2	4	8	16	32
Draws per tour, D	1	1.88	3.35	5.39	7.36	8.54
Relative speedup, S	1.00	1.82	3.23	5.15	6.45	7.34

minutes.

It is seen that for the particular model, programming language, implementation and hardware the speedup is acceptably close to the upper bound, at least for $P = 1 - 8$. We conclude that on a representative HPC cluster the RWMH prefetching algorithm can be implemented successfully for a fine-grained problem. For the large-scale models mentioned above we expect $S \approx D$ in this environment.

Discussion

In an article written by researchers at the Riksbank, the central bank of Sweden, some econometric issues related to the estimation of a large-scale linearised open economy DSGE model (the RAMSES model) are addressed (Adolfson, Lindé and Villani (2007*a*)). We present some of their observations because we believe they are representative to this field of research:

1. Computing time is a major concern when Bayesian methods are employed to analyse large-scale DSGE models.

2. The one-block RWMH algorithm has been the common choice of sampler in the estimation of linearised DSGE models. A substantial problem of the one-block ICHM algorithm is that it easily gets stuck for long spells when the parameter space is high-dimensional.

3. Blocking approaches are difficult to implement for these models because full conditional posteriors are not easy to simulate from. Furthermore, this approach requires multiple evaluations of the likelihood per posterior draw.

4. It is found for the RWMH algorithm that decreasing the targeted acceptance rate from 0.25 to 0.10 leaves the inefficiency factors largely unaltered.

5. Reparameterisation increases the efficiency of sampling substantially.

These observations are largely confirmed by our exercise above and taken together we believe that they increase the attractiveness of prefetching methods in the context of large-scale linearised DSGE models.

Computing time The issue of when computing time becomes a real concern is largely context-specific. Clearly in our environment, characterised by a relatively small DSGE model, Fortran code and decent computing power, the rationale for parallel methods is perhaps limited, as indicated by the absolute execution times reported. Even for a model of this size it is however clear that the scope of experiments that can be performed in a given time span is increased.

In a more typical desktop computing environment, and using state-of-the-art DSGE modelling tools like DYNARE or YADA, single runs may take several days for the most recent generation of policy-relevant DSGE models. Total project computing time, including a substantial amount of experimentation, is measured in months.

RWMH vs. ICMH The parallel efficiency of ICMH is far superior to that of RWMH. Using ICMH with $P = 64$ processors to generate one million draws from the posterior distribution of the SW model the total execution time is around 2.5 minutes on the Lenngren cluster and the relative speedup is $S(64) = 63$. Although the ICMH sampler has much to recommend to it, especially in a parallel setting, in our example problem it does not seem straightforward to find an efficient proposal density. In a serial programming context the choice of estimation strategy would certainly be to use the RWMH algorithm. In a parallel framework the trade-offs are slightly more complicated and several factors must be taken into account.

The following example clarifies the trade-offs for our example under the assumption that ICMH requires a thinning factor which is 8 times that of RWMH to achieve roughly the same sampling efficiency. If $P = 128$ processors are used to estimate the model above with the two algorithms, parallel ICMH and RWMH prefetching, then a posterior sample of a given quality, as judged by the inefficiency factor, can be obtained roughly twice as fast for the ICMH algorithm in comparison with prefetching. If $P = 8$ processors are used the execution time of prefetching is more than 5 times faster than for parallel ICMH.

Acceptance rate The fourth observation above is especially interesting in relation to prefetching. As demonstrated above the empirical optimal static prefetching efficiency is not far behind the efficiency of the most likely path approach for the relevant range of processors in our problem. These two approaches are also the simplest to implement, since they imply nonbranched tours.

To conclude a simple heuristic strategy is suggested: choose the scaling of the incremental density to obtain the optimal static prefetching acceptance rate conditional on the number of processors (see figure 4.5) and use the ‘always prepare for a reject ’ tour. It can be expected that often this will be reasonably close to the optimal overall prefetching strategy.

4.4.2 Nonlinear estimation of a DSGE model

In this section the Metropolis-Hastings prefetching algorithm is applied to the problem of estimating a nonlinearly approximated small-scale DSGE model using Bayesian methods. In the previous section we established that in a high performance computing (HPC) environment, i.e. on the Lenngren cluster, the prefetching algorithm works successfully for a fine-grained estimation problem. Our main objective in this section is to assess prefetching performance in a particular personal high performance computing (PHPC) environment: using the parallelism of Matlab (the *Distributed Computing Toolbox, DCT*) on a quad-core desktop computer. Alternatively, and perhaps more correctly, we may interpret this exercise as an evaluation of the parallel functionality of Matlab, using the prefetching algorithm as the test ground. The multi-core/Matlab environment is presumably one of the most accessible and easy-to-use PHPC environments.

The nonlinear estimation example is chosen first because it is an estimation problem that should be sufficiently coarse-grained to deliver reasonable parallel efficiency also in the PHPC environment. Note that for a given model the particle filter based evaluation of the likelihood for the quadratically estimated model is roughly 1000 times slower than the Kalman filter likelihood evaluation for the corresponding linearly approximated model. Second, the discontinuous likelihood approximation in the nonlinear case implies that prefetching methods which rely on a posterior approximation (variants 4 and 5 above) cannot be implemented without modification. In other words, the most successful variants of prefetching in the linear estimation example of the previous section are not readily available to us here.

Model, prior, solution and likelihood

The prototypical small-scale New Keynesian model is borrowed from An (2005). Again, the economic content of the model is largely irrelevant for our purposes here and therefore no discussion of the model is provided.

The policy function of the model is approximated to the second order using the approach of Schmitt-Grohe and Uribe (2004). The approximative solution can be cast in the following state-space form. The state equation is separated into an equation for the exogenous state variables (the shocks)

$$X_{1t} = AX_{1t-1} + \varepsilon_t, \tag{4.13}$$

and an equation for the endogenous predetermined variables and a subset of the nonpredetermined variables

$$X_{2t} = B\tilde{X}_{t-1} + Cvech(\tilde{X}_{t-1}\tilde{X}_{t-1}^T) + e, \tag{4.14}$$

where $\tilde{X}_{t-1} = \left[\begin{array}{cc} X_{1t}^T & X_{2t-1}^T \end{array} \right]^T$ and $X_t = \left[\begin{array}{cc} X_{1t}^T & X_{2t}^T \end{array} \right]^T$.

In this way a linear observation equation,

$$Y_t = d + ZX_t + v_t, \tag{4.15}$$

is obtained. The innovations and measurement errors are assumed to be independent and normally distributed, $\varepsilon_t \sim N(0, Q)$ and $v_t \sim N(0, H)$. The second order policy function approximation of a large class of DSGE models can be cast in this form. The state-space representation of the corresponding linearly approximated model, which was used in the previous example, is obtained by letting $C = 0$ and $e = 0$. The matrices $A(\theta_2), B(\theta_1)$, and $C(\theta_1)$ and the vectors $e(\theta)$ and $d(\theta_1)$ are functions of the parameters of the economic model, which are collected in $\theta = \begin{pmatrix} \theta_1^T & \theta_2^T \end{pmatrix}^T$ where θ_1 consists of the structural parameters of the model and θ_2 contains the auxiliary parameters, i.e. the parameters describing the shock processes. The dimensions of the vectors are $n_{x_1} = 3$, $n_{x_2} = 4$, $n_x = 7$, $n_y = 3$, $n_\theta = 13$ and $n_\epsilon = 3$.

The model is estimated on simulated data and we use the same data-generating process, θ^{dgp}, as An (2005) and a prior distribution for θ similar to the one in An's paper. The likelihood function of the model is evaluated using an Adaptive Linear Particle Filter (ALPF), designed for the particular state-space model described by equations [4.13]-[4.15] [Strid (2007*a*)]. The particle filter and its application to the nonlinear estimation of DSGE models is not discussed in detail here and the reader is referred to the referenced articles (Arulampalam et al. (2002); Doucet et al. (2000); Fernández-Villaverde and Rubio-Ramírez (2007); Amisano and Tristani (2007)).

Implementation

The estimation routine is implemented in Matlab, using a Fortran mex function only for the systematic resampling (SR) step of the Adaptive Linear Particle Filter. The SR algorithm cannot be implemented as vectorised code, implying that a Fortran implementation of this part of the particle filter is considerably faster than its Matlab counterpart. The likelihood evaluation accuracy of the ALPF filter applied here is *at least* as good as for a standard particle filter with $40,000$ particles and the time of a posterior evaluation is roughly $2.5s$ for this Matlab implementation.

The parallel section of the prefetching algorithm, step 4 of algorithm 13, is implemented using the parallel for-loop (*parfor*) construct contained in the Parallel Computing Toolbox. The number of calls to *parfor* is thus equal to the total number of tours, N.

Experiment

The Metropolis-Hastings prefetching algorithm is executed on an Opteron 275, 2.2 Ghz, using three of the four available cores. Three variants of prefetching are tested: (i) static prefetching, (ii) dynamic prefetching based on the uniform random numbers and (iii) a modified most likely path (MLP) algorithm. The modification of the MLP algorithm in (iii) consists of updating the mode of the normal approximation to the posterior in order to obtain a successively better approximation, even though the exact mode cannot be obtained. This type of adaptation only affects the quality of prefetching and not the statistical properties of the chain. The starting mode of the approximation is the linear posterior mode.

An acceptance rate $\tilde{\alpha} = 0.25$ is targeted and the actual rate is 0.24. The potential

Table 4.3 Performance of prefetching algorithms for the nonlinear DSGE model in a multi-core/Matlab environment.

	Static prefetching		Dynamic prefetching		Modified MLP	
Processors	1	3	1	3	1	3
Draws per tour	1	2.32	1	2.37	1	2.32
Speedup	1.0	2.09	1.0	2.11	1.0	2.06

suboptimality of this choice for $P = 3$ is disregarded. In each case $50,000$ draws from the posterior are obtained. Draws per tour and speedup are presented in Table 4.3. First, the problem is sufficiently coarse-grained for implementation in the particular PHPC environment, in the sense that absolute speedup is acceptably close to draws per tour. Second, the results suggest that it is hard to improve on static prefetching when a good posterior approximation is not immediately available. All three variants roughly halves estimation time when three cores are used, from 35 to 17 hours of computing time.

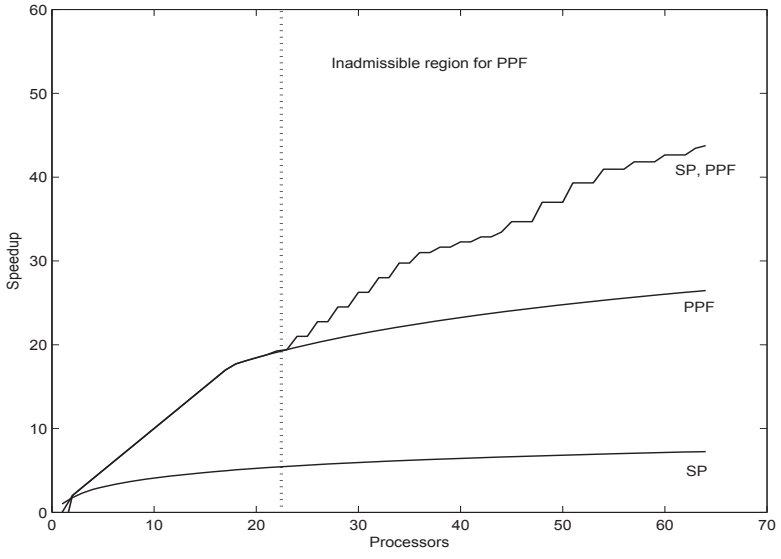
4.5 Two-layer parallelism

In this section we briefly discuss the possibility of either (i) combining prefetching with lower level parallelism or (ii) using prefetching performance to evaluate competing parallel algorithms. In the context of the Metropolis-Hastings algorithm lower level, and hence competing, parallelisation means any type of within-draw parallelisation of the posterior evaluation. Since prefetching is a general, i.e. largely problem independent, single-chain parallel algorithm it can be used to suggest admissible regions for the number of processors used by the lower level parallel algorithm. Competing parallel algorithms are necessarily problem-specific, presumably more complicated both to develop and implement and they certainly require more frequent communication between processors. This suggests a natural benchmark role for the prefetching algorithm.

The potential performance gain of combining prefetching with a lower level parallel algorithm is illustrated using the nonlinear estimation example considered above. A Parallel Standard Particle Filter (PPF) is used for the likelihood evaluation. The speedup numbers for the PPF refer to Fortran/MPI implementations run on the Lenngren cluster (Strid (2007*b*)). The number of particles employed is $N = 40,000$. For the static prefetching (SP) algorithm an acceptance rate of $\alpha = 0.25$ is targeted.

In figure 4.7 it is seen that the PPF is much more efficient than the prefetching approach. (The kink in speedup for the PPF is explained by the fact that we estimate a speedup function based on a set of observations $S(P)$, $P = 1, 2, 4, \ldots, 64$, while ruling out the possibility of superlinear speedup.) However, when the number of processors exceed $P^*_{opt} = 22$ it becomes optimal to switch to the two-layer parallel algorithm which combines static prefetching with the PPF. As the number of processors increase the speedup difference between the two-layer algorithm and the PPF becomes quite pronounced. It is also possible that a minor improvement

Figure 4.7 Two-layer parallel performance. Static prefetching and a parallel particle filter applied for the estimation of a nonlinear DSGE model.



in speedup, adjusted for statistical efficiency, can be achieved by targeting a lower acceptance rate for prefetching.

The simple calculations underlying figure 4.7 show how consideration of the prefetching alternative can guide decisions on the tolerable number of processors to use for the lower level parallel algorithm for a given estimation problem and problem size (here the number of particles employed in the particle filter). Importantly this type of scalability analysis can be performed without ever implementing the prefetching algorithm.

4.6 Conclusions

Prefetching algorithms have obvious limitations in terms of parallel efficiency. This is due to the inherently sequential nature of the Metropolis-Hastings algorithm. The advantages of the independence chain Metropolis-Hastings algorithm in a parallel context are obvious.

In this paper we have shown how to substantially increase the efficiency of prefetching using some simple techniques. Even using these techniques on reasonably well-behaved, unimodal, problems it is hard to imagine anyone applying more than, say, $10 - 15$ processors to a prefetching algorithm. Despite this the simplicity of implementing the method and the possibility of a multiplication effect if combined

with lower level parallelisation were claimed to motivate interest in the method.

The paper has highlighted some complicated trade-offs. First, for the prefetching algorithm a parallel efficiency perspective suggests targeting of a low acceptance rate. This must be weighted against Markov chain efficiency considerations. For the random walk Metropolis-Hastings algorithm it was demonstrated that the optimal acceptance rate decreases with the number of processors applied to the algorithm. Second, 'good' scalability of prefetching when a posterior approximation is used to construct tours may indicate that other sampling approaches should also be considered.

Bayesian estimation of DSGE models was identified as one potential arena for prefetching methods. In this context the experiences reported by researchers at the Riksbank, which largely correspond to the results reported here for a smaller model, suggest that prefetching methods is a viable alternative in reducing estimation time. The generality of the proposed method, however, suggests that it could be applied in other contexts as well. Brockwell (2006) applies prefetching to long memory time series models. The application of prefetching with the hyperblock sampling approach for GMRF models and in related contexts should be more thoroughly explored in future research.

Finally, it would be straightforward to implement a prefetching version of the adaptive RWMH algorithm proposed by Haario, Saksman and Tamminen (2001). In many cases we expect the proposal distribution of the RWMH and the posterior approximation used to make prefetching predictions to coincide, possibly with a different scaling as in our linear estimation example. The adaptive RWMH can then potentially increase statistical and parallel efficiency simultaneously.

Bibliography

Adolfson, M., Laséen, S., Lindé, J. and Villani, M. (2007), 'Bayesian Estimation of an Open Economy DSGE Model with Incomplete Pass-Through', *Journal of International Economics* **72(2)**, 481–511.

Adolfson, M., Lindé, J. and Villani, M. (2007), 'Bayesian Inference in DSGE Models - Some Comments', *Econometric Reviews* **26(2-4)**, 173–185.

Amisano, G. and Tristani, O. (2007), 'Euro Area Inflation Persistence in an Estimated nonlinear DSGE model', *Journal of Economic Dynamics and Control* . Forthcoming.

An, S. (2005), 'Bayesian Estimation of DSGE Models: Lessons from Second-order Approximations'. Working Paper, University of Pennsylvania.

Arulampalam, S., Maskell, S., Gordon, N. and Clapp, T. (2002), 'A Tutorial on Particle Filters for On-Line Non-Linear/Non-Gaussian Bayesian Tracking', *IEEE Transactions on Signal Processing* **50(2)**, 174–188.

Azzini, I., Girardi, R. and Ratto, M. (2007), 'Parallelization of Matlab codes under Windows platform for Bayesian estimation: a Dynare application'. Working Paper 1, Euro-area Economy Modelling Centre.

Belaygorod, A. and Dueker, M. (2006), 'The Price Puzzle and Indeterminacy in an Estimated DSGE model'. Working Paper 2006-025, Federal Reserve Bank of St. Louis.

Brockwell, A. (2006), 'Parallel Markov Chain Monte Carlo Simulation by Pre-fetching', *Journal of Computational and Graphical Statistics* **15(1)**, 246–261.

Brockwell, A. E. and Kadane, J. B. (2005), 'Identification of Regeneration Times in MCMC Simulation, with Application to Adaptive Schemes', *Journal of Computational and Graphical Statistics* **14(2)**, 436–458.

Chib, S. and Carlin, B. P. (1999), 'On MCMC Sampling in Hierarchical Longitudinal Models', *Statistics and Computing* **9(1)**, 17–26.

Christiano, L. J., Trabandt, M. and Walentin, K. (2007), 'Introducing Financial Frictions and Unemployment into a Small Open Economy Model'. Working paper 214, Sveriges Riksbank.

del Negro, M., Schorfheide, F., Smets, F. and Wouters, R. (2005), 'On the Fit and Forecasting Performance of New-Keynesian Models'. Working Paper Series, No. 491, European Central Bank.

Doucet, A., Godsill, S. and Andrieu, C. (2000), 'On Sequential Monte Carlo Sampling Methods for Bayesian Filtering', *Statistics and Computing* **10**, 197–208.

