Theoretical Computer Science 814 (2020) 234–248

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Paraunitary matrices, entropy, algebraic condition number and Fourier computation

Nir Ailon [1]

Technion Israel Institute of Technology, Israel

ARTICLE INFO

Article history:
Received 8 November 2018
Received in revised form 9 September 2019
Accepted 1 February 2020
Available online 4 February 2020
Communicated by P. Lu

Keywords:
Fourier transform
Lower bounds
Complexity
Linear algebraic computation

ABSTRACT

The Fourier Transform is one of the most important linear transformations used in science and engineering. Cooley and Tukey's Fast Fourier Transform (FFT) from 1964 is a method for computing this transformation in time $O(n \log n)$. From a lower bound perspective, relatively little is known. Ailon shows in 2013 an $\Omega(n \log n)$ bound for computing the normalized Fourier Transform assuming only unitary operations on two coordinates are allowed at each step, and no extra memory is allowed. In 2014, Ailon then improved the result to show that, in a κ-well conditioned computation, Fourier computation can be sped up by no more than $O(\kappa)$. The main conjecture is that Ailon's result can be exponentially improved, in the sense that κ-well condition cannot admit $\omega(\log \kappa)$ speedup.
The main result here is that 'algebraic' κ-well condition cannot admit $\omega(\sqrt{\kappa})$ speedup. One equivalent definition of algebraic condition number is related to the degree of polynomials naturally arising as the computation evolves. Using the maximum modulus theorem from complex analysis, we show that algebraic condition number upper bounds standard condition number, and equals it in certain cases. Algebraic condition number is an interesting measure of numerical computation stability in its own right, and provides a novel computational lens. Moreover, based on evidence from other recent related work, we believe that the approach of algebraic condition number has a good chance of establishing an algebraic version of the main conjecture.

1. Introduction

The (discrete) normalized Fourier transform is a complex linear mapping sending an input $x \in \mathbb{C}^n$ to $y = Fx \in \mathbb{C}^n$, where F is an $n \times n$ unitary matrix defined by

$$F(k, \ell) = n^{-1/2} e^{-i2\pi k\ell/n} . \tag{1}$$

The Fast Fourier Transform (FFT) of [6] is a method for computing the Fourier transform of a vector $x \in \mathbb{C}^n$ in time $O(n \log n)$ using a so called linear-algebraic algorithm. A linear-algebraic algorithm's state at each point in the computation is a vector in $\mathbb{C}^n$, representing a linear transformation of the input. Each coordinate of the vector is a complex number stored in a memory location (or rather, two real numbers, one representing the real part and the other the imaginary part). The next state is obtained from the current one by a simple linear algebraic transformation. Before continuing, we just note here that

E-mail address: nailon@cs.technion.ac.il.
[1] This work was supported by a European Research Commission (ERC) grant GA682203 (SpeedInfTradeoff).

https://doi.org/10.1016/j.tcs.2020.02.002

the matrix F is unitary, which means that it preserves Euclidean norms, and equivalently, it has condition number of 1. The notion of condition number will be elaborated upon in what follow.

As for lower bounds, it is trivial that computing the Fourier Transform requires a linear number of steps, because each coordinate in the output depends on each coordinate in the input. There has not been much prior work on better bounds, for Fourier transform over the real or complex field, with respect to a reasonable model of computation.

In 1973, Morgenstern proved that if the modulus of constants used in an unnormalized Fourier transform algorithm are bounded by a global constant, then the number of steps required is at least $\Omega(n \log n)$. The unnormalized Fourier transform is defined as $\sqrt{n}$ times the Fourier transform, as defined above in (1). He used a potential function related to matrix determinant. The normalization issue may seem pedantic, however, a close inspection reveals that Morgenstern's technique does not provide any meaningful lower bound for the normalized Fourier transform.

Pan [13] shows an $\Omega(n \log n)$ bound for so-called *synchronized* algorithms which are, roughly speaking, based on a layered computational graph (as in the well-known FFT). It is not clear why synchronicity is required for optimality.

Papadimitriou [14] proves that the graph structure of Cooley and Tukey's FFT algorithm is optimal if we are only allowed to multiply by n'th roots of unity constants, and the transformation is computed in a finite field. It should be noted that removing the finite field requirement from Papadimitriou's paper results in a lower bound that is subsumed by that of Morgenstern's, and in particular, applies only to the unnormalized FT. Again, it is not clear how to obtain something for a normalized FFT from this result.

Winograd [16] was able to reduce by 20% the number of multiplications in Cooley and Tukey's FFT, without changing the number of additions. The same author [17] shows that at least linearly many multiplications are needed for performing Cooley and Tukey's algorithm for computing the *unnormalized* FFT, defined as $\sqrt{n}$ times (1).

We also note that in the quantum world, an $O(\log^2 n)$-time algorithm is known using quantum gates, each unitary acting on only 4 entries. Refer to [9] and to references within for this line of work, which is not further discussed here.

Ailon [1] considered a computational model allowing only 2×2 rotation gates acting on a pair of coordinates. He showed an $\Omega(n \log n)$ lower bound on the number of such gates required for the normalized Fourier transform. The proof was done by defining a potential function on the matrices $M^{(t)}$ defined as the transformation that takes the input to the machine state in step t. The potential function is simply the sum of Shannon entropy of the probability distributions defined by the squared modulus of elements in the matrix rows. (Due to orthogonality, each row, in fact, defines a probability distribution.) That work raised the question of *why shouldn't it be possible to gain speed by 'escaping' the unitary group?* Later ([3]) he added nonzero, constant multiplication gates acting on a single coordinate to the model. By generalizing the entropy potential function to the *quasi-entropy* function, so called because it is applied to possibly negative numbers, the approach allows proving that a speedup in computing *any* scaling of FFT *requires* either working with very large or very small numbers (asymptotically growing accuracy). The bounds are expressed using a quantitative relationship between time (number of gates) versus the standard notion of condition number of matrices. More precisely, it is shown that speedup of FFT by factor of b implies that at least one $M^{(t)}$ must be $\Omega(b)$-ill-conditioned. In [2], he further showed that speedup implies ill-condition at *many* points in the computation, affecting independent information in the input. The results of Ailon [1–3] are believed not to be tight. The main open question, informally stated, is as follows. We will restate it as Conjecture 14 in a precise way in Section 6, after building our framework.

Conjecture 1. *[Informal] Any speedup of FFT (counting linear algebraic operations), if existed, would require 'exponential (in the speedup) ill-condition'.*

Formal versions of Conjecture 1 are given in Conjectures 14 and 15 below.

1.1. Our contribution

This work defines a new and natural *algebraic* notion of matrix condition number, which upper bounds the usual definition of condition number, and equals it in some cases. Using this new notion, we are able to show that κ-well conditioned computation allows only $O(\sqrt{\kappa})$ speedup of FFT, in other words, a quadratic improvement on the previous result, albeit with a weaker notion of condition. More importantly, we believe that an algebraic version of Conjecture 1 is achievable with respect to algebraic condition, and justify this belief in Section 6.