Durbin, J. and Koopman, S. J. (2001), *Time Series Analysis by State Space Methods*, Oxford University Press.

Fernández-Villaverde, J. and Rubio-Ramírez, J. F. (2007), 'Estimating Macroeconomic Models: A Likelihood Approach', *Review of Economic Studies* **74(4)**, 1059–1087.

Gamerman, D., Moreira, A. R. and Rue, H. (2003), 'Space-varying regression models: Specifications and Simulation', *Computational Statistics and Data Analysis* **42(3)**, 513–533.

Garside, L. and Wilkinson, D. (2004), 'Dynamic Lattice-Markov Spatio-Temporal Models for Environmental Data', *Bayesian Statistics* **7**, 535–542.

Haario, H., Saksman, E. and Tamminen, J. (2001), 'An adaptive Metropolis algorithm', *Bernoulli* **7(2)**, 223–242.

Hastings, W. (1970), 'Monte Carlo Sampling Methods Using Markov Chains and Their Applications', *Biometrika* **57(1)**, 97–109.

Klein, P. (2000), 'Using the Generalized Schur Form to Solve a Multivariate Linear Rational Expectations Model', *Journal of Economic Dynamics and Control* **24(10)**, 1405–1423.

Knorr-Held, L. and Rue, H. (2002), 'On Block Updating in Markov Random Field Models for Disease Mapping', *Scandinavian Journal of Statistics* **29(4)**, 597–614.

Knorr-Held, L. and Rue, H. (2005), *Gaussian Markov Random Fields: Theory and Applications*, Chapman and Hall.

Lombardi, M. J. (2007), 'Bayesian Inference for Alpha-stable Distributions: A Random Walk MCMC Approach', *Computational Statistics and Data Analysis* **51(5)**, 2688–2700.

Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A. and Teller, E. (1953), 'Equation of State Calculations by Fast Computing machines', *Journal of Chemical Physics* **21(6)**, 1087–1092.

Moreira, A. R. and Gamerman, D. (2002), 'Bayesian Analysis of Econometric Time Series Models using Hybrid Integration Rules', *Communications in Statistics. Theory and Methods* **31(1)**, 49–72.

Omori, Y. and Watanabe, T. (2008), 'Block Sampler and Posterior Mode estimation for Asymmetric Stochastic Volatility Models', *Computational Statistics and Data Analysis* **52(6)**, 2892–2910.

Roberts, G., Gelman, A. and Gilks, W. (1997), 'Weak Convergence and Optimal Scaling of Random Walk Metropolis Algorithms', *The Annals of Applied Probability* **7(1)**, 110–120.

Roberts, G. O. and Rosenthal, J. S. (2001), 'Optimal Scaling for Various Random Walk Metropolis Algorithms', *Statistical Science* **16(4)**, 351–367.

Rosenthal, J. S. (2000), 'Parallel Computing and Monte Carlo algorithms', *Far Eastern Journal of Theoretical Statistics* **4**, 207–236.

Rue, H. (2001), 'Fast Sampling of Gaussian Markov Random Field Models', *Journal of the Royal Statistical Society Ser B.* **63(2)**, 325–338.

Rue, H., Steinsland, I. and Erland, S. (2004), 'Approximating Hidden Gaussian Markov Random Models', *Journal of the Royal Statistical Society* **66(4)**, 877–892.

Schmitt-Grohe, S. and Uribe, M. (2004), 'Solving Dynamic General Equilibrium Models Using a Second-Order Approximation to the Policy Function', *Journal of Economic Dynamics and Control* **28(4)**, 755–775.

Smets, F. and Wouters, R. (2003), 'An Estimated Stochastic Dynamic General Equilibrium Model of the Euro Area', *Journal of the European Economic Association* **1(5)**, 1123–1175.

Steinsland, I. (2007), 'Parallel Exact Sampling and Evaluation of Gaussian Markov Random Fields', *Computational Statistics and Data Analysis* **51(6)**, 2969–2981.

Strickland, C. M., Forbes, C. S. and Martin, G. (2006), 'Bayesian Analysis of the Stochastic Conditional Duration Model', *Computational Statistics and Data Analysis* **50(9)**, 2247–2267.

Strickland, C. M., Forbes, C. S. and Martin, G. M. (2009), 'Efficient Bayesian Estimation of Multivariate State Space Models', *Computational Statistics and Data Analysis* **53(12)**, 4116–4125.

Strid, I. (2007*a*), 'A Simple Adaptive Particle Filter for Second-Order Approximated DSGE Models'. Mimeo, Stockholm School of Economics.

Strid, I. (2007*b*), 'Parallel particle filters for likelihood evaluation in DSGE models: An assessment'. Mimeo, Stockholm School of Economics.

Strid, I. and Walentin, K. (2008), 'Block Kalman Filters for Large-scale DSGE models', *Computational Economics* **33(3)**, 277–304.

Takahashi, M., Omori, Y. and Watanabe, T. (2009), 'Estimating Stochastic Volatility Models using Daily Returns and Realized Volatility Simultaneously', *Computational Statistics and Data Analysis* **53(6)**, 2404–2426.

Whiley, M. and Wilson, S. P. (2004), 'Parallel Algorithms for Markov Chain Monte Carlo in Latent Spatial Gaussian Models', *Statistics and Computing* **14(3)**, 171–179.

Wilkinson, D. (2006), Parallel Bayesian Computation, *in* E. J. Kontoghiorghes, ed., 'Handbook of Parallel Computing and Statistics', Chapman and Hall, chapter 16, pp. 477–508.

Wilkinson, D. and Yeung, S. K. (2004), 'A Sparse Matrix Approach to Bayesian Computation in Large Linear Models', *Computational Statistics and Data Analysis* **44(3)**, 493–516.

Yan, J., Cowles, M. K., Wang, S. and Armstrong, M. P. (2007), 'Parallelizing MCMC for Bayesian Spatiotemporal Geostatistical Models', *Statistics and Computing* **17(4)**, 323–335.

Appendix A

The prefetching algorithm

Let $L = L(i_p)$ be the function that maps Metropolis tree node indices to levels, e.g. $L(10) = 4$. The expected number of draws $D(P)$ for the tour $T(P) = \{i_1, \ldots, i_P\}$ is given by

$$D(P) = \sum_{p=1}^{P} p \Pr(\tilde{D} = p) = \sum_{p=1}^{P} p \left[\Pr(\tilde{D} \geq p) - \Pr(\tilde{D} \geq p+1) \right] =$$

$$= \sum_{p=1}^{P} \Pr(\tilde{D} \geq p) = \sum_{p=1}^{P} \left(\sum_{i_p : L(i_p) = p} \Pr(i_p) \right) = \sum_{p=1}^{P} \Pr(i_p), \tag{4.16}$$

where $\Pr\left(\tilde{D} = p\right)$ is the probability of obtaining p draws and $\Pr(i_p)$ is the probability of reaching node i_p of the Metropolis tree, where $\Pr(i_p) = 1$.

A general prefetching algorithm which constructs the optimal tour, i.e. the tour that maximises [4.16], conditional on the probabilities assigned to the branches of the Metropolis tree, is presented below. The algorithm satisfies the obvious need to avoid calculation of all branch probabilities up to level P in order to obtain a tour of size P. Instead the number of probabilities that must be calculated grows linearly in P. Observe that if i_p belongs to the tour then its parent must belong to the tour. This 'connectedness property' is used to restrict the number of probabilities calculated.

The expected number of draws per tour may be written recursively as

$$D(P) = D(P-1) + \Pr(i_P),$$

and the nodes of the optimal tour satisfy

$$1 = \Pr(1) = \Pr(i_1) \geq \ldots \geq \Pr(i_P) > 0,$$

such that the marginal value of an additional processor is decreasing.

Algorithm 16 *Prefetching (tour construction)*

1. Set $p = 2$, $D(1) = 1$, $T(1) = \{i_1; \theta_{i_1}\}$ where $i_1 = 1$ and define the sets $\Pi(1) = I(1) = \varnothing$.

2. When the p^{th} step begins the tour is $T(p-1) = \left\{i_1, \ldots, i_{p-1}; \theta_{i_1}, \ldots, \theta_{i_{p-1}}\right\}$ and i_{p-1} was added to the tour in the previous iteration. Construct a set of candidate nodes/indices

$$I(p) = I(p-1) \cup \{2i_{p-1}, 2i_{p-1}+1\}.$$

3. Construct a set containing the probabilities of reaching the candidate nodes

$$\Pi(p) = \Pi(p-1) \cup \{\Pr(2i_{p-1}), \Pr(2i_{p-1}+1)\}.$$

The required probabilities are

$$\Pr\left(2i_{p-1}\right) = \pi_p\left(L\right)\Pr\left(i_{p-1}\right) \tag{4.17}$$

and

$$\Pr\left(2i_{p-1}+1\right) = \left(1-\pi_p\left(L\right)\right)\Pr\left(i_{p-1}\right), \tag{4.18}$$

where $\Pr\left(i_{p-1}\right)$ is available from the previous iteration and where $L = L\left(i_{p-1}\right)$. The probabilities attached to the branches of the Metropolis tree, π_p, define the different variants of prefetching and they are given below.

4. Find the index $i_{\max}$, the node with the largest probability of being reached among the candidate nodes in $I\left(p\right)$, i.e. $\Pr(i_{\max}) = \max\Pi\left(p\right)$ and let $i_p = i_{\max}$.

5. Add i_p to the tour, $T\left(p\right) = T\left(p-1\right) \cup \{i_p\}$, and calculate the maximum expected draws per tour, $D\left(p\right) = D\left(p-1\right) + \Pr(i_p)$. Then remove i_p and $\Pr(i_p)$ from $I\left(p\right)$ and $\Pi\left(p\right)$ respectively but store $\Pr(i_p)$ temporarily for calculation of [4.17] and [4.18] in the next iteration.

6. If $p < P$ go to step 2, otherwise stop. ⊡

Variants of prefetching

At iteration p of the algorithm above a node i_p is allocated to the p^{th} process. The node i_{p-1} was added in the previous iteration. Note that for branched algorithms i_p is not necessarily the child of i_{p-1}. For a nonbranched algorithm the p^{th} processor will necessarily evaluate a parameter at level p in the tree, i.e. $i_p = 2i_{p-1}$ or $i_p = 2i_{p-1}+1$ and hence $L(i_p) = p$. For branched algorithms at least one node i_p will satisfy $L\left(i_p\right) < p$.

The static prefetching algorithm (2) is defined by

$$\pi_p = \alpha,$$

and the basic prefetching algorithm (1) is obtained as the special case

$$\pi_p = \alpha = 0.5.$$

The prefetching algorithm which utilises information from the sequence of uniform random numbers (3) is obtained when

$$\pi_p = \alpha\left(u^L\right),$$

where L is the level of the parent

$$L = L\left(i_{p-1}\right).$$

The conditional acceptance rate, [4.10], is repeated here

$$\alpha\left(u^L\right) = \sum_{k=1}^{K} \alpha^k I\left(u^L \in I_k\right),$$

where $I_k = [\frac{(k-1)}{K}, \frac{k}{K}]$ and where α^k was defined in [4.9]. Note that the uniform random numbers $\{u^p\}_{p=1}^P$ are connected to levels 1 to P of the Metropolis tree. Since the random numbers are draw-specific the attachment of random numbers to levels of the Metropolis tree requires that we keep track of the number of draws obtained previously.

The algorithm which is based on a posterior approximation (4) is obtained when

$$\pi_p = \min\left\{\beta, \frac{p^*\left(\theta_{i_{p-1}}\right)}{p^*\left(\theta_{i_{p-1}} - \epsilon^L\right)}\right\},$$

where $0 < \beta \leq 1$ and ϵ^L is the random vector associated with level $L\left(i_{p-1}\right)$. Note that $\theta_{i_{p-1}} - \epsilon^L$ is the *state* associated with node i_{p-1}. The random vectors $\{\epsilon^p\}_{p=1}^P$ are specific to levels 1 to P of the Metropolis tree, in the same way as the uniform random numbers.

The most likely path algorithm (5) is defined by

$$\pi_p = I\left\{u^{p-1} < \min\left\{1, \frac{p^*\left(\theta_{i_{p-1}}\right)}{p^*\left(\theta_{i_{p-1}} - \epsilon^{p-1}\right)}\right\}\right\},$$

since the tour is nonbranched and thus we know that $L = L\left(i_{p-1}\right) = p - 1$.

For the latter two algorithms, with our notation it is more convenient to express the posterior ratios in terms of proposed parameters. In implementations it is of course easy to keep track of the associated states as well.

Optimal static prefetching for the linear DSGE model

The DSGE model is estimated ten times, targeting acceptance rates $\alpha_1, \ldots, \alpha_{10}$ evenly spread in the interval $(0.015, 0.40)$. In each estimation $500,000$ draws from the posterior are obtained and the mean $\bar{\tau}\left(\alpha\right)$ of the inefficiency factors

$$\hat{\tau}_j\left(\alpha\right) = 1 + 2\sum_{i=1}^{I} Corr\left(\theta_j^t, \theta_j^{t+i}\right),\ j = 1, \ldots, n_\theta,$$

is calculated for each α. The estimated output is $\hat{E}\left(\alpha, P\right) \propto D\left(\alpha, P\right)\tau_m/\bar{\tau}\left(\alpha\right)$ where $\bar{\tau}\left(\alpha\right)$ is approximated using simple linear interpolation on the grid of acceptance rates and $\tau_m = \min_\alpha \bar{\tau}\left(\alpha\right)$. The empirical optimal acceptance rates across processors obtained in this way are quite similar to those in figure 4.5 and therefore we are content with reporting only the values that appear in table 4.1.

Appendix B

The one-block sampler for Gaussian Markov random field models

The reader is referred to the articles referenced below and Knorr-Held and Rue (2005) for a treatment of Gaussian Markov random field (GMRF) models and their use in

various areas of statistics. Let $x = (x_1, \ldots, x_n)^T$ be a GMRF which depends on a vector of hyperparameters, θ, with prior density $p(\theta)$. Let y denote the data and assume that the posterior density of interest is of the form

$$p(x, \theta|y) \propto p(x|\theta)\, p(\theta) \prod_i p(y_i|x_i),$$

where $p(y_i|x_i)$ is the likelihood for one observation.

The current state of the chain is denoted $\eta^i = \left(x^i, \theta^i\right)$ and $\tilde{\eta} = (\tilde{x}, \tilde{\theta})$ is the state at the start node of a tour. Knorr-Held and Rue (2002) suggest the updating scheme

$$\theta_z \sim f_1\left(.|\theta^i\right), \qquad x_z \sim f_2\left(.|\theta_z, y, x^i\right), \tag{4.19}$$

where the proposal $\eta_z = (\theta_z, x_z)$ is accepted/rejected jointly using a Metropolis-Hastings step. The vector of hyperparameters θ may be sampled using a random walk step. If the likelihood function is Gaussian the full conditional posterior of the field is a GMRF and it is used to sample the proposal, i.e. $f_2\left(.|\theta_z, y, x^i\right) = p(.|\theta_z, y)$. Otherwise f_2 is a GMRF approximation to this conditional. GMRFs can be sampled using the fast sampling methods presented by Rue (2001). The proposed joint update is accepted with probability $\min\{1, A\}$ where

$$A = \frac{p(\theta_z)\, p(x_z|\theta_z)\, p(y|\theta_z, x_z)\, f_2\left(x^i|\theta^i, y, x_z\right)}{p(\theta^i)\, p(x^i|\theta^i)\, p(y|\theta^i, x^i)\, f_2(x_z|\theta_z, y, x^i)}, \tag{4.20}$$

if we assume a symmetric proposal for θ.

In the one-block prefetching algorithm presented in the paper generation of proposals is performed serially in the prefetching step. In typical applications in spatial statistics the dimension of the latent field x is large and sampling from the GMRF (approximation) f_2 accounts for a major part of computational time. Therefore it is required that prefetching is applied only to the, typically low-dimensional, set of hyperparameters, θ. The GMRF proposals for the latent field, $x_{i_1}, \ldots, x_{i_P}$, are then sampled in parallel and conditional on the hyperparameters of the tour $T = \{\theta_{i_1}, \ldots, \theta_{i_P}\}$.

If the observation model is Gaussian the ratio [4.20] simplifies to

$$A = \frac{p(\theta_z)\, p(y|\theta_z)}{p(\theta^i)\, p(y|\theta^i)}, \tag{4.21}$$

and all the strategies for prefetching presented in the paper apply to the vector of hyperparameters, θ. In Knorr-Held and Rue (2002) $p(x|\theta, y)$ is approximated using a second order Taylor approximation around the current state of the latent variable, x^i, such that $f_2\left(.|\theta_z, y, x^i\right) = \tilde{p}\left(.|\theta_z, y, x^i\right)$. In this case there is only one possibility for prefetching: to target a low acceptance rate in order to obtain the 'always prepare for a reject' static prefetching tour. All proposals, $x_{i_1}, \ldots, x_{i_P}$, are generated conditional on the latent field at the start node of the tour, $\tilde{x}$, and the process-specific proposal $\theta_{i_p} \sim f_1\left(.|\theta^i\right)$.

It is realised that this strategy also applies in the more general case where there is x dependence in f_1 since all θ_{i_p} are generated conditional on the state at the start

node, $\tilde{\eta}$. This proposal is suggested by Wilkinson and Yeung (2004) in the context of linear Gaussian directed acyclic graph (LGDAG) models.

Further, if the proposal in [4.19] is used with the LGDAG model parallel sampling could proceed in two stages. First the marginal chain $\{\theta^i\}_{i=1}^R$ is obtained using prefetching. The acceptance probability is given by [4.21]. Next the latent variables $\{x^i\}_{i=1}^R$ are sampled in embarrassingly parallel fashion from $f_2 = p\left(x|\theta^i, y\right) = N_C(h^i, K^i)$. Here it is assumed that $h^i = K^i E(x|\theta^i, y)$ and the Cholesky factor G^i of the sparse precision matrix K^i, which are used to evaluate $p\left(\theta|y\right)$ in the first stage, have been stored. The two-stage approach is possible since $p\left(y|\theta\right)$, can be evaluated in a simplified way for the LGDAG model. However, sampling from f_2 is cheap given that the Cholesky factor G is available. Hence we expect the, seemingly wasteful, approach where x is instead sampled *inside* the prefetching algorithm to be more efficient in practice. In this case there is no need to store h^i and G^i after x^i has been sampled.