1.2. The RAM computational model and this work

Before we continue, we shall emphasize a point about a popular computational model: *RAM*. Roughly speaking, in this model, a computer word contains $\Theta(\log n)$ bits of precision, for input of size n, and each addition/multiplication is counted as 1 operation. While highly suitable for problems of complexity that allows "hiding poly-log factors", this model should be used with caution for lower bounding FT computation. One of the main arguments often stated against Conjecture 1 is that poly(n) ill-condition, in the extreme case of $\Theta(\log n)$ speedup of FFT, roughly means $\Theta(\log n)$ bit words, which is assumed by the RAM model anyway, so why is the conjecture interesting!? But why do we assume the RAM model? It is tempting to think that we must assume it, due to the following example: If we were to allow the more realistic $O(1)$ bits per word, and compute the normalized Fourier transform of $(1, 1, 1, \ldots, 1)$, then the result would have a coordinate equaling $\sqrt{n}$ and

hence would overflow. But notice that all the other coordinates of the output equal 0. Requiring $\Theta(\log n)$ bits per word in all words is an overkill that merely satisfies an instinct to work with equal, fixed sized words.

For readers who insist on a RAM-like model: Our line of work studies the complexity of computing the FT for inputs in n dimensional real space, containing $O(1)$ bits per word *on average* (over the coordinates). Such representation allows describing a point inside the Euclidean ball of radius $\sqrt{n}$, to within fixed accuracy per coordinate. The size of the word can vary with the computation, according to the number stored in the word. Additionally, the cost of performing each operation grows with the number of bits involved in the operands. The main argument is that speedup of FFT requires a growing (in the speedup) number of bits in the words participating in the algorithm steps, and to quantify this growth.

Alternatively, the reader is invited to think about well-conditioned linear computation as a computational model worthy of discussion in its own right, and will hopefully be convinced that it is more suitable to think about FT complexity in such a model, rather than under a RAM or a RAM*ish* lens.

2. Fourier transform types

In theory and practice of engineering and computer science, there are two main types of Fourier transforms considered. The Discrete Fourier Transform (DFT) in n dimensions, as defined in (1), is a complex unitary mapping defined by the characters of the Abelian group $\mathbb{Z}/n\mathbb{Z}$. The Walsh-Hadamard transform is a real orthogonal mapping defined by the characters of the n dimensional binary hypercube $(\mathbb{Z}/2\mathbb{Z})^n$. More precisely, for n an integer power of 2, the (i,j)'th matrix element is defined as $\frac{1}{\sqrt{n}}(-1)^{\langle [i-1],[j-1]\rangle}$, where $\langle\cdot,\cdot\rangle$ is dot-product, and $[p]$ denotes the bit representation of the integer $p \in \{0,\ldots,n-1\}$ as a vector of $\log n$ bits. Similarly to FFT for DFT, there is an $O(n\log n)$ algorithm for computing the Walsh-Hadamard transform of an input x.

Throughout, we will assume n is an integer power of 2 and will use F to denote either the $n-$Walsh-Hadamard, which is a real orthogonal transformation, or the real representation of the $(n/2)$-DFT, which is a complex unitary transformation. By *real representation*, we mean the standard view of a complex number $a+\iota b$ as a column vector $(a,b)^T$ in two dimensional real space, and the multiplicative action $a+\iota b \mapsto (c+\iota d)(a+\iota b)$ given by left-multiplication by the matrix $\begin{pmatrix} c & -d \\ d & c \end{pmatrix}$. In that way, a vector in n dimensional complex space embeds to a $2n$ dimensional real space, and a $n\times m$ complex matrix embeds as a $(2n)\times(2m)$ real matrix. This allows us to avoid defining a computational model over the complex field, because such computation can be emulated by reals anyway. Nevertheless, our analysis will require extension to the complex field.

3. Computational model and notation

We remind the reader of the computational model discussed by [1,3], which is a special case of the linear computational model. The machine state represents a vector in $\mathbb{R}^\ell$ for some $\ell \geq n$, where initially it equals the input $x \in \mathbb{R}^n$ (with possible padding by zeroes, in case $\ell > n$). Each step (gate) is either a *rotation* or a *constant*.

A rotation applies a 2×2 orthogonal operator mapping a pair of machine state coordinates (rewriting the result of the mapping to the two coordinates). Note that a rotation includes the 2×2 mapping affecting a single coordinate, e.g. $\begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix}$. (Technically this is a reflection, but we refer to it as a *rotation gate* here.)

A constant gate multiplies a single machine state coordinate (rewriting the result) by a positive real constant. In case $\ell = n$ we say that we are in the in-place model. Any nonsingular linear mapping over $\mathbb{R}^n$ can be decomposed into a sequence of rotation and constant gates in the in-place model, and hence our model is universal. Normalized FFT works in the in-place model, using rotations only. A restricted method for dealing with $\ell > n$ was developed by [3]. We focus in this work on the in-place model only.

Since both rotations and constants apply a linear transformation on the machine state, their composition is a linear transformation. If $\mathcal{A}_n$ is an in-place algorithm for computing a linear mapping over $\mathbb{R}^n$, it is convenient to write it as $\mathcal{A}_n = (M^{(0)} = \mathrm{Id}, M^{(1)}, \ldots, M^{(m)})$ where m is the number of steps, $M^{(t)} \in \mathbb{R}^{n\times n}$ is the mapping that satisfies that for input $x \in \mathbb{R}^n$ (the initial machine state), $M^{(t)}x$ is the machine state after t steps (Id is the identity matrix). The matrix $M^{(m)}$ is the target transformation, which will typically be F in our setting. For $t \in [m] := \{1,2,\ldots,m\}$, if the t'th gate is a rotation, then $M^{(t)}$ differs from $M^{(t-1)}$ in at most two rows, and if the t'th gate is a constant, then $M^{(t)}$ defers from $M^{(t-1)}$ in at most one row.

Throughout, for a matrix M, we let $M_{i,:}$ denote the i'th row of M. The symbol ι denotes $\sqrt{-1}$. For an integer a, notation $[a]$ is shorthand for the set $\{1,\ldots,a\}$. All logarithms are assumed to be base 2, unless the base is explicit, as in $\log_5 6$. For a matrix M over the field of real or complex numbers, $\|M\|$ is the spectral norm.

3.1. A note on row-addition matrices

A row addition operation is defined by adding two input coordinates, and saving the result in-place in one of the two. In matrix form, this is represented by $\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$. The row addition operation is a natural one, and hence it is worth asking whether our computational model subsumes it. Notice that by the SVD theorem, this 2-by-2 matrix can be decomposed

as a product of 3 matrices, the first and the third being rotation matrices, and the middle being a diagonal matrix of the corresponding singular values, which are $O(1)$. The diagonal matrix itself can hence be emulated by two *constant* gates. Therefore, up to a factor of 4 in accounting for number of operations, this operation is subsumed here.