The crudest approximation considered by Rue, Steinsland and Erland (2004) is a GMRF approximation in the mode x^m of $p\left(x|\theta, y\right)$ such that $f_2\left(.|\theta_z, y, x^i\right) = \tilde{p}\left(.|\theta_z, y, x^m(\theta_z)\right)$. In this case all the strategies for θ-prefetching apply. This follows since the current value of the latent field, x^i, is not explicitly needed to construct the GMRF approximation. However, a 'suitable value' of x is required as a starting value for finding the mode, x^m. A natural candidate in a prefetching context is $\tilde{x}$ or, if it is important to reduce communication requirement, the locally stored mode, $x^m_{i_p}$, from the previous tour. In the latter case the latent field need not be transferred between processors.

A potential weakness of the resulting prefetching algorithm is the inhomogeneous computing time in the optimisation step but we expect this to be of minor importance. Some efficiency loss could be acceptable and, if thought necessary, the number of iterations of the optimiser, e.g. Newton-Raphson, could be fixed at a low number to restore load balance. Note that a cruder approximation yields a lower acceptance rate which, in turn, increases prefetching efficiency. Finally it would be possible to construct a two-layer parallel algorithm using the parallel sampling approach for GMRFs suggested by Steinsland (2007).

Multiple block prefetching

Consider the multiple block Metropolis-Hastings algorithm with B blocks where the parameter vector θ is split into blocks $\theta^0, \ldots, \theta^{B-1}$. A chain of length R requires $\tilde{B} = RB$ block updates. For notational convenience, assume that the blocks are updated in a fixed order from block 0 to $B-1$ using the proposal densities

$$\theta^b \sim f_b\left(.|\theta^{i-1,b}, \theta^{i,<b}, \theta^{i-1,>b}\right), \; b = 0, \ldots, B-1,$$

where $\theta^{i,>b} = (\theta^{i,b+1}, \theta^{i,b+2}, \ldots, \theta^{i,B-1})$ and $\theta^{i-1,b}$ is the state of block b when the i^{th} sweep begins. The choice of scanning strategy does not affect prefetching since the scan order for the complete chain is realised at the outset of sampling.

The probability of a move for block b is given by

$$\alpha^b = \min\left\{1, \frac{p\left(\theta^b, \theta^{i,<b}, \theta^{i-1,>b}\right)}{p\left(\theta^{i-1,b}, \theta^{i,<b}, \theta^{i-1,>b}\right)} \frac{f_b\left(\theta^{i-1,b}|\theta^b, \theta^{i,<b}, \theta^{i-1,>b}\right)}{f_b\left(\theta^b|\theta^{i-1,b}, \theta^{i,<b}, \theta^{i-1,>b}\right)}\right\}, \tag{4.22}$$

and we must, as before, assume that evaluation of p is 'expensive'. For simplicity we consider only nonbranched tours. In this context a tour is a collection of *proposed block updates* of the form

$$T = \left\{\theta_{i_1}^{\mathrm{mod}(b+1,B)}, \theta_{i_2}^{\mathrm{mod}(b+2,B)}, \ldots, \theta_{i_P}^{\mathrm{mod}(b+P,B)}\right\},$$

where it is assumed that block b has just been updated when the tour is entered. Tours and sweeps thus overlap and blocks are associated with levels of the Metropolis tree. Let $\tilde{\theta} = \left(\theta^{i,\leq b}, \theta^{i-1,>b}\right)$ be the parameter value at the start node of the tour. For random walk updates it will be desirable to tune each f_b such that bold moves are proposed, in order to decrease block acceptance rates and increase parallel efficiency.

For example, assume $B = 3$, $P = 6$ and that block 1 has just been updated such that the current parameter is $\tilde{\theta} = (\theta^{i,0}, \theta^{i,1}, \theta^{i-1,2})$. The 'always prepare for a reject' tour is $T = \left\{\theta_1^2, \theta_3^0, \theta_7^1, \theta_{15}^2, \theta_{31}^0, \theta_{63}^1\right\}$ and each processor evaluates $p(\theta_{i_p}^b, \tilde{\theta}^{-b})$ where $\theta_{i_p}^b \in T$. The tour produces at least one block update and at most six block updates, i.e. two draws.

A Gibbs step yields perfect prefetching since

$$f_b\left(.|\theta^{i-1,b}, \theta^{i,<b}, \theta^{i-1,>b}\right) = p\left(.|\theta^{i,<b}, \theta^{i-1,>b}\right),$$

implies $\alpha^b = 1$ and hence prefetching does not apply for a pure Gibbs sampler. A Gibbs component within a Metropolis block sampler would be handled by grouping the Gibbs update with a Metropolis update on a single processor, under the assumption that the Gibbs update is 'cheap'. Clearly, whenever a Gibbs step accounts for a dominant part of computational time prefetching is not feasible. An example of this is when the simulation smoother is used to sample the latent variables in linear Gaussian state-space models.

Chapter 5

Adaptive hybrid Metropolis-Hastings samplers for DSGE models

Adaptive hybrid Metropolis-Hastings samplers for DSGE models[1]

abstract

Bayesian inference for DSGE models is typically carried out by single block random walk Metropolis, involving very high computing costs. This paper combines two features, adaptive independent Metropolis-Hastings and parallelisation, to achieve large computational gains in DSGE model estimation. The history of the draws is used to continuously improve a t-copula proposal distribution, and an adaptive random walk step is inserted at predetermined intervals to escape difficult points. In linear estimation applications to a medium scale (23 parameters) and a large scale (51 parameters) DSGE model, the computing time per independent draw is reduced by 85% and $65 - 75\%$ respectively. In a stylised nonlinear estimation example (13 parameters) the reduction is 80%. The sampler is also better suited to parallelisation than random walk Metropolis or blocking strategies, so that the effective computational gains, i.e. the reduction in wall-clock time per independent equivalent draw, can potentially be much larger.

[1]This paper is joint work with Paolo Giordani, Sveriges Riksbank, and Robert Kohn, Australian School of Business, University of New South Wales.

We would like to thank Mattias Villani for providing computer code for the RAMSES model and for helpful discussions.

5.1 Introduction

Dynamic Stochastic General Equilibrium (DSGE) models are commonly estimated using Bayesian methods. A prior distribution for the model parameters is updated to a posterior distribution using likelihood information, with sampling from the posterior carried out using Markov Chain Monte Carlo (MCMC) inference. The main features of the approach are well described in the review article by An and Schorfeide (2007*a*), and some notable contributions to the field are Smets and Wouters (2003), Adolfson, Laséen, Lindé and Villani (2007) and Fernández-Villaverde and Rubio-Ramírez (2007). A key feature in this growing literature is the almost exclusive use of the single-block random walk Metropolis (RWM) algorithm to sample from the posterior distribution of the model parameters. As noted by An and Schorfeide (2007*b*) '. . . there is little research that tries to compare the performance of alternative sampling schemes and develop potentially better MCMC schemes' for DSGE models and Adolfson, Lindé and Villani (2007*a*) states that 'there is clearly room for big improvements in simulation efficiency'. Simulation efficiency is important because MCMC inference is very time consuming for DSGE models, which considerably slows down the process of model development.

The purpose of our paper is to evaluate adaptive MCMC algorithms applied to the estimation of DSGE models. The main element of adaptive sampling schemes is the use of previous MCMC draws for the design of efficient proposal densities. The key sampler of interest in this paper is based on two main ideas. The first is to use the history of posterior draws to repeatedly estimate t-copula densities with mixture of normal marginals, and use these as proposal distributions in an independence Metropolis-Hastings sampler. The efficient performance of proposals constructed in this way has been established for a variety of models in Kohn and Giordani (2008) and Silva, Kohn, Giordani and Mun (2008). Second, in order to alleviate some potential shortcomings of a pure independence chain approach in high-dimensional problems, we propose simple hybrid, deterministic cycling, algorithms, which occasionally use random walk proposals to escape points in the posterior parameter space where the posterior-to-proposal ratio is large.

An attractive feature of all the samplers presented here is that the computing time per posterior draw does not increase significantly in comparison with the RWM sampler. In the case of the main sampler, this is explained by the fast estimation of the mixture of normal marginal densities using clustering. Furthermore, our preferred algorithms are very suitable for parallel implementation. Parallel computation is becoming increasingly accessible and has the potential to drastically reduce computing time in a variety of problems, and different MCMC schemes differ greatly in their suitability for parallel implementation.

The sampling performance of the competing algorithms is first evaluated using two linearised DSGE models: A modified version of the medium-scale DSGE model developed by Smets and Wouters (2003), one of the core models at the European Central Bank, and the large-scale open economy model, nicknamed RAMSES, in use at the central bank of Sweden (Adolfson, Laséen, Lindé and Villani (2007)). Based on a comparison of inefficiency factors the efficiency of the main sampler presented here

is respectively $6-7$ and $3-4$ times larger than for the standard single-block RWM algorithm in these estimation examples. Next, the samplers are evaluated using a nonlinearly approximated small-scale New Keynesian model, where a particle filter is employed for the likelihood evaluation. In this case, the efficiency of the main sampler is roughly 5 times larger than for the RWM algorithm. Taking parallelisability into account the total efficiency gain, as measured by independent equivalent draws per time unit, is potentially much larger in all these examples.

Recently Chib and Ramamurthy (2010) have suggested a tailored randomised-block (TaRB-MH) algorithm for the estimation of DSGE models. They show that the TaRB-MH algorithm is able to obtain the correct posterior distribution in two well-known DSGE model estimation problems where the single-block RWM algorithm fails to explore all local modes. The reported computational times, however, are a factor $150-1100$ times larger per posterior draw in comparison with the RWM algorithm, and the TaRB-MH algorithm is not suitable for parallelisation.

The sampling algorithms presented here are entirely different in character: they have roughly the same computing time per posterior draw as the single-block RWM and are easily parallelised. The emphasis is on reduced computing time per independent equivalent draw rather than on an increased ability to explore multiple modes. We choose to consider only single-block samplers in this paper because multiple block MCMC algorithms cannot be parallelised efficiently. Obviously, as the dimension of the parameter vector increases one-block independence samplers will accept less and less often, and at some point it will be necessary to apply blocking methods. However, the successful application to RAMSES, with its 51 parameters and 60 state variables, suggests that our single block algorithms should prove useful in a large number of problems.

The paper proceeds as follows. In section 5.2, the econometrics of DSGE models is briefly described. In section 5.3, the sampling algorithms are presented and in section 5.4, parallelisation of the algorithms is described on a conceptual level. In section 5.5, the statistical performance of the samplers is evaluated in three DSGE model estimation examples.

5.2 DSGE models

The equilibrium of a DSGE model is described by a set of nonlinear expectational equations. These consist of constraints and first-order conditions derived from the optimising behaviour of various agents, such as consumers and firms, in a model economy. The solution to the equations is given by a policy function

$$X_t = g\left(X_{t-1}, \epsilon_t; \theta\right), \tag{5.1}$$

which relates a vector of state variables, X_t, to its lagged value, a vectors of innovations, ϵ_t, and the parameter vector, θ. The policy function cannot typically be derived in closed form. This implies that numerical approximation methods must be used to obtain an approximation, $\hat{g}$, to g.

The likelihood evaluation for a DSGE model consists of two parts. First, given a parameter vector, θ, an approximation to the policy function is obtained (the solution) and represented as a state-space model. DSGE models with non-stationary variables, e.g. induced by a non-stationary technology shock, are transformed to stationarity prior to solution of the model, such that X_t is a vector of stationary variables. Second, filtering methods are used to evaluate the likelihood function.

We first consider (log-)linear approximations, the associated linear and Gaussian state-space (LGSS) model and Kalman filtering. Several methods are available to solve DSGE models linearly, e.g. Anderson and Moore (1985), Sims (2000) and Klein (2000). In general the fastest, and also most accurate, method available is the Anderson-Moore algorithm (AIM), see the comparison in Anderson (2008).

In the (log-)linear approximation case, the resulting state-space model is

$$X_t = \hat{g}\left(X_{t-1}, \epsilon_t; \theta\right) = T\left(\theta\right) X_{t-1} + R\left(\theta\right) \epsilon_t, \tag{5.2}$$

and

$$Y_t = d\left(\theta\right) + Z X_t + v_t,\ t = 1, \ldots, T\ , \tag{5.3}$$

where [5.2] is the state equation, and [5.3] is the observation equation. Here X_t (dimension n_x) is a vector containing the state variables, in the econometric sense, and Y_t (n_y) is a vector containing the observed variables. The parameters of the model are collected in the vector θ (n_θ) and the coefficient matrices, T (which is typically dense) and R, and the vector d are nonlinear functions of θ. For the DSGE models considered in this paper Z is a selector matrix which does not depend on θ. The innovations, ϵ_t (n_ϵ), and the measurement errors, v_t (n_v), are assumed to be independent and normally distributed, $\epsilon_t \backsim N\left(0, \Sigma_\epsilon\right)$ and $v_t \backsim N\left(0, \Sigma_v\right)$. The resulting model is therefore a particular type of LGSS model. Importantly, in DSGE models the functions $T\left(\theta\right), R\left(\theta\right), d\left(\theta\right)$ are not available analytically (except in very special cases), but are obtained for a given θ by numerically solving for the rational expectation equilibrium, so Gibbs sampling is not feasible in general. The distribution of the initial state vector, X_0, is typically assumed to be equal to the stationary distribution of the state vector, $X_0 \backsim N\left(0, \Sigma_x\right)$, where Σ_x is the solution to the Lyapunov equation

$$\Sigma_x = T \Sigma_x T^T + R \Sigma_\epsilon R^T. \tag{5.4}$$

In the nonlinear estimation example of this article a second order approximation to the policy function, [5.1], is used, following Schmitt-Grohe and Uribe (2004) and Klein (2000). The approximate solution can be cast in the following state-space form. The state equation vector is separated into an equation vector for the exogenous state variables (the shocks)

$$X_{1t} = A X_{1t-1} + \varepsilon_t, \tag{5.5}$$

and an equation vector for the endogenous predetermined variables and a subset of the nonpredetermined variables of the model

$$X_{2t} = B \tilde{X}_{t-1} + C vech(\tilde{X}_{t-1} \tilde{X}_{t-1}^T) + e, \tag{5.6}$$

where $\tilde{X}_{t-1} = (X_{1t}^T \; X_{2t-1}^T)^T$ and $X_t = (X_{1t}^T \; X_{2t}^T)^T$. If we let $C = 0$ and $e = 0$, [5.5] and [5.6] reduce to [5.2], and partitioning $B = [B_1 \; B_2]$ the state transition matrix in [5.2] is

$$T = \begin{pmatrix} A & 0 \\ B_1 A & B_2 \end{pmatrix}. \tag{5.7}$$

Having expressed the nonlinear state equation in this way, the measurement equation is linear and given by [5.3]. In the nonlinear case the normality assumption for ϵ_t and v_t is no longer crucial but we retain it here for convenience.

The likelihood function is

$$p(Y_{1:T}|\theta) = \prod_{t=1}^{T} p(Y_t|Y_{1:t-1};\theta) = \prod_{t=1}^{T} \int p(Y_t|X_t)p(X_t|Y_{1:t-1})dX_t,$$

where $Y_{1:T} = (Y_1, \ldots, Y_T)$ is the data. In the (log-)linear case the likelihood is evaluated using the prediction error decomposition and the Kalman filter, e.g. Harvey (1989). In estimating large DSGE models, and assuming that the fastest solution method (AIM) is used, the dominant share of computational time is spent on Kalman filtering. DSGE-specific Kalman filter implementations, mainly based on exploitation of the special structure of the state transition matrix T in [5.7], can reduce computing time significantly in the case of large models (Strid and Walentin (2008)).

In the nonlinear case, a standard particle filter (SPF) is applied for the likelihood evaluation, see Arulampalam et al. (2002) for an introduction to particle filtering and Fernández-Villaverde and Rubio-Ramírez (2007), Amisano and Tristani (2007) and An (2005) for applications to DSGE models. The SPF requires two conditions to be fulfilled: the ability to simulate from the state equation, [5.5] and [5.6], and the ability to evaluate the observation density, $p(Y_t|X_t)$. The filter yields an unbiased estimate, $\hat{p}(Y_{1:T}|\theta)$, to the likelihood, $p(Y_{1:T}|\theta)$. Flury and Shephard (2008) and Silva, Kohn, Giordani and Pitt (2009) show that it is feasible to use this estimate inside a non-adaptive or adaptive Metropolis-Hastings algorithm. The computational burden associated with the particle filter implies that the demand for efficient sampling schemes is even larger in this case. For the small-scale New Keynesian model used for the illustration in section 5.5.3, the particle filter likelihood evaluation, with $N = 40,000$ particles, is more than $1,000$ times slower than the Kalman filter evaluation for the corresponding linearised model.

The kernel of the posterior density of the parameter vector θ is

$$p(\theta|Y_{1:T}) \propto p(Y_{1:T}|\theta)\tilde{\pi}(\theta)I(\theta \in \Theta_D),$$

where attention is usually restricted to the determinacy region, Θ_D (see Lubik and Schorfheide (2004) for an exception). In the nonlinear case, the exact likelihood is replaced by the particle filter approximation. The determinacy region is the subset of the parameter space where the model has a unique and stable solution. (DSGE models typically admit an infinite number of explosive solutions.) Here, we interpret the restriction as being part of the formulation of the prior distribution, i.e. the prior

density π is truncated at the boundary of the indeterminacy region. The effective prior distribution is typically formulated as a set of marginal prior distributions[2]*:*

$$\pi(\theta) = I(\theta \in \Theta_D)\tilde{\pi}(\theta) = I(\theta \in \Theta_D)\Pi_{j=1}^{n_\theta}\pi_j(\theta^j),$$

where θ^j is the j^{th} element of the vector θ.