4. Replacing constant gates with an indeterminate

Fix an in-place algorithm $\mathcal{A} = (M^{(0)} = \mathrm{Id}, M^{(1)}, \ldots, M^{(m)} = F)$ computing F. Let $\mathcal{C}_{\mathcal{A}}$ denote the set of values used in the constant gates. Let $0 < \Delta < 1$ be some real number. If each $c \in \mathcal{C}_{\mathcal{A}}$ is an integral power of Δ, then we say that $\mathcal{A}$ is integral with respect to Δ. We will assume that $\Delta \geq 2/3$ (otherwise we could replace Δ with $\Delta^{1/\ell}$ for a suitable integer ℓ, with respect to which $\mathcal{A}$ is still integral).

Assume that $\mathcal{A}$ is integral with respect to some $2/3 \leq \Delta < 1$. The integrality assumption will be removed in Section Appendix A using limit arguments. We define a new algorithm $\mathcal{A}_\Delta$ with corresponding matrices $(M_\Delta^{(0)}, \ldots, M_\Delta^{(m)})$ where each $M_\Delta^{(t)}$ is now an $n \times n$ matrix over the ring of Laurent polynomials with complex coefficients in an indeterminate z.[2] (Throughout, by *polynomials* we refer to Laurent polynomials, and by *proper polynomials* we mean that only nonnegative power monomials are allowed.) In what follows, we may interchangeably write p or $p[z]$ if it is clear from the context that p is a (Laurent) polynomial in z.

To get accustomed to our notation, note that for some matrix $A = A[z]$ in this ring we can write e.g. $A_{1,3}[7]$, referring to the (complex) number in the first row, third column of the matrix obtained by assigning the value 7 to z in $M_\Delta^{(t)}$. We now inductively define $M_\Delta^{(t)}$. The first matrix $M_\Delta^{(0)}$ is simply $M^{(0)} = \mathrm{Id}$. Having defined $M_\Delta^{(t-1)}$, we define $M_\Delta^{(t)}$ depending on whether the t'th gate is a rotation, or a constant gate. In case of rotation, the matrix $M_\Delta^{(t)}$ is obtained from $M_\Delta^{(t-1)}$ by performing the same transformation giving $M^{(t)}$ from $M^{(t-1)}$. More precisely, if $M^{(t)}$ is obtained from $M^{(t-1)}$ by a rotation matrix R, namely $M^{(t)} = RM^{(t-1)}$, then $M_\Delta^{(t)} = RM_\Delta^{(t-1)}$.

In case of a constant gate with value $0 < c \in \mathcal{C}_{\mathcal{A}}$, we replace the value c with the monomial $z^{\log_\Delta c}$. More precisely, if $M^{(t)}$ is obtained from $M^{(t-1)}$ by left-multiplication with the matrix $\mathrm{diag}(1, \ldots, 1, c, 1, \ldots, 1)$, then

$$M_\Delta^{(t)} = \mathrm{diag}(1, \ldots, 1, z^{\log_\Delta c}, 1, \ldots, 1) M_\Delta^{(t-1)} .$$

By the integrality of $\mathcal{A}$ with respect to Δ, the number $\log_\Delta c$ is indeed an integer. From the definition of $M_\Delta^{(t)}[z]$, a simple induction implies that for all $t \in \{0 \ldots m\}$, $M_\Delta^{(t)}[\Delta] = M^{(t)}$. In words, this means that the t'th matrix in the original algorithm $\mathcal{A}$ is obtained by polynomial evaluation at Δ.

Throughout, for a polynomial $p = p[z]$, we let $\mathrm{coeff}_i(p)$ denote the coefficient (matrix, vector or scalar) of p corresponding to z^i. In particular, $p = \sum_i \mathrm{coeff}_i(p) z^i$.[3]

4.1. Paraunitary matrices and algebraic condition number

Here and throughout, for a complex number u, $\bar{u}$ is its complex conjugate. For a complex polynomial $p = p[z]$, complex conjugation $\bar{p}$ is the polynomial obtained by complex-conjugating the coefficients of p and substituting z^{-1} for z. For a polynomial matrix $M = M[z]$, the matrix M^* is obtained by transposing M and applying *polynomial* complex conjugation (as just defined) to all its elements. For a nonzero polynomial $p[z]$, let $\deg(p)$ denote the maximal degree of z (with a nonzero coefficient), and let $\mathrm{val}(p)$ denote the minimal degree. (The operator name val is short for *valuation*, a standard algebraic abstraction for which "min degree" is the most common special case.) For example, if $p[z] = -6z^{-2} + 3 + z^5$, then $\deg(p) = 5, \mathrm{val}(p) = -2$. Note that $\deg(p)$ may be negative, and $\mathrm{val}(p)$ may be positive. For a polynomial matrix $M[z]$, $\deg(M) = \max_{i,j} \deg(M_{i,j})$ and $\mathrm{val}(M) = \min_{i,j} \mathrm{val}(M_{i,j})$.

Definition 2. A complex polynomial matrix $U[z]$ is *paraunitary* if $U^*U = \mathrm{Id}$.

Paraunitary matrices are used in signal processing (see e.g. [15], chapter 14), and have some useful properties for our purposes. First, they form a multiplicative group. This fact is useful for verifying the following:

Claim 3. *The matrices $M_\Delta^{(t)}$ are paraunitary for all t.*

The following is easy to see from the definitions, using simple induction:

Claim 4. *For paraunitary $U[z]$, the evaluation $U[\omega]$ is a (complex) unitary matrix for any ω a complex number on the complex unit circle.*

[2] We remind the reader that a Laurent polynomial possibly has negative degree monomials.

[3] We avoid subscripting for denoting polynomial coefficients, because that would be confused with matrix or vector indexing.

This will be used below. We now remind the reader of the definition of a κ-well conditioned algorithm, as defined in [2,3]. In this work, we will refer to this standard notion of well conditionedness *geometric*, because it is related to geometric stretching and shrinking of vectors under linear operators.

Definition 5. The (geometric) condition number of a nonsingular complex matrix M is $\|M\| \cdot \|M^{-1}\|$. A nonsingular matrix M is (geometrically) κ-well conditioned if its (geometric) condition number is at most κ. Otherwise it is (geometrically) κ-ill conditioned. The algorithm $\mathcal{A} = (\mathrm{Id}, M^{(1)} \dots M^{(m)})$ is (geometrically) κ-well conditioned if $M^{(t)}$ is (geometrically) well conditioned for all $t = 0 \dots m$. Otherwise, it is (geometrically) κ-ill conditioned. The (geometric) condition number of $\mathcal{A}$ is the smallest κ such that $\mathcal{A}$ is (geometrically) κ-well conditioned.