There are essentially three categories of structural parameters: unbounded parameters, parameters which are bounded from below or above, and parameters which are bounded both from below and above. In the DSGE literature, parameters belonging to these classes are typically given normal, gamma (or inverse gamma) and beta prior distributions, respectively.[3] Parameters with a (inverse) gamma prior density on $[0, \infty)$ are commonly reparameterised using the log transformation, and for parameters with beta prior densities on $[0, 1)$ the logit transformation is used, see e.g. Adolfson, Laséen, Lindé and Villani (2007).

Reparameterisation serves two purposes. First, the transformation to an unbounded parameter space simplifies optimisation of the posterior, which is often used as a preliminary step for MCMC analysis. Second, the posterior of the transformed set of parameters is typically closer to a normal distribution in shape, and therefore simpler to sample from. Reparameterisation is important in order to avoid exaggerating the gains of more elaborate MCMC approaches, such as those presented below (in section 5.3), and furthermore it is easy to implement (Adolfson, Lindé and Villani (2007*a*)).

Having obtained the posterior density of the parameters $p(\theta|Y_{1:T})$, the posterior density of the state variables $p(X_{1:T}|Y_{1:T})$ can be obtained via smoothing techniques using the relation

$$p(X_{1:T}|Y_{1:T}) = \int p(X_{1:T}|\theta, Y_{1:T})p(\theta|Y_{1:T})d\theta,$$

where $X_{1:T} = (X_1, \ldots, X_T)$, see e.g. Durbin and Koopman (2001) in the linear case. It is useful to contrast the 'marginalisation' approach used in the case of DSGE models, where $X_{1:T}$ is integrated out using the Kalman filter, with the classic two-block scheme for sampling in more 'typical' LGSS models, where one alternately samples from the densities $p(\theta|X_{1:T}, Y_{1:T})$ and $p(X_{1:T}|\theta, Y_{1:T})$ using Gibbs and/or Metropolis-Hastings updates. The inability to sample from $p(\theta|X_{1:T}, Y_{1:T})$ distinguishes the DSGE-LGSS model from other LGSS models.

[2]The marginal priors are not, strictly speaking, independent due to the indeterminacy constraint.

[3]In all models considered in this paper the parameter describing the steady state technology growth rate has a truncated normal prior distribution. However, the mass attached to the truncation region is small enough to be disregarded, e.g. in the case of the RAMSES model the distance between the prior mean and the truncation point is 12 standard deviations.

5.3 Sampling algorithms

5.3.1 Random walk and independence Metropolis-Hastings

The objective of sampling algorithms is to generate a sequence of draws, θ_i, $i = 1, \ldots, R$, from the posterior kernel, $p(\theta) = p(\theta|Y_{1:T})$, where R is the length of the chain. In the context of Bayesian estimation of DSGE models, the single-block random walk Metropolis (RWM) algorithm has been the preferred sampling method. In the RWM algorithm, a proposal, θ_p, is generated using a symmetric proposal density, $q(.|\theta_i)$, where θ_i is the current state. The proposal is then accepted with probability

$$\alpha_{i+1} = \min\left\{1, \frac{p(\theta_p)}{p(\theta_i)}\right\}.$$

The proposal distribution is typically chosen to be a normal distribution, $q(\theta_p|\theta_i) = N(\theta_i, \Sigma)$, with Σ proportional to the inverse of the Hessian at the posterior mode, θ_m, $\Sigma = -\varkappa H_m^{-1}$, where $\varkappa > 0$. A crucial step in this approach is then the optimisation of the posterior density, since the quality of the RWM sampler will rely on the quality of the estimated inverse Hessian at the posterior mode. As demonstrated starkly by both Andreasen (2008) and Chib and Ramamurthy (2010), finding the global posterior (or likelihood) mode is a non-trivial problem in large-dimensional DSGE models. An adaptive RWM approach (see below) is potentially more robust; because the proposal covariance matrix is updated throughout the chain, a poor starting value for this covariance matrix does not affect sampling efficiency as heavily.

There are several reasons for the popularity of the single-block RWM algorithm in the DSGE model context. First, a standard B-block Metropolis-Hastings sampler is automatically penalised by a B-factor increase in computational time, since the likelihood needs to be evaluated once per block update. Furthermore, it is non-trivial, at least a priori, to group parameters into fixed blocks such that there is weak dependence between parameters in separate blocks. Second, for large dimensional parameter vectors the independence Metropolis-Hastings (IMH) is fragile, in the sense that it can occasionally get trapped for long spells at points in the parameter space where the posterior-to-proposal ratio, $p(\theta_i)/q(\theta_i)$, is high. For simplicity we will refer to such occurrences as 'difficult points'. The acceptance rate of the IMH sampler is

$$\alpha_{i+1} = \min\left\{1, \frac{p(\theta_p)/q(\theta_p)}{p(\theta_i)/q(\theta_i)}\right\},$$

where $q(\theta)$ denotes the independent proposal density.

5.3.2 The hybrid adaptive sampler

The samplers used in this paper arise from a general deterministic hybrid formulation. Proposals are generated from three densities, $q_{j,i+1}(\theta_p|\Theta_i)$, $j = 1, 2, 3$, according to a deterministic cycling scheme, where $\Theta_i = \{\theta_0, \ldots, \theta_i\}$ denotes the draws up to iteration i, and θ_0 is the starting value of the chain. The fraction of draws from the proposal distributions are denoted α_j, $j = 1, 2, 3$, and these fractions may change at

prespecified points of the chain (while our notation does not incorporate explicitly that α_j varies). The use of a deterministic cycling scheme, rather than random cycling, facilitates parallelisation of the algorithm, which is discussed in section 5.4.

The first component, q_1, is a random walk Metropolis (RWM) proposal and the second component, q_2, is a unimodal and symmetric independence proposal (UIMH). In the experiments, the latter component is a multivariate t or normal distribution. These proposal distributions may be adaptive or non-adaptive. The *main component*, q_3, is a t-copula with mixture of normal marginal distributions, estimated using clustering methods (TC), as in Silva et al. (2008). The t copula provides a fast and flexible way of estimating a multivariate density and, in general, it yields a more accurate estimate of the posterior density than the 'standard' symmetric independence proposal, q_2. Various samplers thus arise by different choices of the fractions α_j, e.g. the RWM sampler is described by $\alpha_1 = 1$ and $\alpha_2 = \alpha_3 = 0$. The three components are described in detail below, in sections 5.3.3 and 5.3.4.

The *main sampler* of interest, denoted RWM-UIMH-TC, is a sampler containing all three components, i.e. $\alpha_j > 0$ for all j, where the fraction of draws from the main component, α_3, increases gradually through the chain. In an initial stage $\alpha_3 = 0$ and then α_3 is increased to some value $\alpha_3 = \alpha_3^* \geq \tilde{\alpha}$, where $\tilde{\alpha}$ should be 'large', such that, eventually, the majority of the draws are generated by the main component. In the experiments, in section 5.5, $\tilde{\alpha}$ is never below 0.75. For simplicity, the ratio of random walk proposals to unimodal independence chain proposals, α_1/α_2, is kept fixed throughout the chain for all samplers which include both these components.

In the main sampler, one function of the first two components, q_1 and q_2, is to start up the chain and produce a number of accepted draws, in an initial stage of sampling. It is crucial that the acceptance rate in the initial stage is not too low, since this would prevent successful estimation of the mixture of normal marginals of the main component, q_3. In our experience, the ability to recover from a 'bad start' depends, to a large extent, on the dimensionality of the parameter vector. It is straightforward to tune the random walk component, q_1, to get a sufficiently high acceptance rate, and therefore this component should always be included in the initial phase. Experimenting with the algorithm, we have, however, found that the hybrid, consisting of both q_1 and q_2, typically explores the posterior faster in the initial phase. This motivates the inclusion of the unimodal independence component, q_2.

The random walk component, q_1, also has a second function in the sampler. A pure independence sampler, i.e. a sampler which only uses some combination of the independence components, q_2 and q_3, runs the risk of getting trapped for long spells at difficult points in the parameter space, especially as the dimension of the problem increases. The primary motive to include the random walk component, q_1, also *after* the initial phase is, therefore, to be able to escape such difficult points. However, the improved approximating ability of the copula component, q_3, in comparison with the 'standard' independence component, q_2, should allow us to insert these RWM steps less frequently, in comparison with a sampler consisting of only q_1 and q_2.

The *intensity of adaptation* depends on (i) how early adaptation begins, (ii) the frequency of adaptation, i.e. how often q_3 is re-estimated, and (iii) how quickly the share of draws from the main component, α_3, increases. The *adaptation schedule*, the

set of points where q_3 is re-estimated, is denoted

$$A = \left\{ i_0^A, i_1^A, i_2^A, \ldots, i_M^A \right\},$$

where $i_0^A = 0$. In order not to complicate the sampler further, these are also the points where we allow the shares α_j to change, such that there is a *hybrid schedule*

$$H = \left\{ \alpha_3^0, \alpha_3^1, \alpha_3^2, \ldots, \alpha_3^M \right\},$$

corresponding to the adaptation schedule. Here α_3^j denote the fraction of draws from the main component between the adaptation points i_j^A and i_{j+1}^A, for $j \geq 0$. The first adaptation point, i.e. the first time q_3 is estimated, is i_A^1. At this point α_3 is switched from $\alpha_3^0 = 0$ to $\alpha_3^1 = 0.5$. This is illustrated in figure 5.1 where two adaptation schemes are pictured. The adaptation points are marked as dots on the hybrid schedule. In the relatively intensive adaptation scheme, adaptation is started early, at $i_A^1 = 5,000$, adaptation is frequent and the share of draws from the main component, α_3, increases quickly. In the less intensive scheme more trust is placed in the exploring capabilities of q_1 and q_2 and i_A^1 is therefore larger. Alternatively, the initial phase must be longer, in order to get a reasonably good initial estimate of q_3. Note the relative sense in which we use the term 'intensive'; since both schedules displayed here are much less intensive than those used for the lower dimensional models in Kohn and Giordani (2008). In the evaluation, in section 5.5, the intensive scheme is used with a medium-scale (number of parameters, $n_\theta = 23$) DSGE model and the less intensive scheme with a large-scale ($n_\theta = 51$) model.

In our experience, with a high-dimensional DSGE model the danger of an intensive adaptation scheme is that q_3 fitted on too few draws may provide a poor approximation to the posterior. If, in addition, the share of draws from the main component, α_3, is increasing steeply early in the chain, the overall acceptance rate can be low and in the worst case the sampler collapses. The adaptation schedules presented in figure 5.1 may serve as a useful starting point in estimating models of the indicated sizes.

5.3.3 ARWM and AUIMH components

The adaptive Metropolis algorithm was introduced by Haario et al. (2001). The proposal distribution for the ARWM used in this paper is

$$q_{1,i+1}(\theta|\Theta_i) = \alpha\beta N\left(\theta_i, \kappa_1\Sigma_{i+1}\right) + (1-\alpha)\beta N\left(\theta_i, \kappa_2\Sigma_{i+1}\right) + (1-\beta)N\left(\theta_i, \kappa_3 I\right), \quad (5.8)$$

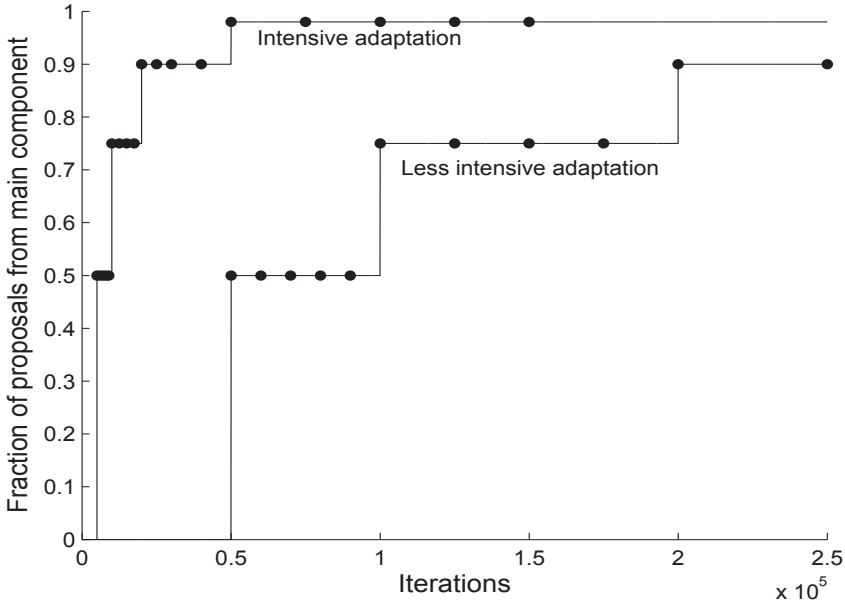
where Σ_i is updated using the recursive formulas

$$\Sigma_{i+1} = \frac{i}{i+i_0}\left(\frac{i-1}{i}\Sigma_i + \frac{(i\bar{\theta}_{i-1}\bar{\theta}_{i-1}^T - (i+1)\bar{\theta}_i\bar{\theta}_i^T + \theta_i\theta_i^T)}{i}\right) + \frac{i_0}{i+i_0}\Sigma_0, \quad (5.9)$$

and

$$\bar{\theta}_i = \frac{i+1}{i_0+i+1}\left(\frac{i}{i+1}\bar{\theta}_{i-1} + \frac{1}{i+1}\theta_i\right) + \frac{i_0}{i_0+i+1}\bar{\theta}_0, \quad (5.10)$$

Figure 5.1 Adaptation and hybrid schedules.



such that i_0 is the 'prior sample size' placed on an initial mean $\bar{\theta}_0$ and covariance matrix Σ_0. Here we will use $\bar{\theta}_0 = \theta_m$, where θ_m is the posterior mode, and $\Sigma_0 = -H_m^{-1}$, where H_m^{-1} is the inverse Hessian at the posterior mode. The scaling parameters are chosen as $\kappa_1 = 2.38^2/n_\theta$, $\kappa_2 = 9\kappa_1$ and $\kappa_3 = 0.1^2/n_\theta$, where n_θ is the dimension of θ. The purpose of the second component is to be able to escape local modes (see Silva et al. (2008)), and we have chosen the scale factor of 9 for κ_2 rather arbitrarily. The scaling parameter κ_1 is optimal when the target density is Gaussian (Roberts and Rosenthal (2001)). When $\alpha = \beta = 1$ and $i_0 = \infty$, the (non-adaptive) RWM sampler is obtained as a special case. In the adaptive case (ARWM), we let $i_0 = 100$, $\beta = 1$ and $\alpha = 0.95$, in the experiments below. We also consider an adaptive sampler, ARWM3C, which does not use any prior information, i.e. $i_0 = 0$, and which uses all three components. Here $\beta = 0$, during a short start phase $i \leq i^*$, and for $i > i^*$ we let $\beta = 0.95$ and $\alpha = 0.9$.

The unimodal independence proposal is given by the multivariate t distribution

$$q_{2,i+1}(\theta) = t\left(\bar{\theta}_i, \Sigma_{i+1}, v\right), \tag{5.11}$$

and this component is referred to as adaptive unimodal independence Metropolis-Hastings (AUIMH). The mean and covariance are updated using [5.9] and [5.10] and the degrees of freedom, v, is fixed. In our experiments below we let $v = 10$ and

$i_0 = 1000$ for the AUIMH sampler. When q_2 is a component in a hybrid sampler, we let $\nu = \infty$ such that

$$q_{2,i+1}(\theta) = N\left(\bar{\theta}_i, \Sigma_{i+1}\right).$$

The presence of fat tails in the proposal distribution, q_2, appears to be less important when it is used as a component in a hybrid sampler (see Kohn and Giordani (2008) for a discussion). In experiments we have found that a hybrid sampler consisting of q_1 and q_2 performs best when $\nu = \infty$. The values chosen for ν are based on a limited amount of experimentation, comparing $v = 5, 10$ and ∞.

5.3.4 The t copula with mixture of normal marginal component (TC)

The main component of the proposal distribution, q_3, is the t copula with mixture of normal marginal distributions introduced by Silva et al. (2008).

Let $t_{n_\theta,v}(x|\mu,\Sigma)$ denote the n_θ-dimensional t density function with mean μ, degrees of freedom v and covariance matrix $\tilde{\Sigma} = \frac{v}{v-2}\Sigma$ and let $T_{n_\theta,v}(x|\mu,\Sigma)$ denote the corresponding cumulative distribution function. Further, let $f_j(\theta^j|\lambda_j)$ and $F_j(\theta^j|\lambda_j)$, $j = 1,\ldots,n_\theta$, denote the density and distribution function, respectively, of a univariate mixture of normal, where $\lambda_j = (m_j^T \; v_j^T \; p_j^T)^T$ collects the parameters describing the mixture, namely the component means, variances and probabilities. The main component of the proposal density is then described by the mixture

$$q_{3,i+1}(\theta|\Theta_i) = \tilde{\beta} q_{31,i+1}(\theta|\Theta_i) + (1-\tilde{\beta}) q_{32,i+1}(\theta|\Theta_i),$$

where

$$q_{31} = \frac{t_{n_\theta,v}(x|\mu,\Sigma)}{\Pi_{i=1}^{n_\theta} t_{1,v}(x^j|0,1)} \Pi_{i=1}^{n_\theta} f_j(\theta^j|\lambda_j), \tag{5.12}$$

and where x^j and θ^j are related by

$$T_{1,v}(x^j|0,1) = F_j(\theta^j|\lambda_j) \text{ for } j = 1,\ldots,n_\theta. \tag{5.13}$$

The second component q_{32} is a fat-tailed version of q_{31} obtained by inflating the univariate mixture of normal variances by a factor ψ. The purpose of this component is, as above, to be able to escape local modes. In our experiments the fraction of draws from q_{32} is set to $1-\tilde{\beta} = 0.05$ and the inflation factor is $\psi = 9$. We next describe how to (i) estimate q_{31} based on a set of draws, $\theta_0,\ldots,\theta_{\tilde{R}}$, (ii) how to evaluate q_3, and (iii) how to generate a sample from q_3.

Estimation and evaluation of $q_{3,1}$

The mixture of normal marginal densities, f_j, are estimated using clustering based on a sample $\theta_0^j,\ldots,\theta_{\tilde{R}}^j$, where $\tilde{R}$ is the number of draws obtained from the posterior distribution at the point of (re-)estimation, as described in Kohn and Giordani (2008) and Silva et al. (2008). A brief description of the clustering algorithm is provided in Appendix A. The number of mixture components, n_j, is selected based on the

Bayesian information criterion (BIC), where the maximum number of components is set to $n_{\max} = 4$.

An additional step is added here in comparison with Silva et al. (2008). In an attempt to improve the fit of the normal mixture, its likelihood function is maximised subject to some restrictions: we condition on the number of components n_j selected by the BIC criterion and restrict the means, m_j^T, and variances, v_j^T, to some interval around the clustering estimates, $\tilde{m}_j^T$ and $\tilde{v}_j^T$. The probabilities p_j^T are restricted from below.[4] The clustering estimate is used as the starting value for the optimisation routine and a maximum number of mixture of normal density evaluations, e.g. 1,000, is specified. The modified estimate obtained is denoted $\hat{\lambda}_j$ and it is ensured that $L(\hat{\lambda}_j) \geq L(\tilde{\lambda}_j)$ where L is the likelihood function. The procedure is computationally cheap and works well for simulated iid data. In section 5.5.5, the procedure is assessed directly in the MCMC sampling context.

The estimated MN densities $f_j(\theta^j|\hat{\lambda}_j)$ and the degrees of freedom v of the t copula are then used to obtain latent variables

$$x^{ji} = T_v^{-1}(F_j(\theta^{ji}|\hat{\lambda}_j)),$$

for $j = 1, \ldots, n_\theta$ and $i = 1, \ldots, \tilde{R}$. By the integral transform x^{ji} has a univariate t distribution, $t(0,1,v)$.

Next, the latent variables x^{ji} are used to obtain an estimate, $\hat{\Sigma}$, of the covariance matrix $\tilde{\Sigma}$. Finally, the degrees of freedom, v, is updated by selecting the v which maximises [5.12] on a grid $v \in \{3, 5, 10, 1000\}$, where $v = 1000$ represents a Gaussian copula. The evaluation of the density q_3 can be inferred from the steps above. The details are given in Silva et al. (2008).

Generating a sample

To generate a sample from [5.12], we proceed as follows. First a vector $x = (x^1, \ldots, x^{n_\theta})^T$ is drawn from the multivariate t distribution with mean 0, covariance $\hat{\Sigma}$ and degrees of freedom v. Each latent variable, x^j, is transformed into a mixture of normal draw, θ^j, via

$$\theta^j = F_j^{-1}(T_{1,v}(x^j|0,1)),\ j = 1, \ldots, n_\theta,$$

which can be achieved by numerically solving

$$T_{1,v}(x^j|0,1) = F_j(\theta^j|\lambda_j),$$

either using a nonlinear equation solver, e.g. Newton-Rhapson, or by tabulating the implicit relationship between x^j and θ^j on a grid for each parameter $j = 1, \ldots, n_\theta$. In our implementation of the algorithm, we use the former approach.

[4] The restrictions are meant to prevent degenerate components with near-zero variances that are likely to otherwise arise in a MCMC setting (Kohn and Giordani (2008)).

Table 5.1 Sampler component acronyms

RWM, Random walk, eq. [5.8], [5.9], [5.10] $\alpha = \beta = 1$, $i_0 = \infty$,$\Sigma_0 = -H_m^{-1}$, $\kappa_1 = 2.38^2/n_\theta$, $n_\theta = \dim(\theta)$
ARWM, Adaptive random walk, eq. [5.8], [5.9], [5.10] $\alpha = 0.95$, $\beta = 1$, $\kappa_1 = 2.38^2/n_\theta$,$\kappa_2 = 9\kappa_1$, $i_0 = 100$, $\bar{\theta}_0 = \theta_m$, $\Sigma_0 = -H_m^{-1}$
ARWM3C, Adaptive random walk, 3 components, eq. [5.8], [5.9], [5.10] $\alpha = 0.95$, $i_0 = 0$, $\kappa_1 = 2.38^2/n_\theta$, $\kappa_2 = 9\kappa_1$, initially $\beta = 0$, then $\beta = 0.95$
UIMH, Unimodal symmetric independence, eq. [5.11], [5.9], [5.10] $i_0 = \infty$, $\bar{\theta}_0 = \theta_m$, $\Sigma_0 = -H_m^{-1}$, $v = 10$ (when $\alpha_2 = 1$), $v = \infty$ ($\alpha_2 < 1$)
AUIMH, Adaptive unimodal symmetric independence, eq. [5.11], [5.9], [5.10] $i_0 = 1,000$, $\bar{\theta}_0 = \theta_m$, $\Sigma_0 = -H_m^{-1}$, $v = 10$ (when $\alpha_2 = 1$), $v = \infty$ ($\alpha_2 < 1$)
TC, t copula with mixture of normal marginals, estimated using clustering
GC, Gaussian copula with mixture of normal marginals, estimated using clustering

5.3.5 Sampler acronyms

In this subsection, the sampler acronyms are briefly explained. The components of the samplers are defined in table 5.1. The RWM-UIMH and the adaptive ARWM-AUIMH are the hybrid samplers where the fraction of (A)RWM steps is α_1 and the fraction of (A)UIMH proposals is $\alpha_2 = 1 - \alpha_1$, and these shares remain fixed for the entire chain.

The main sampler is denoted ARWM-AUIMH-TC. Here, a fraction α_3 of the proposals are generated by the TC component, and this share is increasing through the chain as described above. In the tables of results below, the overall fraction of draws from the main component for the chain is reported. The first two components each generate a fraction $\alpha_1 = \alpha_2 = (1-\alpha_3)/2$ of the draws. In the RWM-TC sampler $\alpha_1 = 1 - \alpha_3$ and $\alpha_2 = 0$, and α_3 is increased gradually.

5.4 Parallel adaptive hybrid MCMC

An important advantage of independence MH samplers over RWM samplers is their suitability for parallel implementation. The purpose of this section is, first, to reiterate this fundamental point and, second, to explain how parallelisability is affected by adaptation and hybridisation, i.e. the introduction of RWM steps into an otherwise independence chain sampler. A good introduction to parallel MCMC, albeit from a different perspective, is provided by Wilkinson (2006).

Parallel independence chain samplers are attractive in most hardware contexts and we provide two examples of environments which are believed to be relevant to economists and econometricians. First, for personal high performance computing (PHPC) environments, characterised by relatively few, say between 4 and 8, processor cores and perhaps limited network performance. An example is use of Matlab and its parallel functionality (the Parallel Computing Toolbox) on a multicore desktop computer. Second, for massively parallel computing (e.g. $P > 50$ processors)

in high performance computing (HPC) environments, characterised by reasonably fast interconnection networks, allowing for fast interprocessor communication. For convenience the discussion is restricted to single block Metropolis-Hastings samplers, which are most suitable for parallelisation.

Let K be the number of synchronisation points in a parallel algorithm, i.e. points of communication between processes/processors (the terms are used interchangeably here). We loosely define a limited communication (LC) parallel MCMC algorithm as an algorithm for which the number of posterior draws per synchronisation point, R/K, is 'large', where R is the total number of draws from the posterior. Therefore LC algorithms are algorithms which should display good scalability properties on a large variety of parallel computers, such that hardware issues can largely be abstracted from. The typical measure of scalability, or parallel efficiency, is the *relative speedup*

$$S(P) = \frac{T(1)}{T(P)} \leq P$$

where $T(P)$ is the wall-clock time of running the parallel program on P processors. An alternative measure, conveying the same information, is the *relative efficiency*, $S(P)/P$.

The parallel approach based on regeneration discussed by Brockwell and Kadane (2005) is an example of LC parallel MCMC whereas the 'parallel blocks' approaches of Wilkinson (2006) and Whiley and Wilson (2004) and the prefetching approach of Brockwell (2006) and Strid (2010) are better described as communication intensive.

Two factors determine whether an adaptive MCMC algorithm can be implemented as an LC parallel algorithm: the proposal density and the intensity of adaptation. Consider first the form of the proposal density in the non-adaptive Metropolis-Hastings case. We can distinguish two main classes of algorithms: independence samplers, and samplers where the proposal density depends on the current state of the chain, e.g. the RWM sampler. For a single-block independence sampler a trivial two-stage parallel algorithm can be constructed. In the first stage, proposals for the complete chain are evaluated in parallel. Using a homogeneous parallel computer, i.e. one where all processors have the same performance, each process generates and evaluates R/P proposed parameters and a master process collects the results. In the second stage, the MH algorithm is run serially by the master process with the posterior values at the proposed parameters available from the first stage. This parallel independence MH algorithm displays extremely good scalability. Furthermore, parallel efficiency is largely independent of the properties of the parallel computer since there is only one synchronisation point in the algorithm. It is also easy to adapt the algorithm to the less relevant case of inhomogeneous processors.

Employing $P = 64$ processors on a standard HPC cluster to estimate a medium-scale DSGE model with this simple algorithm and a t proposal distribution, Strid (2010) reports the speedup $S(64) = 63$. The unreliable statistical performance of the independence sampler in that example, however, implies that it is an extremely wasteful approach. We return to this model below.

The single-block RWM or ARWM algorithm has $K = R$ such that limited communication parallel implementation is precluded. Parallel prefetching RWM or

ARWM algorithms can be implemented with $K < R$ but these implementations will remain communication intensive. Using $P = 8$ processors for a close to optimal prefetching RWM algorithm Strid (2010) reports $R/K \approx 5$. Prefetching algorithms can be expected to perform well on clusters with a fast interconnection network and when few, maximally $P = 10 - 15$, processors, are employed.

In the adaptive sampling framework, the frequency of adaptation, i.e. how frequently the proposal density is updated, is also of importance. Naturally, an adaptation point implies a synchronisation point, such that $M \leq K$ where M is the number of adaptation points. If adaptation is performed at each iteration ('online') we have $K = M = R$. The 'pure', i.e. not hybrid, adaptive independence sampler with batch estimation of the proposal density is characterised by $K = M << R$. Kohn and Giordani (2008) suggest $R/M = 1000$ in their implementations, abstracting from a short initial phase where updating of the proposal is more frequent. It is immediately clear that a parallel version of their batch updating adaptive independence algorithm will display very good parallel efficiency. (Note, however, that the adaptation schedule was motivated solely on statistical grounds rather than being based on parallel efficiency concerns.)

Three basic requirements for limited communication single-chain parallel adaptive MCMC algorithms follow. First, the majority of the draws must be based on independently generated proposals. Random walk steps may be used but parallel efficiency will suffer. Second, adaptation can not be too frequent. In the context of adaptive independent samplers with batch updating this does not appear to be restrictive. Third, deterministic hybrid schemes should be used, rather than random mixture hybrids, in order to avoid randomly distributed synchronisation points in the algorithm[5].

The key trade-off in our framework is related to hybridisation, i.e. the insertion of random walk steps. In practise, we expect the random walk steps to be much more frequent than the adaptation points and hence the number of synchronisation points, K, will coincide with the number of random walk steps. From a statistical perspective, these are expected to make the sampler more robust, reliable and efficient in medium and high dimensional settings. From the parallel computing perspective, a random walk step implies a synchronisation point, i.e. interprocessor communication, which is costly.

Based on these considerations, we construct an attractive simple benchmark parallel hybrid single-block MCMC algorithm. The algorithm combines draws from an independence proposal, q_2, which approximates the posterior well on most of its support (e.g. the t copula used in this paper), with occasional RWM or 'always reject' prefetching RWM steps, q_1, in a deterministic cycling scheme[6]. The prefetching RWM step is simply the parallel analogue of the RWM step. With P processors, in the 'always reject' variant of prefetching the posterior kernel is evaluated in parallel at

[5]While randomly distributed synchronisation points does not change the theoretical parallel efficiency properties of the hybrid algorithm, it complicates the implementation.

[6]For a sampler with several independence components, e.g. the main sampler presented in section 5.3, all the independence components are here assumed to be contained in q_2.

the points

$$\theta^p = \theta_i + \varepsilon^p,\ p = 1, \ldots, P, \tag{5.14}$$

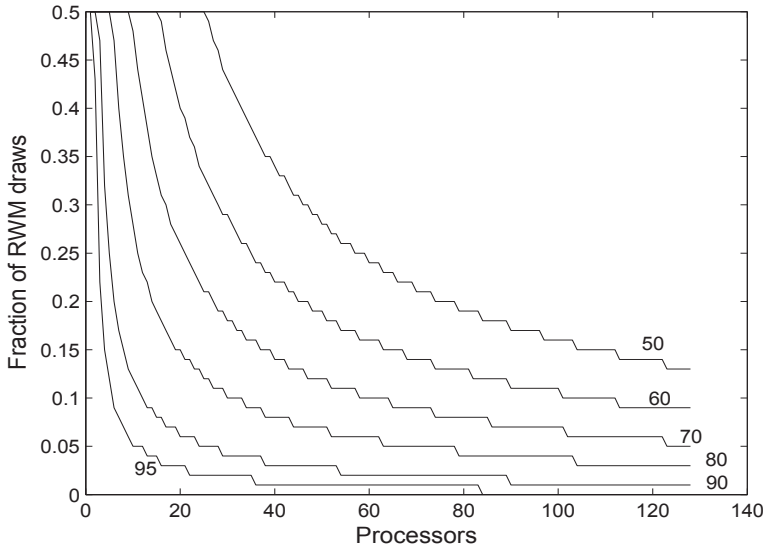
where θ_i is the current state of the chain. Intuitively, the 'always reject' prefetching RWM step either allow us to escape a 'difficult point', which is desirable from a Markov chain efficiency perspective, or it does not. In the latter event, the prefetching step will 'at least' be computationally efficient, since all P posterior evaluations generate draws. Pseudo-code for the algorithm is given in appendix B.

Ultimately, overall efficiency (i.e. combined Markov chain statistical efficiency and parallel computational efficiency), as measured by iid equivalent draws per time unit, hinges mostly on the quality of the independence proposal. The hybrid sampler consists of the random walk component, q_1, and an independence proposal component, q_2, such that $\alpha_1 + \alpha_2 = 1$, where α_1 and α_2 are the fractions of the components. Let $f(\alpha_1)$ denote the number of iid equivalent draws per time unit obtained with the sampler when a single processor ($P = 1$) is applied with the algorithm. Our discussion rests on the assumption that f is concave, such that there exists an interior value for α_1, $0 < \alpha_1^* < 1$, which maximises f. The shape of f and the optimal value, α_1^*, is related to the quality of the independence proposal, q_2. A 'good' independence proposal increases Markov chain efficiency and therefore allows for infrequent RWM steps, i.e. it shifts the function f upwards and lowers the optimal value α_1^*, which in turn boosts parallel efficiency by reducing the number of synchronisation points, $K = \alpha_1 R$, in the algorithm.

In the parallel setting, the number of iid equivalent draws per time unit for the hybrid algorithm is given by $S(P, \alpha_1) \times f(\alpha_1)$, where $S(P, \alpha_1)$ is the speedup of the algorithm. Assume that a suitable fraction of draws from the random walk component, α_1, has been established on statistical grounds, for a given choice of independence component, q_2, and for the case $P = 1$. It would then presumably be possible to increase the overall efficiency by decreasing α_1 further when $P > 1$. If α_1 is decreased, the speedup of the hybrid algorithm, $S(P, \alpha_1)$, is increased (see below). Decreasing α_1 when the fraction of random walk proposals is initially below the optimum ($\alpha_1 \leq \alpha_1^*$), implies that the Markov chain efficiency, as measured by $f(\alpha_1)$, is reduced, due to the concavity of f. In this case there is a trade-off between statistical and parallel efficiency. Assume, for example, that we manage to choose α_1 optimally in the $P = 1$ case, i.e. $\alpha_1 = \alpha_1^*$. The possibility of increasing the overall efficiency by lowering α_1 in the $P > 1$ case then depends on the curvature of the functions S and f. This may be compared with the 'always reject' RWM prefetching algorithm, where increased parallel efficiency is traded against lower Markov chain efficiency. The optimal scaling factor, κ_1, in [5.8], is smaller with $P > 1$ processors than with $P = 1$ processor (Strid (2010)).

In figure 5.2, theoretical relative efficiency contours, $S(P, \alpha_1)/P$, of the algorithm are plotted as a function of the fraction of RWM proposals, α_1, and the number of processors, P. The assumptions and calculations underlying the figure are described in appendix B. For example, if $\alpha_1 = 10\%$ and $P = 64$ the theoretical relative efficiency is roughly 70%, implying that the theoretical speedup is $S(64, 0.10) = 45$. In practise, the speedup would be lower, but the numbers are still indicative of the enormous

Figure 5.2 Relative efficiency contours, $S(P)/P$, for the parallel adaptive and hybrid Metropolis-Hastings algorithm (in percent).



computational gains associated with 'largely independence' samplers.

Finally, for samplers with the TC component, we note that the adaptation step, i.e. the mixture of normals estimation, is easy to parallelise since it is performed independently for each parameter θ^j. In general $\text{mod}(n_\theta, P) \neq 0$ implying that, on a homogeneous parallel computer, each process performs adaptation for $floor(n_\theta/P)$ or $ceil(n_\theta/P)$ parameters. Possible implications are that we may want to perform adaptation more frequently and/or spend more resources on obtaining a good proposal distribution.

5.5 Algorithm comparison

5.5.1 A Smets and Wouters type model

The statistical performance of the sampling algorithms is first compared using one of the core macroeconomic models at the European Central Bank, a slightly simplified version of the Smets and Wouters (SW) model (Smets and Wouters (2003)). The nominal frictions in the model are sticky prices, sticky wages and money demand by both households and firms. The real frictions are monopolistic competition in the markets for intermediate goods and labor, internal habit formation in preferences for consumption, investment adjustment costs and capital utilisation costs. In our version of the model, there are $n_\epsilon = 8$ structural shocks: consumption preference, labor

supply, stationary technology, unit root technology, government spending, monetary policy, inflation target and investment adjustment cost shocks. Similar models have been analysed and/or estimated in many studies (Smets and Wouters (2003); del Negro et al. (2005)).

The number of state variables in the model is $n_x = 15$ and $n_\theta = 23$ parameters are estimated (as customary a subset of the structural parameters are calibrated, e.g. parameters which are not well identified by the data). The model is estimated on Euro Area data for the period 1982Q1-2003Q4 (88 observations). The $n_y = 5$ data series used for estimation are the short-term interest rate, inflation, output growth, the consumption-to-output ratio and the investment-to-output ratio.