We define a different, though not unrelated (as we shall see below) notion of condition number of an algorithm. We first define it for Δ-integral algorithms, and then extend to general algorithms.

Definition 6. A Δ-integral algorithm $\mathcal{A}$ is Δ-*algebraically* κ-well conditioned if for all $t = 1 \dots m$, $\Delta^{\mathrm{val}(M_\Delta^{(t)}) - \deg(M_\Delta^{(t)})} \leq \kappa$. Otherwise, it is Δ-algebraically κ-ill conditioned.

Although the definition seems to depend on it, the parameter Δ is immaterial for defining (and thinking about) algebraic condition number. It is trivial to see that if $\mathcal{A}$ is integral with respect to both Δ and Δ' and is Δ-algebraically κ-well conditioned, then it is also Δ'-algebraically κ-well conditioned. In fact, even if an algorithm is integral with respect to *no* Δ, we can extend the definition by using limits. Being algebraically κ-well conditioned is henceforth a property of any algorithm $\mathcal{A}$, regardless of integrality. The precise definitions are given in Section Appendix A, for completeness. We henceforth define:

Definition 7. The algebraic condition number of $\mathcal{A}$ is the smallest κ such that $\mathcal{A}$ is algebraically κ-well conditioned.

If we consider some intermediate matrix $M^{(t)}$ given rise to by our algorithm $\mathcal{A}$, then algebraic well conditioning cannot be determined from $M^{(t)}$ 'in vitro', because it depends on *how* we reached $M^{(t)}$ from the initial state Id. This is in contrast with the notion of (geometric) condition, which can be determined from a single matrix. Nevertheless, the following lemma tells us that the two notions are quantitatively related.

Lemma 8. *If an algorithm $\mathcal{A}$ is algebraically κ-well conditioned, then it is geometrically κ-well conditioned. There exist algorithms for which the two notions of condition number coincide.*

The proof of this connection requires some complex analysis, and in particular the maximum modulus theorem. We provide it here only for the case $\mathcal{A}$ is integral with respect to some Δ. The extension to general $\mathcal{A}$ is not difficult using limit arguments, and we present it in Section A.2.

Proof. Assume $\mathcal{A}$ is integral with respect to $\frac{2}{3} \leq \Delta < 1$, and fix $t \in [m]$. Let $v = \mathrm{val}(M_\Delta^{(t)})$ and $d = \deg(M_\Delta^{(t)})$. By definition of val and deg, the polynomial matrix $M^v[z] := z^{-v} M_\Delta^{(t)}$ is a proper polynomial matrix (with no negative degree monomials). Therefore, it is an entire (matrix valued) function of z, in the complex sense. By the (weak) maximum modulus theorem for Banach space valued functions on the complex plane ([7], page 230), we conclude that $\|M^v[\Delta]\| \leq \max_{\omega \in \mathbb{C}:|\omega|=1} \left\|M^v[\omega]\right\|$, because Δ is contained in the open unit disk. The right hand side is identically 1 by Claims 3 and 4. The left hand side equals exactly $\Delta^{-v}\|M_\Delta^{(t)}[\Delta]\| = \Delta^{-v}\|M^{(t)}\|$. Therefore,

$$\|M^{(t)}\| \leq \Delta^v \,. \tag{2}$$

By Claim 3 and Definition 2, the inverse of the matrix $M_\Delta^{(t)}$ is $(M_\Delta^{(t)})^*$.[4] Let

$$M^d := z^d (M_\Delta^{(t)})^* \,.$$

By construction, the polynomial matrix M^d is a proper polynomial, and hence again by the (weak) maximum modulus theorem, $\Delta^d \|(M_\Delta^{(t)})^*[\Delta]\| = \|M^d[\Delta]\| \leq \max_{\omega \in \mathbb{C}:|\omega|=1} \left\|M^d[\omega]\right\| = 1$. But by paraunitarity $(M_\Delta^{(t)})^*$ is the inverse of $M_\Delta^{(t)}$, and this holds in particular for $z = \Delta$. Hence the last inequality gives $\Delta^d \|(M^{(t)})^{-1}\| \leq 1$ or:

$$\|(M^{(t)})^{-1}\| \leq \Delta^{-d} \,. \tag{3}$$

Combining (2) with (3), we get $\|(M^{(t)})^{-1}\| \cdot \|M^{(t)}\| \leq \Delta^v \cdot \Delta^{-d} \leq \kappa$, where the rightmost inequality is by the lemma's assumption. This concludes the first part of the result.

[4] The matrix $M_\Delta^{(t)}$ is real, and hence its transpose equals its conjugate-transpose. We prefer to keep the complex notation $(\cdot)^*$.)

To show tightness, it is enough to work with 2×2 matrices. Consider an algorithm $\mathcal{A}$ performing two steps. The first multiplies the first coordinate by a positive number Δ, and the second multiplies the second coordinate by Δ^{-1}. It is easy to see that both the algebraic and the standard notions of condition number coincide for this case. □

We are now ready to state our main result.

Theorem 9. *Assume algorithm $\mathcal{A} = (M^{(0)}, \ldots, M^{(m)})$ computes F, and is algebraically κ-well conditioned. Then*

$$m = \Omega\left(\frac{n \log n}{\sqrt{\kappa}}\right). \tag{4}$$

The proof is presented in the remainder of the paper, for the case $\mathcal{A}$ is integral with respect to some $\frac{2}{3} \leq \Delta \leq 1$. The general case, using limit arguments, is deferred to Section A.3. But first,

4.2. Algebraic-condition number: a novel computational lens

Geometric condition number is a standard matrix invariant that appears in classic textbooks on matrix theory, and is a useful invariant when analyzing convergence rates of certain numerical algorithms on matrices. It is important to discuss whether the bound of Theorem 9 is interesting under the non-standard algebraic condition number notion. Admittedly, we were not able to prove a version of Theorem 9 with 'geometrically κ-well conditioned' replacing 'algebraically κ-well conditioned'. However, we have good reason to believe that there is a chance of proving an algebraic version of Conjecture 1, in which the denominator of (4) is logarithmic in κ (see Conjecture 14 below for an exact statement). Evidence for this is presented in Section 6. Proof of Conjecture 14 would be an important step toward understanding the complexity of computing the Fourier transform (and perhaps other problems). Additionally, proof of the algebraic conjecture would highlight the importance of its (stronger) geometric counterpart.

One issue is, that it may seem unnatural to introduce polynomials for a problem involving only real arithmetic. But there is a simple alternative way to think about algebraic condition number that does not involve polynomials: If we think of the constant gates as elements in the multiplicative group $\mathbb{R}_{>0}$, and the coefficients of the rotation and reflection operations as elements of the field $\mathbb{R}$, then the machine state at each step t is now a vector with coefficients in the group algebra $\mathbb{R}[\mathbb{R}_{>0}]$. Accordingly, the matrices $M^{(t)}$ are also defined over $\mathbb{R}[\mathbb{R}_{>0}]$. The algebraic condition number of a matrix is now the ratio between the largest group element $c \in \mathbb{R}_{>0}$ with nonzero coefficient in some matrix element, and the smallest one. Intuitively, a large (resp. small) group element $c \in \mathbb{R}_{>0}$ occurs in the computation as a result of a chain of constant gates, the product of which is c, that affect some direction in the input n-dimensional vector space.