The model is solved using the method of Klein (2000) and the Kalman filter is initialised using the stationary distribution of the state variables, which is obtained by solving the Lyapunov equation, [5.4], numerically. The prior distribution is 'standard' in two senses: first, parameters which are unbounded, bounded from below and bounded both from below and above have normal, (inverse) gamma and beta prior distributions, respectively. Second, we have attempted to select hyperparameters for these distributions which are similar to those used in previous studies based on similar models.

Chains are obtained for (at least) two initial values per algorithm, the (presumed) posterior mode, θ_m, and the prior mode, θ_{prior}. The log posterior is optimised using simulated annealing (SA), with the set of optimisation parameters 'conservatively' chosen such that the mode is slowly approached, and using Christopher Sims' optimiser *csminwel*, for a variety of starting values drawn from the prior distribution (Sims (1999)). The log posterior kernel (and likelihood) values at θ_m and θ_{prior} are 1754.32 (1768.15) and 1630.69 (1632.06) respectively. The initial value for the chain does not influence the algorithmic comparison and therefore results are provided for the chains initialised at the prior mode. For each algorithm, $R = 500,000$ draws are obtained and the first $100,000$ are discarded as burn-in.

The empirical optimal acceptance rate for the standard RWM algorithm is in the range $0.20 - 0.24$, based on the average inefficiency factors from 10 chains with acceptance rates in the interval $0.02 - 0.40$. In table 5.2, the inefficiency factors for the algorithms are presented. The inefficiency factors are given by

$$\tau_j = 1 + 2\sum_{k=1}^{K} Corr\left(\theta_i^j, \theta_{i+k}^j\right), j = 1, \ldots, n_\theta, \tag{5.15}$$

where we let $K = 500$. This measure indicates how many draws are needed using a given algorithm to achieve the same numerical efficiency as when (hypothetically) sampling an independent draw from the posterior distribution.

The main observations are as follows. First, all samplers obtain the same posterior distribution. In the case of the UIMH sampler, the inefficiency factors differ in repeated runs and only the best result, i.e. the run which produced the lowest mean inefficiency, is reported here[7]. This is due to the randomness in the occurrence of difficult points, see also the discussion in Adolfson, Lindé and Villani (2007*a*).

[7] We have recorded significantly larger inefficiency factors for the UIMH sampler in other runs.

Table 5.2 Inefficiency factors and acceptance rates for the SW type model

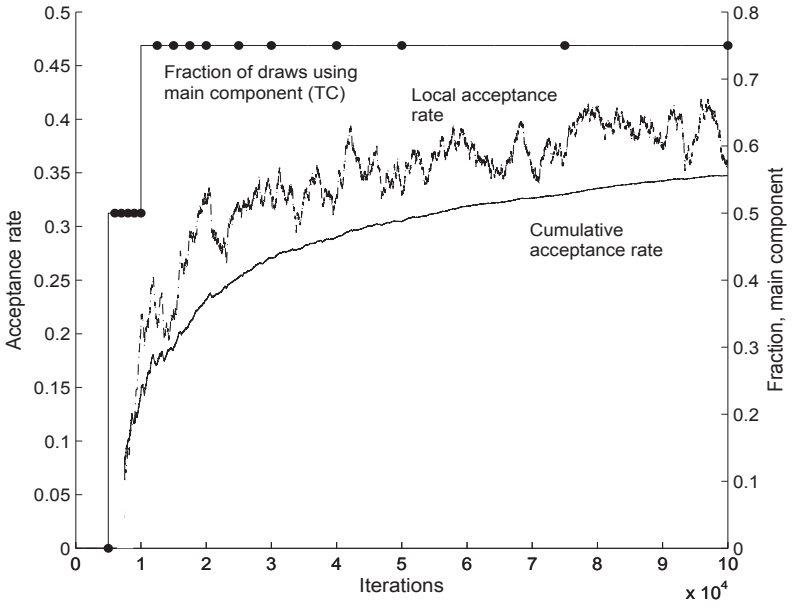
Algorithm	Accept. rate (%)	Inefficiency Min	Median	Mean	Max	Fraction main (%)	Max clus.
RWM	24.2	66.0	85.4	99.2	163.9	-	-
UIMH (t, df=10)	17.0	62.6	104.8	141.3	410.1	-	-
ARWM3C	24.0	83.3	104.0	105.7	140.6	-	-
RWM-UIMH	24.6	31.2	48.6	64.1	126.1	50 (UIMH)	-
ARWM-AUIMH	21.8	20.2	22.8	23.9	32.3	50 (AIMH)	-
RWM-TC	37.7	9.2	14.6	15.0	36.8	74.0 (TC)	4
RWM-UIMH-TC	44.5	8.2	15.2	17.0	57.7	95.6 (TC)	4
ARWM-AUIMH-TC	44.0	8.7	12.9	14.1	40.6	95.6 (TC)	4

Second, the mean (or median) inefficiency for the main sampler, the ARWM-AIMH-TC hybrid algorithm, is roughly a factor $6-7$ lower than for the RWM algorithm. Note again that the model parameters have been transformed in the 'standard' way, described in section 5.2. This improves the performance of the RWM algorithm. In the original parameterisation, the difference in inefficiency factors would presumably be larger. Third, the choice of initial sampler for the hybrid algorithm does not appear crucial. The ARWM-AIMH initial sampler delivers a better approximation to the posterior, q_3, quicker but the simpler RWM initial sampler yields similar inefficiency factors. Fourth, the adaptation schedule used with the RWM-UIMH-TC and ARWM-AUIMH-TC samplers is displayed in figure 5.1 above. In our experience, it does not appear to be important to get the adaptation and hybrid schedules, A and H, exactly 'right', i.e. the required tuning of the schedules was very limited. If, initially, a too intensive schedule is used, it will be revealed quickly by the acceptance rate of the main component, q_3, and the schedule can be adjusted. Fifth, the simple ARWM-AIMH sampler reduces inefficiency factors by roughly a factor 4 in comparison with the RWM algorithm.

In figure 5.3, the performance of the hybrid RWM-TC sampler is illustrated graphically using the first $100,000$ draws from the sampler, corresponding to the burn-in phase. The acceptance rate of the main component is displayed along with the fraction of draws from the main component. The adaptation points are marked as dots on the hybrid schedule. The acceptance rate is also calculated for a moving window of $2,000$ draws to provide an idea of how the quality of the main component improves, a 'local' acceptance rate. Adaptation starts after $i_A^1 = 5,000$ draws and is thereafter frequent. The main component generates 50% of the draws up to $i_A^6 = 10,000$ and after that point it accounts for 75% of the proposals. At the end of the run, 74% of the proposals have been generated by the TC component. The acceptance rate improves quickly during the intensive adaptation phase and after roughly $30,000$ draws it appears that a good proposal has been found. The acceptance rate of the TC component is initially below 10% and after $30,000$ iterations it has

Furthermore, the recorded inefficiencies vary substantially with the choice of the degrees of freedom for the t density.

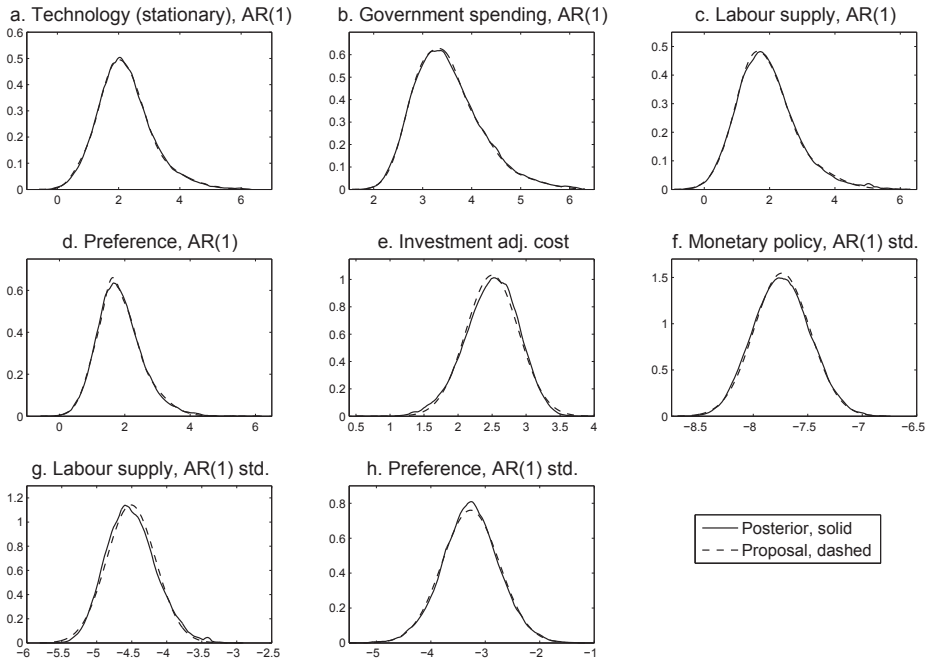
Figure 5.3 Performance of the RWM-TC sampler for the SW type model.



reached roughly 30%. The quality of the posterior approximation improves at a slower rate subsequently and adaptation becomes less frequent. Eventually, the acceptance rate of the main component settles down at 44%, which can be compared to the 17% acceptance rate obtained with a standard independence sampler which uses a t proposal (UIMH).

The final proposal distribution for the samplers with a TC component, which is constructed at iteration 150,000, has more than one component in the mixture of normal marginal density for 5 of the 23 estimated parameters. The degrees of freedom estimate for the copula in [5.12] is $v = 1000$ for the samplers with a TC component, i.e. a Gaussian copula is selected by the adaptive procedure. Four of these five parameters are $AR(1)$ coefficients of shock processes, with beta prior distributions on the interval $[0, 1]$. The marginal posterior density (Epanechnikov kernel smoothing density estimate) and the final mixture of normal density estimate for these parameters are displayed in figure 5.4, panels a-d (in the transformed parameter space). The marginal proposals appear to provide a good fit to the marginal posteriors, which are skewed to the right for these parameters. Intuitively, the improved ability to fit these skewed parameters explain the difference in sampling performance between the samplers with a TC component, and the (A)RWM-(A)UIMH samplers. In panels *e*-*h*, the proposal and posterior densities are shown for the four parameters which, by visual inspection, display the largest discrepancy between the marginal proposal and posterior densities.

Figure 5.4 Marginal posterior and mixture of normal proposal distributions, SW type model.



For these parameters the final marginal proposal densities are normal distributions, i.e. the BIC criterion selects one component MN densities. The marginal posterior distributions of the remaining 15 parameters, those which are not displayed here, are symmetric in shape and apparently well approximated by a normal distribution.

5.5.2 The RAMSES model

The RAMSES model is a large-scale DSGE model developed and estimated by researchers at the Swedish central bank (Adolfson, Laséen, Lindé and Villani (2007); Adolfson, Lindé and Villani (2007*b*); Adolfson, Laséen, Lindé and Villani (2008*a*); Adolfson, Laséen, Lindé and Villani (2008*b*)). It is essentially the extension of the SW model to a small open economy setting. A number of interesting econometric issues related to the estimation of the model are discussed in Adolfson, Lindé and Villani (2007*a*). The analysis here, is restricted to a comparison of sampling performance and the reader is referred to the mentioned papers for a detailed discussion of the content of the economic model, variations of the model, the data used for estimation and the choice of prior distribution.

Table 5.3 Inefficiency factors and acceptance rates for the RAMSES model

Algorithm	Accept. rate (%)	Inefficiency Min	Median	Mean	Max	Fraction main (%)	Max clus.
RWM	22.7	27.4	41.2	67.2	748.4	-	
UIMH (t, df=10)	8.0	125.8	455.6	459.1	948.2	-	
RWM-UIMH	16.7	26.9	41.4	92.3	830.5	50%(IMH)	
ARWM-AUIMH	16.1	13.0	15.6	22.4	135.3	50%(AIMH)	
RWM-TC	19.8	7.9	11.2	14.2	49.5	77.5%(TC)	4
ARWM-AIMH-TC	23.5	8.3	12.8	14.8	33.7	77.5%(TC)	4

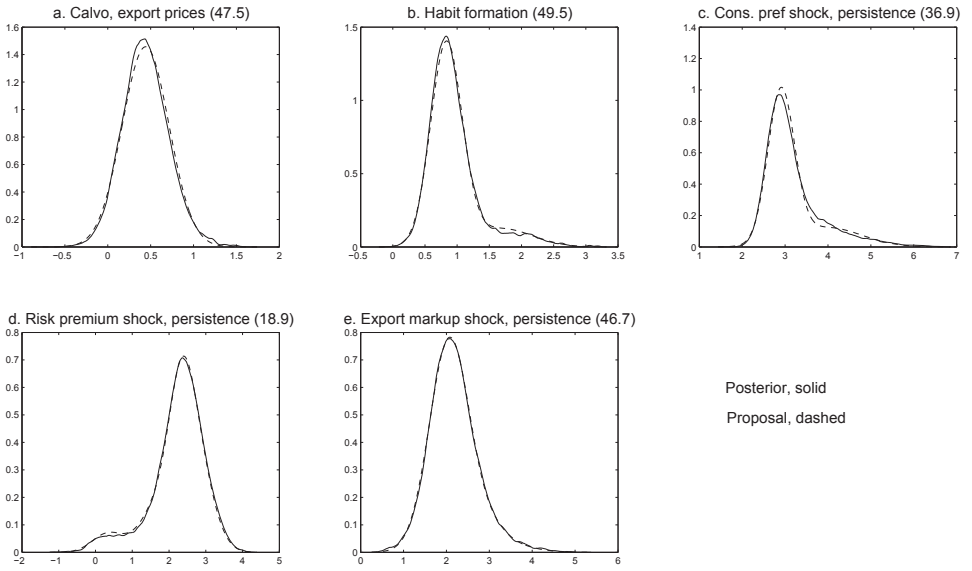
The LGSS model contains $n_x = 60$ state variables and $n_\epsilon = 21$ shocks. The model is estimated using Euro Area data for the period $1970Q1 - 2002Q4$ and $n_y = 15$ observed series are used to estimate $n_\theta = 51$ of the model's structural and auxiliary parameters. Again a number of parameters are calibrated. In Adolfson, Laséen, Lindé and Villani (2007) the model is estimated with and without variable capital utilisation. Here only the former case is considered.

The model is solved using the AIM algorithm. The initial, $1970Q1$, state distribution for the Kalman filter is diffuse. The observations for the $1970s$ are only used to obtain a prior for the state variables in $1979Q4$ and inference is based on data for the period $1980Q1 - 2002Q4$. The prior distribution is standard in the sense described in the previous section.

For each algorithm, $1,000,000$ draws are obtained and every 5^{th} is kept for inference. The burn-in is $200,000$ draws. Inefficiency factors and acceptance rates for the various samplers are presented in table 5.3, calculated based on the thinned chains and using the cut-off $K = 1,000$ in [5.15]. The performance ranking of the algorithms resemble the one for the smaller DSGE model above. All samplers find the same, and therefore presumably correct, posterior distribution (with the UIMH algorithm possibly being an exception). The median or mean inefficiency factor is roughly $3 - 4$ times lower for the samplers with a TC component when compared to the benchmark RWM algorithm.

The final proposal distributions of the RWM-TC and ARWM-AIMH-TC samplers, constructed at iteration $500,000$, have more than one component in the mixture of normal density for 8 out of the 51 parameters and, notably, 7 of those 8 parameters have beta prior distributions. In these instances, the marginal posterior distributions are apparently skewed, even after reparameterisation. Furthermore, among the 25 parameters with beta prior distributions, these 7 are among those with the most skewed prior, e.g. shocks which are a priori thought to be highly persistent. These eight multi-component, or 'problematic', parameters also correspond to those with the largest chain autocorrelations when the RWM algorithm is used. The median inefficiency factor for this subgroup of parameters is 180, which can be compared to the overall median inefficiency factor of 41 presented in the table. For the RWM-TC sampler, the median inefficiency for the group is 24, to be compared with the overall median of 11.2.

Figure 5.5 Marginal posterior and mixture of normal proposal distributions for the five parameters with largest inefficiency factors, RAMSES model.

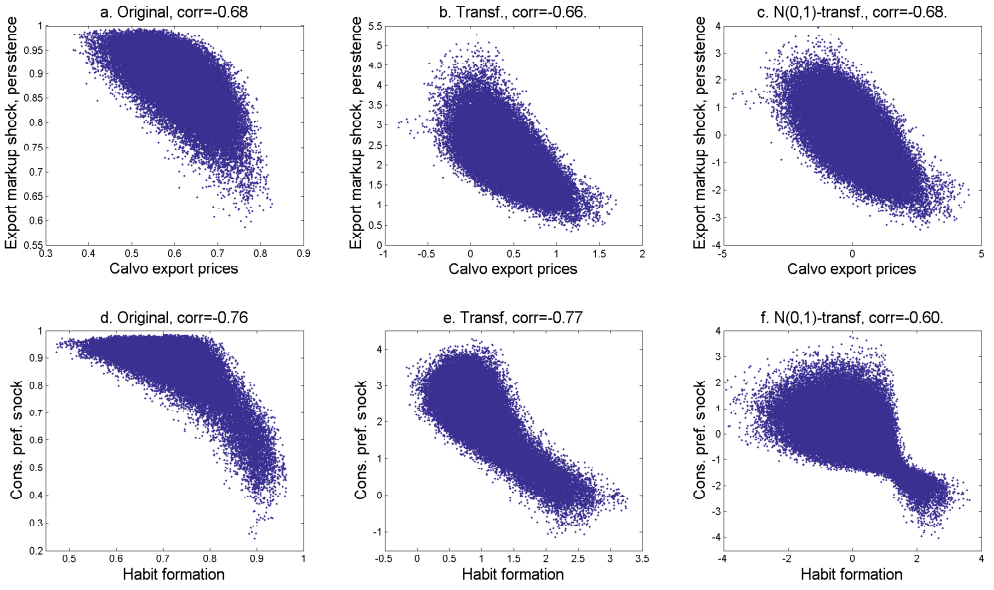


In figure 5.5, the marginal posterior (again the Epanechnikov kernel smoothing density estimate) and the final RWM-TC marginal proposal distributions for the five parameters with the largest RWM-TC sampler inefficiency factors are displayed (with the inefficiency factors, τ_j, in parenthesis). Note that these are also the parameters with the largest RWM inefficiency factors. Clearly, the marginal posterior densities with prolonged tails in panels b, c and d are not possible to approximate well with a symmetric proposal density. The reduction in inefficiency factors when using the TC-component samplers is most dramatic for these parameters.