Note that numbers that are introduced in the computation from rotations (trigonometric functions of rotation angles), however miniscule, do not affect the algebraic condition number, because they are used as coefficients in the group algebra $\mathbb{R}$, and not as elements in the group $\mathbb{R}_{>0}$. This property is shared with geometric condition number, which is also not affected by rotation action.

4.3. Polynomial matrices: norms and entropy

We will need to define a norm for polynomials, polynomial vectors and polynomial matrices, over the complex field. For a polynomial $p = p[z] \in \mathbb{C}[z]$, we define $\|p\| = \sqrt{\sum_k |\mathrm{coeff}_k(p)|^2}$. The following is well known from harmonic analysis:

$$\|p\| = \sqrt{\int_0^1 \left|p\left[e^{2\pi \imath t}\right]\right|^2 dt}. \tag{5}$$

For a polynomial vector $v = v[z] \in \mathbb{C}[z]^n$ we define

$$\|v\| = \sqrt{\sum_{i=1}^{n} \|v_i\|^2}, \tag{6}$$

where we remind the reader that v_i is the i'th coordinate of v. Finally, for a polynomial matrix A, we define $\|A\|_{2,\infty}$ as shorthand for $\max_i \|A_{i,:}\|$. Let M denote a real paraunitary matrix. Then we extend the definition of the quasi-entropy potential from [2,3] as follows:

$$\Phi(M) = \sum_k \sum_{i,j} h\left(\left|\mathrm{coeff}_k(M_{i,j})\right|^2\right) \tag{7}$$

where $h : \mathbb{C} \mapsto \mathbb{C}$ is defined by $h(u) = -u \log |u|$. (Note: No, this is not a mistake, and we do not want to define h as $-|u| \log |u|$. We will be using h for possibly negative and even complex values of u, where this makes a difference.[5]) For the Fourier transform F,

$$\Phi(F) = \Theta(n \log n) \ . \tag{8}$$

We also define a preconditioned version of Φ for any paraunitary matrix M, parameterized by two polynomial matrices A, B:

$$\Phi_{A,B}(M) = \sum_k \sum_{i,j} h\left(\text{coeff}_k((MA)_{i,j})\overline{\text{coeff}_k((MB)_{i,j})}\right), \tag{9}$$

where the outer sum is over k for which the internal sum is not null. Note that $\Phi_{\text{Id},\text{Id}}(M) = \Phi(M)$. The general idea of a preconditioned quasi-entropy potential was developed in [2]. The idea there (and here) is to carefully choose fixed A and B, and to track the potential along the evolution of M under algorithm steps. The following key lemma bounds the absolute value of the change in entropy of a paraunitary matrix, incurred by multiplication of rows by monomials (corresponding to constant multiplication in the original algorithm) and by planar rotations (corresponding to rotations in the original algorithm).

Lemma 10. *(1) Let $M[z]$ be a paraunitary matrix, and let $A[z]$, $B[z]$ be some complex matrices. Let $M'[z]$ be a paraunitary matrix obtained from $M[z]$ by multiplication by a diagonal matrix with monomials on the diagonal, that is $M'[z] = \text{diag}(z^{i_1}, \ldots, z^{i_n})M[z]$. Then $\Phi_{A,B}(M') = \Phi_{A,B}(M)$. (2) If M' is obtained from M by a planar rotation action, then*

$$\left|\Phi_{A,B}(M') - \Phi_{A,B}(M)\right| = O\left(\|MA\|_{2,\infty} \cdot \|MB\|_{2,\infty}\right) \ .$$

Proof. Part (1) is trivial, from the definitions. For part (2), polynomial matrix potential function is, equivalently, a scalar matrix potential function (defined in [3]) if we replace each polynomial by a row-vector of its coefficients. Therefore, using the potential change bound theorem of [3], we have the statement of the theorem. For self containment, we also provide a full proof in Section Appendix B in the appendix. □

5. The preconditioners

We now define our two polynomial matrix preconditioners, $A[z]$ and $B[z]$, that will be used to define a potential for proving the main theorem. First some extra notation: $\rho := \frac{\log \kappa}{2 \log \Delta^{-1}}$, or $\Delta^{-\rho} = \sqrt{\kappa}$. Using the notation, we get that for all $t = 0, \ldots, m$, $\deg(M_\Delta^{(t)}) - \text{val}(M_\Delta^{(t)}) \le 2\rho$. This implies that either

(i) $\text{val}(M_\Delta^{(t)}) \ge -\rho$, or
(ii) $\deg(M_\Delta^{(t)}) \le \rho$.

We will assume the case (i) (a symmetric construction can be done for the case (ii), and hence there is no loss of generality). We will also make the implicit assumption that $\kappa = n^{o(1)}$, because otherwise the statement of the theorem is obvious from the trivial observation that computation of F requires linearly many steps. For convenience, we will define $\mu = 1 - \Delta$, and let ℓ be the smallest integer such that $\Delta^\ell \le 1/2$. By our assumptions on Δ, this also implies $\Delta^\ell \ge 1/4$, hence $\Delta^\ell = \Theta(1)$. Also note that

$$\ell = \Theta(1/\mu) \ . \tag{10}$$

Let d be $\deg(M_\Delta^{(m)})$. The preconditioner A is defined as

$$A[z] = \text{Id}(1 + \Delta z^{-1} + \Delta^2 z^{-2} + \Delta^3 z^{-3} + \cdots + \Delta^{\rho+d+1+\ell} z^{-\rho-d-1-\ell}) \ .$$

Consider now the polynomial matrix $P[z] = M_\Delta^{(m)}[z] \cdot A[z]$.

Claim 11. *The (matrix) coefficient of $P[z]$ corresponding to $z^{-\rho-i}$ is exactly $F\Delta^{i+\rho}$ for all i in the range $R := \{0, 1, \ldots, \ell\}$.*

Proof. By polynomial multiplication definition, $\text{coeff}_{-\rho-i}(P)$ is

$$\sum_{k=-\rho-d-1-\ell}^{0} \text{coeff}_{-\rho-i-k}(M_\Delta^{(m)})\text{coeff}_k(A) = \sum_{k=-\rho-d-1-\ell}^{0} \Delta^{-k}\text{coeff}_{-\rho-i-k}(M_\Delta^{(m)})$$

[5] This comment will be removed from published versions.

$$= \Delta^{i+\rho} \sum_{k=-\rho-d-1-\ell}^{0} \Delta^{-\rho-i-k} \mathrm{coeff}_{-\rho-i-k}(M_\Delta^{(m)}) = \Delta^{i+\rho} M_\Delta^{(m)}[\Delta] = \Delta^{i+\rho} F \ , \tag{11}$$

as required. □

Remember that for our construction, for all i in the range $[0, \ell]$, $\Delta^{i+\rho} = \Theta(\sqrt{\kappa})$. We now define the other preconditioner, $B[z]$ as $B[z] := (M_\Delta^{(m)})^* \sum_{i=-\rho-\ell}^{-\rho} z^i F$.

We make a short note to demystify and provide intuition for our choice of the two preconditioners. By the definition of B, $M_\Delta^{(m)} B = \sum_{i=-\rho-\ell}^{-\rho} z^i F$. This implies that for all $i = -\rho-\ell, \ldots, -\rho$, the coefficient corresponding to z^i of both $M_\Delta^{(m)} A$ and of $(M_\Delta^{(m)})^* B$ are proportional to F. Thus we have aligned the two operands of the product inside $h(\cdot)$ in (9) so that the potential is high. At the same time, in the beginning of the computation, the potential is low because A has only very few nonzeroes (namely, on the diagonal). Creating this big potential gap, which we quantify in Lemma 13 below, is one of the requirements for our proof to work. The other is upper bounding the potential change at each step, as we now do.

Lemma 12. *For any paraunitary M, $\|MA\|_{2,\infty} \le \Theta(\sqrt{\ell})$ and $\|MB\|_{2,\infty} \le \Theta(\sqrt{\ell})$.*

Proof. The proof relies on the fact that both matrices MA and MB belong to the family $\mathcal{X}$ defined as:

$$\{U[z]p[z]X : \ U \text{ is paraunitary, } p \text{ is a polynomial in } \mathbb{C}[z],\ X \in \mathbb{C}^{n\times n} \text{ is unitary}\} \ .$$

The main technical claim, is to show that for $Y = U[z]p[z]X \in \mathcal{X}$, any row of Y has norm $\|p\|$, as can be seen by the following chain for fixed $i \in [n]$:

$$\begin{aligned}
\|Y_{i,:}\|^2 &= \sum_{j=1}^{n} \|Y_{i,j}\|^2 = \sum_{j=1}^{n} \int_0^1 |Y_{i,j}[e^{2\pi \iota t}]|^2 dt = \sum_{j=1}^{n} \int_0^1 Y_{i,j}[e^{2\pi \iota t}] \overline{Y_{i,j}[e^{2\pi \iota t}]} dt \\
&= \int_0^1 \sum_{j=1}^{n} Y_{i,j}[e^{2\pi \iota t}] \overline{Y_{i,j}[e^{2\pi \iota t}]} dt = \int_0^1 \sum_{j=1}^{n} (U[z]p[z]X)_{i,j}[e^{2\pi \iota t}] \overline{(U[z]p[z]X)_{i,j}[e^{2\pi \iota t}]} dt \\
&= \int_0^1 \sum_{j=1}^{n} (U[z]p[z]X)_{i,j}[e^{2\pi \iota t}] \overline{(U[z]p[z]X)_{i,j}}[e^{2\pi \iota t}] dt \\
&= \int_0^1 \left(\sum_{j=1}^{n} (U[z]p[z]X)_{i,j} \overline{(U[z]p[z]X)_{i,j}} \right) [e^{2\pi \iota t}] dt \\
&= \int_0^1 \left((U[z]p[z]X)(U[z]p[z]X)^* \right)_{i,i} [e^{2\pi \iota t}] dt = \int_0^1 \left(U[z]p[z]XX^*p^*[z]U^*[z] \right)_{i,i} [e^{2\pi \iota t}] dt \\
&= \int_0^1 \left(U[z]p[z]p^*[z]U^*[z] \right)_{i,i} [e^{2\pi \iota t}] dt = \int_0^1 \left(p[z]p^*[z] \right) \left(U[z]U^*[z] \right)_{i,i} [e^{2\pi \iota t}] dt \\
&= \int_0^1 \left(p[z]p^*[z] \right) [e^{2\pi \iota t}] dt = \int_0^1 \left| p[e^{2\pi \iota t}] \right|^2 dt = \|p\|^2 \ .
\end{aligned}$$

The first equality was by definition, the second by (5), the fourth by changing order of integration and summation, the fifth by the definition of Y, the sixth by our definition of polynomial conjugation, the seventh by the fact that polynomial evaluation commutes with polynomial multiplication, the eighth by definition of matrix multiplication, the ninth by properties of conjugation of products, the tenth by unitarity of X, the eleventh by commutativity of p (a polynomial over scalars) and P (a polynomial over matrices), the twelfth by paraunitarity of U, and the fourteenth again by (5).

Now, by our construction, MA equals $U[z]p[z]X \in \mathcal{X}$ for $U = M$, $X = \mathrm{Id}$, and

$$p[z] = 1 + \Delta z^{-1} + \Delta^2 z^{-2} + \cdots \Delta^{\rho+d+1+\ell} z^{-\rho-d-1-\ell} \ .$$

Using the technical claim,

$$\|A\|_{2,\infty} = \|p\| = \sqrt{\sum_{k=0}^{\rho+d+1\ell} \Delta^{2k}} \leq \sqrt{\frac{1}{1-\Delta^2}} = \sqrt{\frac{1}{1-(1-\mu)^2}} = \Theta\left(\sqrt{1/\mu}\right) = \Theta(\sqrt{\ell}) \,.$$

As for MB, notice that it equals $U[z]p[z]X \in \mathcal{X}$ for $U = M \cdot (M_\Delta^{(m)})^*$, $p = \sum_{i=\rho}^{\rho+\ell} z^{-i}$, $X = F$. Therefore again we use the technical claim to get $\|B\|_{2,\infty} = \|p\| = \sqrt{\sum_{k=\rho}^{\rho+\ell} 1} = \Theta(\sqrt{\ell})$. □

Let us now compute the preconditioned quasi-entropy of $M_\Delta^{(0)} = \mathrm{Id}$ and of $M_\Delta^{(m)}$.