These parameters also display irregular posterior dependencies, which may be illustrated via bivariate posterior plots. The correlation between the export parameters, the Calvo export price stickiness (ξ_x, in the notation of Adolfson, Laséen, Lindé and Villani (2007)) and the export markup shock persistence (ρ_{λ^x}) parameters, is -0.66 and the correlation between the consumption parameters, the habit formation (b) and preference shock (ρ_{ζ^c}) parameters, is -0.77. In both instances, these correlations can be intuitively explained, e.g. concerning the export parameters the lack of appropriate export price data explains the weak identification of these parameters (Adolfson et al. (2008*a*), p. 8).

In figure 5.6, bivariate plots of the draws from the RWM-TC sampler for these two pairs of parameters are displayed. In the left part of the figure, panels a and d, the parameters are displayed in the original (bounded) parameterisation and in the middle panels, panels b and e, in the standard (unbounded) transformation

Figure 5.6 Bivariate posterior plots for two pairs of parameters, RAMSES model.



described previously. For both pairs of parameters reparameterisation straightens out the banana-shaped bivariate posterior surface somewhat but the shape is still non-elliptical.[8]

In the final proposal distribution, the degrees of freedom estimate for the copula in [5.12] is, as for the SW type model above, $v = 1000$ for the samplers with a TC component, i.e. a Gaussian copula is selected endogenously by the adaptive procedure. In the right part of the figure, panels c and f, the parameter draws have been transformed to $N(0,1)$-variables via the relation [5.13]. The mixture of normal densities, used for these transformations, are estimated using all the $160,000$ post burn-in draws obtained from the RWM-TC sampler. The racquet shape of the transformed bivariate posterior in panel f is suggestive of the limits as to what can be achieved with a Gaussian copula approximation in this case. Although the marginal posteriors in figure 5.5 appear reasonably well approximated by mixture of normals densities their joint dependence does not appear well captured by the simple Gaussian copula.

It is not surprising that estimation of the RAMSES presents more of a challenge than estimation of the SW type model in the previous section, given the higher dimensionality of the parameter vector of estimated parameters. Figures 5.4 and 5.5, which display the subset of the most problematic parameters for each model, also

[8]The mean inefficiency factors of the RWM sampler reported by Adolfson, Lindé and Villani (2007*a*) based on $500,000$ draws (and no thinning) are 339.25 and 217.13 in the original and transformed parameter spaces, respectively.

suggest that the posterior distribution of the RAMSES model is further away from normality than the posterior of the smaller SW model.

In our experience, with the SW type model even a poorly designed adaptation schedule, i.e. too intensive given the number of parameters, works well, in the sense that the sampler is able to recover from a bad start and eventually obtains the posterior distribution. For the RAMSES model, it is more important to choose a slower intensity of the adaptation scheme: if adaptation is too intensive the sampler will not be able to recover from an initial poor approximation of the posterior.

5.5.3 Nonlinear estimation of a small-scale DSGE model

In this section the sampling algorithms are applied to the estimation of a small-scale second order approximated DSGE model. The likelihood function is estimated using a standard particle filter, see e.g. Arulampalam et al. (2002). The computational cost of particle filtering implies that the models that are considered for nonlinear estimation in practise are, typically, small. This, in turn, implies that the chances of success for a one-block MCMC approach increases, since the dimension of the parameter vector, n_θ, is smaller. (An exception is the model estimated nonlinearly recently by Fernández-Villaverde et al. (2010), where $n_\theta = 30$.)

The prototypical small New Keynesian (NK) model is borrowed from An (2005). The state-space representation of the model is given by the state equation, [5.5] and [5.6], and the observation equation [5.3], which are obtained by first solving the model using the approach of Schmitt-Grohe and Uribe (2004). The model contains $n_{x_1} = 3$ shocks, a total of $n_x = 7$ state variables and $n_y = 3$ observed variables.

Since the main focus here is on comparing sampling performance, we choose to use simulated data, in order to abstract, as much as possible, from issues related to particle filtering performance. A bad model fit or the presence of outliers in the data, for example, can have a severe impact on the performance of the filter, which would prevent a meaningful comparison of samplers. The construction of improved particle filters to be able to deal with such issues, although a very important topic, is beyond the scope of this article.

The data-generating process (DGP), i.e. the calibration of the model parameters, and the prior distribution for $n_\theta = 13$ estimated parameters is chosen largely as in An (2005). Data for $n_y = 3$ series - output growth, inflation and the nominal interest rate - is simulated from the nonlinear state-space model for $T = 90$ time periods. In the estimation, the first 10 periods of the time series are merely used to provide a prior for the state distribution at time $t = 10$ and the remaining 80 periods are used for inference. The model is estimated with the variance of the measurement errors, $\Sigma_v = diag(\Sigma_v)$, calibrated at values which imply that roughly $15 - 20\%$ of the variance of the observed series is due to these errors. (Note that the variance of the measurement errors affects the performance of the particle filter.) The data set generated by the DGP and the prior distribution imply a unimodal posterior distribution.

The key structural parameters for generating nonlinearity in the model are, first, the inverse elasticity of intertemporal substitution, τ, (which determines the

curvature of the household's utility function) and, second, the inflation response of the monetary authority, ψ_1. The degree of nonlinearity increases as τ increases and as ψ_1 approaches the boundary of the indeterminacy region, $\tilde{\psi}_1 = 1$ (from above). In the DGP, these parameters are assigned the values $\tau = 2$ and $\psi_1 = 1.5$, which implies a 'mildly nonlinear' parameterisation. By experimentation, we have found the remaining parameters to be largely unimportant in producing any substantial differences between the linear and nonlinear state-space models.[9]

In the nonlinear case, given that the PF yields only an approximation to the likelihood, it is not possible to obtain the exact mode of the posterior density. The standard approach in the linear case, i.e. to use the negative inverse Hessian evaluated at the posterior mode as the covariance of a RWM proposal, is therefore not available. The RWM sampler is instead initialised using the estimated covariance matrix from a preliminary run, a form of preliminary adaptation, and the chosen scaling factor implies an acceptance rate of 24.0% when the number of particles is $N = 40,000$. In the preliminary run, the scaled negative inverse Hessian at the posterior mode of the corresponding log-linear model is used as the covariance of the RWM proposal.

The inefficiency factors of three samplers, RWM, ARWM-GC and ARWM-AUIMH-GC, are presented, for chains, started at the prior mode. In preliminary runs with short chains, it is found that the TC copula selects the degrees of freedom $v = 1000$ and in the reported chains we fix this value, i.e. a Gaussian copula is pre-imposed. In each case, $R = 150,000$ draws from the posterior distribution are obtained and the first $20,000$ are discarded as burn-in. Pure independence samplers, UIMH (t with $v = 10$) and GC, were trapped at 'difficult points' for spells long enough to conclude that these samplers can not be applied successfully to this problem.

The relatively small number of parameters and, more importantly, the computational burden associated with particle filtering suggests a more intensive adaptation scheme, in comparison with the linear estimation examples above. Here, adaptation begins at iteration $i_1^A = 2,000$ and is performed every 500^{th} iteration up to $i_{17}^A = 10,000$. After that, re-estimation of the MN marginal proposals is performed every $1,000^{th}$ iteration up to $i_{37}^A = 30,000$, from there on every $2,000^{th}$ iteration up to $i_{67}^A = 90,000$ and for the remaining part of sampling every $10,000^{th}$ iteration. Note that even with such frequent adaptation, the re-estimation of the proposal density accounts for a negligible share of overall computational time.

In table 5.4, the inefficiency factors are presented. First, the performance of the RWM sampler using $N = 40,000$ and $100,000$ particles in the SPF is compared. The brute force approach of using more particles, in order to obtain a sharper estimate of the likelihood, apparently increases the acceptance rate and decrease the inefficiency factors somewhat. 'Difficult points' now occur *also* as a result of the sampling variability in the likelihood estimator. The standard deviation of the likelihood decreases at rate $\sqrt{N}$, such that increasing the number of particles reduces this problem, but at a high computational cost.

Second, the ARWM-GC and ARWM-AUIMH-GC samplers reduce the inefficiency

[9] By a 'nonlinear parameterisation' θ_{dgp} we, loosely speaking, mean a parameterisation for which data generated from the linear and quadratic state-space models, respectively, display different properties, e.g. the second moments of the data display clear differences.

Table 5.4 Inefficiency factors and acceptance rates for the NK model.

Algorithm	Accept. rate(%)	Inefficiency Min	Med.	Mean	Max	Frac. GC(%)	Max clus.	No. part.
RWM	24.0	79	104	117	217			40k
RWM	24.9	75	84	89	113			100k
ARWM-GC	31.9	12	23	32	86	86.8	4	40k
ARWM-GC	29.7	15	21	28	70	74.0	4	40k
ARWM-AUIMH-GC	30.1	14	17	20	36	74.0	4	40k

factors by roughly a factor 4 and $5-6$, respectively, in comparison with the RWM sampler. Two of the 13 parameters have more than one component in the final MN marginal proposal density. The fraction of copula-generated draws, α_3, of these samplers reaches its final level after $20,000$ iterations and for the ARWM-GC sampler a comparison is made for the cases where this fraction is 75% and 90%, respectively. The difference in inefficiency factors is quite small, suggesting that the exact fraction is largely unimportant, at least in a serial computing context (see the discussion below).

Finally, in our serial Matlab implementations of the samplers the time of a particle filter likelihood evaluation for $T = 90$ time periods and $N = 40,000$ particles is $2-2.5$ seconds depending on the hardware used such that $R = 150,000$ draws are obtained in roughly $80-100$ hours. Two Fortran mex routines are used for the parts of the particle filter that cannot be written as vectorised code, i.e. the systematic resampling procedure and the *vech* operator in [5.6]. These numbers are representative of what can be achieved with reasonably optimised code on a modern desktop computer. The particle filter accounts for more than 98% of estimation time and the time increases linearly in the number of particles, N.

5.5.4 Computing time

The computational time of estimating a large linearised DSGE model, or a nonlinearly approximated DSGE model of any size, is completely dominated by the likelihood evaluation time, implying that there cannot be any large differences in computing time between the samplers. A possible exception, in our context, is the main sampler, and the estimation of the t copula with mixture of normals marginals densities using clustering. Our current code implementation uses optimised code for the AIM solution routine (Matlab), Kalman filtering (Fortran) and particle filtering (Matlab and Fortran), while Klein's solution algorithm (Matlab) and the clustering algorithm (which is implemented in Matlab for this project) are largely non-optimised, e.g. loops are used too frequently. Still, in our examples the estimation of the mixture of normals accounts for a modest fraction of computing time, in comparison with the likelihood evaluation time.

In a serial computing context, a comparison of overall, statistical and computational, efficiency therefore largely reduces to a comparison of inefficiency factors, e.g.

comparing the RWM and RWM-TC algorithms

$$\frac{\bar{\tau}^{rwm}}{\bar{\tau}^{rwm-tc}}\frac{T_{rwm}(1)}{T_{rwm-tc}(1)} \approx \frac{\bar{\tau}^{rwm}}{\bar{\tau}^{rwm-tc}},$$

where $\bar{\tau}$ is the average inefficiency factor and $T(1)$ is the time of execution of the sampling algorithm on one processor. In a parallel computing context

$$\frac{\bar{\tau}^{rwm}}{\bar{\tau}^{rwm-tc}}\frac{T_{prf-rwm}(P)}{T_{rwm-tc}(P)} \approx \frac{\bar{\tau}^{rwm}}{\bar{\tau}^{rwm-tc}}\frac{S_{rwm-tc}(P)}{S_{prf-rwm}(P)} > \frac{\bar{\tau}^{rwm}}{\bar{\tau}^{rwm-tc}},$$

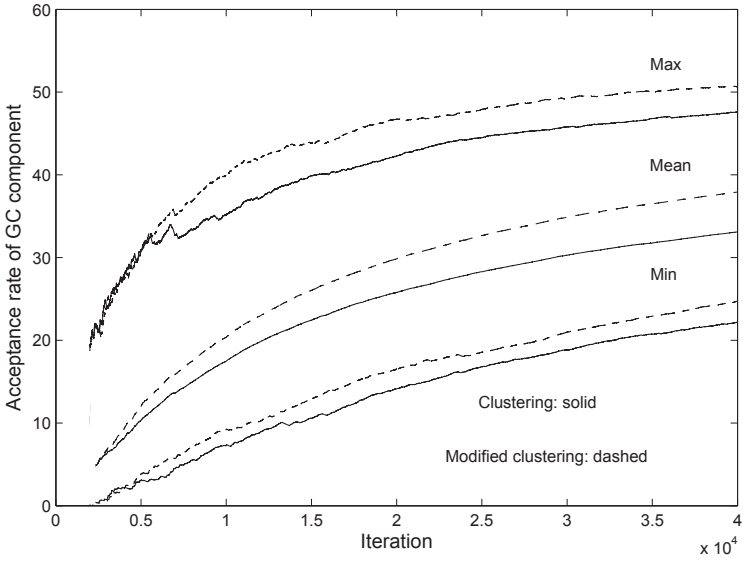
since the ratio of speedups satisfies $S_{rwm-tc}(P)/S_{prf-rwm}(P) > 1$, where $S_{prf-rwm}(P)$ is the speedup of the prefetching RWM algorithm and $S_{rwm-tc}(P)$ is the speedup of the parallel adaptive and hybrid algorithm outlined in section 5.4. The speedup ratio increases in P and decreases in the fraction of random walk steps, α_1. Here the speedup is compared with the prefetching RWM algorithm, based on the assumption that it is the best possible way of parallelising the RWM algorithm. In hardware contexts where prefetching is infeasible, due to immense costs of interprocessor communication, the relative attractiveness of the 'largely independence' approach increases, since then $S_{prf-rwm}(P) = 1$.

5.5.5 Modified estimation of the mixture of normal marginal proposal densities

In the final section of the paper, the modified mixture of normals estimation procedure described in section 5.3.4 is assessed via a simple experiment. The RWM-GC sampler is applied for estimation of the log-linearised version of the small-scale NK model in section 5.5.3. We generate 100 vectors of starting values from the fattened prior distribution and for each starting vector a pair of chains, each of length $R = 40,000$, are obtained. It is sufficient for the purposes here, to study the behaviour of the *GC* component only during the critical start-up phase of the chain. For the first chain in a pair, the clustering approach is used for estimation of the mixture of normal proposal densities, and in the second chain the modified clustering approach is applied. The data set generated by the DGP, the prior distribution and the adaptation schedule are the same as in the previous section, except that adaptation starts already at iteration $1,000$. There are 43 points of adaptation for a chain. The relatively intensive adaptation is desirable since the purpose of the experiment is to isolate the effect of the modified MN estimation procedure on sampling efficiency. The targeted acceptance rate of the RWM component is roughly 24%.

In figure 5.7, the minimum, mean and maximum (cumulative) acceptance rate of the *GC* component at each iteration is displayed for the clustering and modified clustering chains, respectively. The statistics are calculated at each iteration based on 100 observed acceptance rates, one per chain. In this stylised experiment, the acceptance rates of the chains which use modified clustering estimation is higher. Indirectly, this shows that the approximation to the posterior in the early stage of the chain is improved. At iteration $40,000$, the mean of the GC component acceptance

Figure 5.7 Acceptance rate (%) of GC component: min, mean and max for 100 chains, log-linear NK model.



rate is roughly 38% for the modified clustering chains and 33% for the clustering chains. For 99 of the 100 pairs of chains the acceptance rate of the modified clustering chain is larger at the final iteration. The number of parameters with more than one component in the MN density is typically 3 or 4 (out of $n_\theta = 13$) at the final iteration.

The experiment also illustrates the robustifying effect of the RWM step. The worst performing chain initially has a GC component acceptance rate very close to zero, but yet, as the RWM component continue to generate accepted proposals, it eventually manages to 'take off'.

5.6 Conclusions

Bayesian inference for DSGE models is typically carried out by single block random walk Metropolis sampling. The high computing costs involved are an obstacle to model development and use, and the poor mixing of RWM in high dimensions further increases computing times and reduces the probability of escaping local modes. This paper combines two features, parallelisation and adaptive independent Metropolis-Hastings, to achieve large computational gains in DSGE estimation. The history of the draws is used to continuously improve a t-copula proposal distribution. A novel twist is an adaptive random walk step inserted at predetermined intervals. Compared to a pure independent sampler, this step improves the ability of the chain to

escape difficult points of the type often encountered in medium and high dimensional DSGE models, while maintaining a high statistical efficiency. In linear estimation of a medium scale (23 parameters) and a large scale (51 parameters) DSGE model, the efficiency gains are $6-7$ and $3-4$ times respectively. In a nonlinear estimation example, where computing time is even more of a concern, the efficiency gains are $4-6$. The sampler is also better suited to parallelisation than RWM or blocking strategies, so that the effective computational gains can potentially be much larger. Our preferred algorithm employs a copula approach which, based on results by Silva et al. (2008), should provide even greater gains over RWM when using non-standard priors such as truncated or mixture distributions.

Bibliography

Adolfson, M., Laséen, S., Lindé, J. and Villani, M. (2007), 'Bayesian Estimation of an Open Economy DSGE Model with Incomplete Pass-Through', *Journal of International Economics* **72(2)**, 481–511.

Adolfson, M., Laséen, S., Lindé, J. and Villani, M. (2008*a*), 'Empirical Properties of Closed and Open Economy DSGE Models of the Euro Area', *Macroeconomic Dynamics* **12**, 2–19.

Adolfson, M., Laséen, S., Lindé, J. and Villani, M. (2008*b*), 'Evaluating an Estimated New Keynesian Small Open Economy Model', *Journal of Economic Dynamics and Control* **32**, 2690–2721.

Adolfson, M., Lindé, J. and Villani, M. (2007*a*), 'Bayesian Inference in DSGE Models - Some Comments', *Econometric Reviews* **26(2-4)**, 173–185.

Adolfson, M., Lindé, J. and Villani, M. (2007*b*), 'Forecasting Performance of an Open Economy DSGE Model', *Econometric Reviews* **26**, 289–328.

Amisano, G. and Tristani, O. (2007), 'Euro Area Inflation Persistence in an Estimated nonlinear DSGE model', *Journal of Economic Dynamics and Control* . Forthcoming.

An, S. (2005), 'Bayesian Estimation of DSGE Models: Lessons from Second-order Approximations'. Working Paper, University of Pennsylvania.

An, S. and Schorfeide, F. (2007*a*), 'Bayesian Analysis of DSGE Models', *Econometric Reviews* **26(2-4)**, 113–172.

An, S. and Schorfeide, F. (2007*b*), 'Bayesian Analysis of DSGE Models - Rejoinder', *Econometric Reviews* **26(2-4)**, 211–219.

Anderson, G. (2008), 'Solving Linear Rational Expectations Models: A Horse Race', *Computational Economics* **31(2)**, 95–113.

Anderson, G. and Moore, G. (1985), 'A Linear Algebraic Procedure for Solving Linear Perfect Foresight Models', *Economics Letters* **17(3)**, 247–252.

Andreasen, M. M. (2008), 'How to Maximize the Likelihood Function of a DSGE Model', *Computational Economics* . Forthcoming.

Arulampalam, S., Maskell, S., Gordon, N. and Clapp, T. (2002), 'A Tutorial on Particle Filters for On-Line Non-Linear/Non-Gaussian Bayesian Tracking', *IEEE Transactions on Signal Processing* **50(2)**, 174–188.

Bradley, P. S. and Fayyad, U. M. (1998), 'Refining Initial Points for K-means Clustering', *Proceedings of the 15th International Conference on Machine Learning (ICML98)* pp. 91–99.

Brockwell, A. (2006), 'Parallel Markov Chain Monte Carlo Simulation by Pre-fetching', *Journal of Computational and Graphical Statistics* **15(1)**, 246–261.

Brockwell, A. E. and Kadane, J. B. (2005), 'Identification of Regeneration Times in MCMC Simulation, with Application to Adaptive Schemes', *Journal of Computational and Graphical Statistics* **14(2)**, 436–458.

Chib, S. and Ramamurthy, S. (2010), 'Tailored Randomized-block MCMC Methods with Application to DSGE Models', *Journal of Econometrics* **155**, 19–38.

del Negro, M., Schorfheide, F., Smets, F. and Wouters, R. (2005), 'On the Fit and Forecasting Performance of New-Keynesian Models'. Working Paper Series, No. 491, European Central Bank.

Durbin, J. and Koopman, S. J. (2001), *Time Series Analysis by State Space Methods*, Oxford University Press.

Fernández-Villaverde, J., Guerron-Quintana, P. and Rubio-Ramírez, J. F. (2010), 'Fortune or Virtue: Time-Variant Volatilities Versus Parameter Drifting in US Data', *NBER Working Paper No. 15928* .

Fernández-Villaverde, J. and Rubio-Ramírez, J. F. (2007), 'Estimating Macroeconomic Models: A Likelihood Approach', *Review of Economic Studies* **74(4)**, 1059–1087.

Flury, T. and Shephard, N. (2008), 'Bayesian Inference Based Only on Simulated Likelihood: Particle Filter Analysis of Dynamic Economic Models'. Economics working paper 413, Oxford University.

Haario, H., Saksman, E. and Tamminen, J. (2001), 'An adaptive Metropolis algorithm', *Bernoulli* **7(2)**, 223–242.

Harvey, A. (1989), *Forecasting, Structural Time Series Models and the Kalman filter*, Cambridge University Press.

Klein, P. (2000), 'Using the Generalized Schur Form to Solve a Multivariate Linear Rational Expectations Model', *Journal of Economic Dynamics and Control* **24(10)**, 1405–1423.

Kohn, R. and Giordani, P. (2008), 'Adaptive Independent Metropolis-Hastings by Fast Estimation of Mixtures of Normals', *Journal of Computational Graphics and Statistics, Forthcoming* .

Lubik, T. A. and Schorfheide, F. (2004), 'Testing for Indeterminacy: An Application to U.S. Monetary Policy', *American Economic Review* **94(1)**, 190–217.

Roberts, G. O. and Rosenthal, J. S. (2001), 'Optimal Scaling for Various Random Walk Metropolis Algorithms', *Statistical Science* **16(4)**, 351–367.

Schmitt-Grohe, S. and Uribe, M. (2004), 'Solving Dynamic General Equilibrium Models Using a Second-Order Approximation to the Policy Function', *Journal of Economic Dynamics and Control* **28(4)**, 755–775.

Silva, R., Kohn, R., Giordani, P. and Mun, X. (2008), 'A Copula Based Approach to Adaptive Sampling', *Working paper.* .

Silva, R., Kohn, R., Giordani, P. and Pitt, M. K. (2009), 'Particle Filtering Within Adaptive Metropolis-Hastings Sampling', *Mimeo.* .

Sims, C. A. (1999), Matlab Optimization Software. Code package.

Sims, C. A. (2000), Solving linear rational expectations models. Mimeo, Princeton University.

Smets, F. and Wouters, R. (2003), 'An Estimated Stochastic Dynamic General Equilibrium Model of the Euro Area', *Journal of the European Economic Association* **1(5)**, 1123–1175.

Strid, I. (2010), 'Efficient Parallelisation of Metropolis-Hastings Algorithms Using a Prefetching Approach', *Computational Statistics and Data Analysis* **54(11)**, 2814–2835.

Strid, I. and Walentin, K. (2008), 'Block Kalman Filters for Large-scale DSGE models', *Computational Economics* **33(3)**, 277–304.

Whiley, M. and Wilson, S. P. (2004), 'Parallel Algorithms for Markov Chain Monte Carlo in Latent Spatial Gaussian Models', *Statistics and Computing* **14(3)**, 171–179.

Wilkinson, D. (2006), Parallel Bayesian Computation, *in* E. J. Kontoghiorghes, ed., 'Handbook of Parallel Computing and Statistics', Chapman and Hall, chapter 16, pp. 477–508.

Appendix A: k-harmonic means clustering

A very brief summary of the clustering algorithm is provided here, following Kohn and Giordani (2008). The mixture of normal parameters are estimated using k-harmonic means clustering. Let $\theta_0^j, \ldots, \theta_{\tilde{R}}^j$ where $\tilde{R}$ is the number of draws obtained from the posterior distribution at the point of (re-)estimation, and $j = 1, \ldots, n_\theta$. The number of clusters is denoted n.

1. The procedure below is performed for each parameter, $j = 1, \ldots, n_\theta$.

2. The algorithm is initialised with the components centers $c_1, \ldots, c_n$, where the starting values are chosen using the procedure of Bradley and Fayyad (1998), while using the harmonic k-means instead of the k-means algorithm in the initialisation.

3. For each data point θ_r^j, $r = 0, \ldots, \tilde{R}$, a weight function, $w(\theta_r^j)$, and a membership function, $m(c_i|\theta_r^j)$, is computed

$$w(\theta_r^j) = \frac{\sum_{i=1}^{n} \left\|\theta_r^j - c_i\right\|^{-p-2}}{\left(\sum_{i=1}^{n} \left\|\theta_r^j - c_i\right\|^{-p}\right)^2} \text{ and } m(c_i|\theta_r^j) = \frac{\left\|\theta_r^j - c_i\right\|^{-p-n}}{\sum_{i=1}^{n} \left\|\theta_r^j - c_i\right\|^{-p-n}},$$

   $i = 1, \ldots, n$, $r = 0, \ldots, \tilde{R}$, where $\left\|\theta_r^j - c_i\right\|$ is the Euclidean distance.

4. Each center is updated as

$$c_i = \frac{\sum_{r=0}^{\tilde{R}} m(c_i|\theta_r^j) w(\theta_r^j) \theta_r^j}{\sum_{r=0}^{\tilde{R}} m(c_i|\theta_r^j) w(\theta_r^j)}, \; i = 1, \ldots, n.$$

5. This is repeated until convergence. The cluster centers, c_i, $i = 1, \ldots, n$, are the estimates of the component means, m_j^i, i.e. $\tilde{m}_j^i = c_i$. The estimates of the mixture variances and probabilities are then given by

$$\tilde{v}_j^i = \frac{\sum_{r=0}^{\tilde{R}} m(c_i|\theta_r^j) w(\theta_r^j) (\theta_r^j - c_i)^2}{\sum_{r=0}^{\tilde{R}} m(c_i|\theta_r^j) w(\theta_r^j)}, \; \tilde{p}_j^i \propto \sum_{r=0}^{\tilde{R}} m(c_i|\theta_r^j) w(\theta_r^j), \; i = 1, \ldots, n,$$

   and the estimated parameters are collected in $\tilde{\lambda}_j = (\tilde{m}_j^T \; \tilde{v}_j^T \; \tilde{p}_j^T)^T$, where e.g. $\tilde{m}_j^T = (\tilde{m}_j^1, \ldots, \tilde{m}_j^p)$.

6. The number of clusters, $n = n^*$, is chosen using the Bayesian Information Criterion (BIC), where $n^* \leq n_{\max} = 4$.

After obtaining the clustering estimates, $\tilde{\lambda}_j$, for the chosen number of components, n^*, the likelihood function of the mixture of normal density is maximised, while restricting the parameters to some, fairly narrow, interval around the clustering estimates. The maximum number of function evaluations in optimisation is restricted to $1,000$.

Appendix B: Parallel adaptive and hybrid Metropolis-Hastings sampler

In this appendix pseudo-code for a parallel adaptive and hybrid Metropolis-Hastings sampler is presented. Let

$$q_j\left(\theta|\theta^i\right) = q_j\left(\theta\right),\ j = 1, \ldots, M$$

be a sequence of independence proposal densities and let

$$A = \left\{i_0^A, i_1^A, i_2^A, \ldots, i_M^A\right\} \tag{5.16}$$

denote the adaptation points at which these proposal densities are constructed, where $i_0^A = 0$ and where M is the number of adaptation points. Let P denote the number of processes/processors and let R be the desired total number of draws from the sampler. The number of synchronisation points is K.

Algorithm 17 *Parallel adaptive and hybrid Metropolis-Hastings sampler*

1. The current state of the chain is θ_r and the draw counter has the value $D = r$ when the $(r+1)^{th}$ iteration begins and $i_j^A > r \geq i_{j-1}^A$.

2. (Random walk step) Perform a serial RWM step, producing 1 draw, or a parallel 'always reject' RWM prefetching step, producing D_{pf} draws, where $P \geq D_{pf} \geq 1$.

3. (Independence step) Each process p, $p = 1, \ldots, P$, generates $\theta_p = \left\{\theta_{1p}, \theta_{2p}, \ldots, \theta_{R_p p}\right\}$ where
$$\theta_{ip} \sim q_j,\ i = 1, \ldots, R_p,$$
and collects the values of the posterior evaluated at these parameters in the vector $p_p = \left\{p\left(\theta_{1p}\right), p\left(\theta_{2p}\right), \ldots, p\left(\theta_{R_p p}\right)\right\}$ (parallel).

4. The master process gathers θ_p and p_p, $p = 1, \ldots, P$ (*gather*).

5. (Metropolis-Hastings step) Run the Metropolis-Hastings algorithm with the posterior already evaluated at $\Sigma_{p=1}^P R_p$ parameter values (serial).

6. Step 1 to 4 produced $D_{pf}+\ \Sigma_{p=1}^{P}R_p$ draws. Update the draw counter $D = D + D_{pf}+\ \Sigma_{p=1}^{P}R_p$.

7. If $D \geq i_j^A$ perform adaptation of the independence proposal density f and let $j = j + 1$. Otherwise return to 1. (serial or parallel)

8. Return to 1. Stop when $D \geq R$. ⊡

In step 2, the choice between using a RWM step and a prefetching step is essentially determined by the properties of the available hardware. Prefetching is preferable on theoretical grounds but in practise its performance is more sensitive to the processor-network balance of the parallel computer.

If adaptation is much less frequent than the (prefetching) RWM step, i.e. $K >> M$ which is expected with batch adaptation samplers, there is a better way to implement the algorithm.

Algorithm 18 *Parallel adaptive and hybrid Metropolis-Hastings sampler (the case when $K >> M$)*

1. The current state of the chain is θ_r and the draw counter has the value $D = r$ when the $(r + 1)^{th}$ iteration begins and $r = i_{j-1}^A$.

2. (Independence step) Each process p, $p = 1, \ldots, P$, generates $\theta_p = \left\{\theta_{1p}, \theta_{2p}, \ldots, \theta_{R_p p}\right\}$ where
$$\theta_{ip} \sim f_j, i = 1, \ldots, R_p,$$
and collects the values of the posterior evaluated at these parameters in the vector $p_p = \left\{p\left(\theta_{1p}\right), p\left(\theta_{2p}\right), \ldots, p\left(\theta_{R_p p}\right)\right\}$ (parallel). Here R_p is chosen such that
$$\Sigma_{p=1}^{P}R_p = i_j^A - i_{j-1}^A.$$
(parallel)

3. The master process gathers θ_p and p_p, $p = 1, \ldots, P$ (*gather*).

4. (Metropolis-Hastings step) Run the Metropolis-Hastings algorithm with the posterior already evaluated at $\Sigma_{p=1}^{P}R_p$ parameter values and insert (prefetching) RWM steps as desired (serial).

5. Step 1 to 4 produced $\tilde{D}_{pf}+\ \Sigma_{p=1}^{P}R_p$ draws where $\tilde{D}_{pf}$ is the number of draws produced by the (prefetching) RWM steps in 4. Update the draw counter $D = D + \tilde{D}_{pf}+\ \Sigma_{p=1}^{P}R_p$.

6. Perform adaptation of the independence proposal density f and let $j = j + 1$ (serial or parallel).

7. Return to 1. Stop when $D \geq R$. ⊡

We note that the algorithms, as stated here, imply that the adaptation points in [5.16] are stochastic, due to the prefetching RWM step. It is possible, at the expense of more involved notation, to write the algorithms with deterministic adaptation points. However, in practise this does not affect the performance of the algorithm.

Now, let $1 - \alpha_1$ denote the fraction of draws generated by the independence proposal and assume that the parallel computer is homogeneous, such that $R_p = \tilde{R}$ is the same for all processors, and $\Sigma_{p=1}^{P} R_p = P\tilde{R}$. Under the assumption that the posterior evaluation accounts for the dominant fraction of computational time *and* abstracting from communication and adaptation costs the speedup of the hybrid algorithm is

$$S_h(P, \alpha_1) = \frac{1}{\frac{1-\alpha_1}{P} + \frac{\alpha_1}{S_{pf}(P)}}, \tag{5.17}$$

and

$$1 = S_{rwm}(P) \leq S_{pf}(P) \leq S_h(P, \alpha_1) \leq S_i(P) = P,$$

where $S_{pf}(P) = S_h(P, 1)$ is the speedup of the always reject RWM prefetching algorithm and $S_i(P) = S_h(P, 0) = P$ is the speedup of the pure (adaptive) independence sampler. The speedup of a hybrid algorithm with a RWM step (instead of a prefetching RWM step) is obtained by replacing $S_{pf}(P)$ by $S_{rwm}(P) = 1$ in [5.17]. In figure 5.2 in the main text the relative efficiency contours

$$RE_h(P, \alpha_1) = \frac{S_h(P, \alpha_1)}{P} = \frac{1}{1 - \alpha_1 + \frac{P\alpha_1}{S_{pf}(P)}},$$

are displayed. The speedup $S_{pf}(P)$ used in constructing the figure is the optimal speedup of the static prefetching algorithm, under the assumption that the posterior is multivariate normal. Our focus on the always reject variant of prefetching RWM in this paper follows from the idea that the primary role of the RWM steps in the hybrid algorithm is to facilitate the escape from difficult points in the parameter space. For more details on the prefetching RWM approach, see Strid (2010). Note that the optimal acceptance rate of the static prefetching algorithm implies the always reject tour, i.e. it is optimal to allocate processors according to [5.14].

The Stockholm School of Economics

A complete publication list can be found at *www.hhs.se/research/publications.*
Books and dissertations are Published in the language indicated by the title and can be ordered via e-mail: efi.publications@hhs.se

A selection of recent publications

Books

Barinaga, Ester. *Powerful dichotomies.*
Ericsson, Daniel. *Den odöda musiken.*
Ericsson, Daniel. *Scripting Creativity.*
Melén, Sara. *Globala från start. Småföretag med världen som marknad.* Forskning i Fickformat.
Mårtensson, Pär, Mähring, Magnus. *Mönster som ger avtryck: Perspektiv på verksamhetsutveckling.*
Sjöström, Emma. *Ansiktslösa men ansvarsfulla.* Forskning i fickformat.
Wijkström, Filip. *Civilsamhällets många ansikten.*

Dissertations

Alexandersson, Gunnar. *The Accidental Deregulation. Essays on Reforms in the Swedish Bus and Railway Industries.*
Ejenäs, Markus. *Ledning av kunskapsintegration - förutsättningar och hinder : en studie av en fusion mellan IT- och managementkonsulter.*
Engvall, Anders. *Poverty and Conflict in Southeast Asia.*
Juks, Reimo. *Corporate Governance and the Firm's Behaviour towards Stakeholders.*
Lychnell, Lars-Olof . *IT- relaterad verksamhetsförändring. Processer som formar växelspelat mellan utveckling och användning.*
Magnusson Bernard, Kristin. *Remittances, Regions and Risk Sharing.*
Mohlin, Erik. *Essays on Belief Formation and Pro-Sociality.*
Nakatani, Tomoaki. *Four essays on building Conditional Correlation GARCH models.*
Siming, Linus. *Private Equity and Advisors in Mergers and Acquisitions.*
Sjöqvist Rafiqui, Pernilla. *Evolving Economic Landscapes: Institutions and localized economies in time and space.*
Sunesson, Daniel. *School networks and active investors.*
Öhman, Niclas. *Considering intentions.*