Lemma 13.

$$\Phi_{A,B}(M_\Delta^{(m)}) = \Omega\left(\frac{\ell n \log n}{\sqrt{\kappa}}\right) \qquad \Phi_{A,B}(\mathrm{Id}) = o\left(\frac{\ell n \log n}{\sqrt{\kappa}}\right) \,.$$

Proof. For the left bound in the lemma statement:

$$\begin{aligned}
\Phi_{A,B}(M_\Delta^{(m)}) &= \sum_{k=-\rho-\ell}^{-\rho} \sum_{i,j} h\left(\mathrm{coeff}_k((M_\Delta^{(m)}A)_{i,j}) \cdot \overline{\mathrm{coeff}_k((M_\Delta^{(m)}B)_{i,j})}\right) \\
&= \sum_{k=-\rho-\ell}^{-\rho} \sum_{i,j} h\left((\Delta^{-k}F_{i,j}) \cdot (F_{i,j})\right) \qquad (12) \\
&= \sum_{k=-\rho-\ell}^{-\rho} \sum_{i,j} h(F_{i,j}^2\Delta^{-k}) = -\sum_{k=-\rho-\ell}^{-\rho} \sum_{i,j} \Delta^{-k}F_{i,j}^2 \log(\Delta^{-k}F_{i,j}^2) \,.
\end{aligned}$$

Recall that $\Delta^{-\rho} = \kappa^{1/2} = n^{o(1)}$. Also recall that by definition of ℓ, $\Delta^\ell = \Theta(1)$. Therefore $\log \Delta^{-k}$ is $o(\log n)$ for all $k \in R$, and

$$\begin{aligned}
\Phi_{A,B}(M_\Delta^{(m)}) &= -\sum_{k=-\rho-\ell}^{-\rho} \sum_{i,j} \left(\Delta^{-k}F_{i,j}^2 \log F_{i,j}^2 + o(\log n)\Delta^{-k}F_{i,j}^2\right) \\
&= \sum_{k=-\rho-\ell}^{-\rho} \Delta^{-k}\Phi(F) - o(\ell\Delta^\rho n \log n) = \Theta(\ell\Delta^\rho)\Phi(F) - o(\ell\Delta^\rho n \log n) \\
&= \Theta(\ell\Delta^\rho) n \log n - o(\ell\Delta^\rho n \log n) = \Omega(\ell\kappa^{-1/2} n \log n) \,,
\end{aligned}$$

where the second equality is from $\sum F_{i,j}^2 = n$ (unitarity of F) and definition of Φ, the third is from the fact that $\Delta^i = \Theta(1)$ for all $i = 0 \ldots \ell$, and the fourth from (8). For the right bound in the lemma statement:

$$\begin{aligned}
\Phi_{A,B}(\mathrm{Id}) &= \sum_k \sum_{i,j=1}^n h\left(\mathrm{coeff}_k((A)_{i,j}) \cdot \overline{\mathrm{coeff}_k((B)_{i,j})}\right) \\
&= \sum_{k=-\rho-\ell}^{-\rho} \sum_{i=1}^n h\left((\Delta^{-k}1) \cdot \overline{\mathrm{coeff}_k(B_{i,i})}\right) = \sum_{i=1}^n \sum_{k=-\rho-\ell}^{-\rho} h\left((\Delta^{-k}1) \cdot \overline{\mathrm{coeff}_k(B_{i,i})}\right) \,.
\end{aligned}$$

Fixing $i \in [n]$, we upper bound the absolute value of the last inner sum, which equals:

$$\underbrace{\sum_{k=-\rho-\ell}^{-\rho} \Delta^{-k} h\left(\overline{\mathrm{coeff}_k(B_{i,i})}\right)}_{E_1} - \underbrace{\sum_{k=-\rho-\ell}^{-\rho} \Delta^{-k}\overline{\mathrm{coeff}_k(B_{i,i})} \log \Delta^{-k}}_{E_2} \,.$$

To bound E_1, we first use Lemma 12, to argue that for all $i \in [n]$, $\|B_{i,:}\|^2 = \ell + 1$. (To be precise, it was only shown that $\|B_{i,:}\|^2 = \Theta(\ell)$ but it is obvious from the proof of Lemma 12 that the exact value is $\ell + 1$.) This in particular implies that $\sum_{k=-\rho-\ell}^{-\rho} |\mathrm{coeff}_k(B_{i,i})|^2 \leq \ell + 1$. We therefore bound E_1 by:

$$\sup_{\sum \tau_k^2 \leq \ell+1} \left| \sum_{k=-\rho-\ell}^{-\rho} \Delta^{-k}\tau_k \log|\tau_k| \right| \,. \qquad (13)$$

This is shown to be $\Theta(\kappa^{-1/2}\ell)$, using standard calculus, as follows. By the triangle inequality, (13) is at most:

$$\sup_{\sum \tau_k^2 \le \ell+1} \sum_{k=-\rho-\ell}^{-\rho} \Delta^{-k} |\tau_k \log |\tau_k|| \ . \tag{14}$$

We now use the fact that $\Delta^k = \Theta(\Delta^\rho) = \Theta(\kappa^{-1/2})$ for all $k \in R = \{\rho, \ldots, \rho+\ell\}$. Therefore (14) is at most a constant times

$$\kappa^{-1/2} \sup_{\sum \tau_k^2 \le \ell+1} \sum_{k=1}^{\ell+1} |\tau_k \log |\tau_k|| \ . \tag{15}$$

The contribution coming from k's such that $|\tau_k| \le 1$ to (15) is at most $\Theta(\kappa^{-1/2}\ell)$, because $|\tau \log |\tau||$ is a continuous function and hence bounded in the range $[-1, 1]$. We hence bound

$$\sup_{K \le \ell+1} \sup_{\left[\begin{smallmatrix} \forall k \le K : \tau_k > 1 \\ \sum \tau_k^2 \le \ell+1 \end{smallmatrix}\right]} \kappa^{-1/2} \sum_{k=1}^{K} \tau_k \log \tau_k \ . \tag{16}$$

Using the method of Lagrange multipliers, the inner supremum can only be obtained at one of the two points: (1) At the extreme candidate point ($\tau_1 = \sqrt{\ell - K + 2}, \tau_2 = \cdots = \tau_K = 1$), for which the function value is $O(\kappa^{-1/2}\sqrt{\ell}\log\ell)$, or (2) when ($\tau_1 = \cdots = \tau_K = \sqrt{\frac{\ell+1}{K}}$), for which case the value is $\kappa^{-1/2}\sqrt{K(\ell+1)}\log\sqrt{\frac{\ell+1}{K}}$. But the last expression, maximized over $K \in [1, \ell+1]$, is at most $\Theta(\ell)$, obtained at $K = \Theta(\ell)$. Combining these arguments, the value of (14) is $\Theta(\kappa^{-1/2}\ell)$, as required.

To bound E_2, we use the fact that for all $k \in R$, $\Delta^{-k} = \Theta(\Delta^\rho) = \Theta(\kappa^{-1/2})$ and $\log\kappa = o(\log n)$. Therefore $|E_2|$ is $o\left(\kappa^{-1/2}(\log n)\sum_{k=-\rho-\ell}^{-\rho} |\mathrm{coeff}_k(B_{i,i})|\right)$. But by the well known ℓ_1/ℓ_2 bound, we have that $\sum_{k=-\rho-\ell}^{-\rho} |\mathrm{coeff}_k(B_{i,i})| \le \ell + 1$. Therefore, $|E_2|$ is $o(\kappa^{-1/2}\ell\log n)$. Combining, we have that $|\Phi_{A,B}(\mathrm{Id})| = o(\ell\kappa^{-1/2}n\log n)$. □

Combining Lemma 13 and 10 with Claim 12, we get that the algorithm $\mathcal{A}$ increases the potential defined by the preconditioned quasi-entropy $\Phi_{A,B}(\cdot)$ from $o\left(\frac{\ell n \log n}{\sqrt{\kappa}}\right)$ to $\Omega\left(\frac{\ell n \log n}{\sqrt{\kappa}}\right)$, and at each step it can change the potential by no more than $\Theta(\sqrt{\ell})\Theta(\sqrt{\ell}) = \Theta(\ell)$. Hence, the number of steps is as stated in Theorem 9, concluding its proof. (Reminder: See Section Appendix A for generalization to the case $\mathcal{A}$ is integral with respect to no Δ.)

6. Main conjecture with some evidence, and future work

Our main conjecture, which is an algebraic version of Conjecture 1, is as follows:

Conjecture 14. *[Algebraic version of Conjecture 1] κ-Algebraic well conditioned computation allows only $O(\log(\kappa))$ speedup of FFT.*

There is interesting evidence for this conjecture, coming from recent work by [5]. Recall that in our construction, we started with an algorithm

$$\mathcal{A} = (\mathrm{Id} = M^{(0)}, M^{(1)}, \ldots, M^{(m)} = F)$$

working in $\mathbb{R}^n$, and mapped it to a pseudo-algorithm $\mathcal{A}_\Delta$ working in n dimensions over the ring of polynomial matrices. The pseudo-algorithm was used in the analysis only, and it was not assumed to be emulated. However, such an emulation could be done. We now explain how to emulate $\mathcal{A}_\Delta$ using reals (and not polynomials), and then find surprising connections to other work.

If all polynomial matrices in $\mathcal{A}_\Delta$ have monomials of degrees in the range $[-L, L]$, then they can be naturally embedded as real matrices of shape $n(2L+1) \times n(2L+1)$. To explain the embedding, fix a matrix $M[z]$ in the ring of polynomials over $n \times n$ real matrices with monomials in the range $[-L, L]$ only. The embedding will be denoted $\Psi(M) \in \mathbb{R}^{n(2L+1)\times n(2L+1)}$. Divide the index set of size $n(2L+1)$ into $2L+1$ block-indices of size n each. The value of the block-entry (i, j) of $\Psi(M)$ is $\mathrm{coeff}_{(j-i)}(X)$, where the integer $j - i$ is understood to be modulo $2L+1$ in the branch $[-L, L]$. It is not hard to see that this embedding is a homomorphism of the ring of polynomials modulo the identity ($z^{L+1} \equiv z^{-L}$), over $n \times n$ matrices.

The algorithm $\Psi(\mathcal{A}) := \{\Psi(M_\Delta^{(0)}), \ldots, \Psi(M_\Delta^{(m)})\}$ in $\mathbb{R}^{n(L+1)\times n(2L+1)}$ can be emulated as follows. A multiplication of row i of $M_\Delta^{(t)}$ by the monomial z^a for an integer a (giving $M_\Delta^{(t+1)}$) is emulated by performing a cyclic shift of the set of $(2L+1)$ rows in $\Psi(M_\Delta^{(t)})$. This set of rows corresponds exactly to the i'th index of all $(2L+1)$ index-blocks. The cyclic shift can be done by a sequence $2L$ swaps, each swap a rotation (a reflection, in fact). A rotation affecting rows i_1, i_2 of $M_\Delta^{(t)}$ is emulated by performing $2L+1$ rotations, affecting all pairs of rows corresponding to the pair (i_1, i_2) within each of the $2L+1$ index-blocks.

It is easy to be convinced of the emulation correctness. It is also easy to see that all matrices $\Psi(M_\Delta^{(t)})$ are unitary (this property could have been proven directly from the definition of Ψ and the para-unitarity of the $M_\Delta^{(t)}$'s). To summarize, we started with an algorithm over general $n \times n$ matrices making m steps, passed through an algorithm over paraunitary $n \times n$ polynomial matrices making m steps, and arrived at an algorithm over unitary $n(2L+1) \times n(2L+1)$ matrices making $\Theta(Lm)$ steps.

$M^{(t)}$	$\mapsto$	$M_\Delta^{(t)}[z]$	$\hookrightarrow$	$\Psi(M_\Delta^{(t)}[z])$
Original algorithm over reals $n \times n$ non-unitary		Abstract algorithm over polynomials $n \times n$ paraunitary		Emulation of abstract algorithm over reals $n(2L+1) \times n(2L+1)$ unitary

Now, what can be said about the final matrix in the emulation, $\Psi(M_\Delta^{(m)})$? We only care about the case $m = o(n \log n)$ of course, from which we have the entropy bound

$$\Phi(\Psi(M_\Delta^{(m)})) = o(Ln \log(Ln)) .$$

(This can be derived from the definition (7) together with Lemma 10, but it is recommended to use [1] for a simpler treatment of unitary computation.)

Take L to be $\rho + \ell$, as defined in Section 5, and assume that for all $t = 0..m$, $-\rho \le \mathrm{val}(M_\Delta^{(t)}) \le \deg(M_\Delta^{(t)}) \le \rho$. (This is a simplifying assumption that can be removed using a simple trick that we omit.) From Claim 11 we have that the matrix $\Psi(M_\Delta^{(m)})\Psi(A)$ (where A is the preconditioner from Section 5) contains $\Theta(\rho)$ copies of the matrix $F\Theta(1/\sqrt{\kappa})$ as blocks.[6] This is an interesting outcome: We have a matrix $\Psi(M_\Delta^{(m)})\Psi(A)$ of total low entropy (because m is small) which contains $\Theta(\rho)$ copies of F, each scaled down by $\Theta(\sqrt{\kappa})$.[7] This is an odd situation. We have computed, in our emulation, and using an algorithm of perfect condition number 1, a matrix of low entropy that contains (under a low-entropy preconditioning) a scaled-down high-entropy matrix as sub-matrices. What is the computational complexity of such a matrix?

Ailon and Gal tackled this question (see [5]), by studying a toy example: Is it possible to compute a "Fourier Perturbation":

$$\mathrm{Id} + \varepsilon F ,$$

for some small ε using an almost unitary algorithm? (Note that the matrix $\mathrm{Id} + \varepsilon F$ is not quite unitary, but very close.) The answer is that at least $\Omega((n \log n)/\log(1/\varepsilon))$ steps are needed in a computational model very close to unitary. The parameter ε corresponds to our $1/\sqrt{\kappa}$. If we could somehow generalize that result here, we would get a denominator of $\log \kappa$ in (4). Doing that seems to require exploring very interesting derivations of entropy functions, as in [5] for the toy problem. We were unable as of yet to obtain this generalization.

We do not have a matching algorithm for Conjecture 14. The choice of the $O(\log \kappa)$ speedup factor in the statement of the conjecture comes from our belief that the conjecture should be possible to prove, based on the reasoning above.

By Lemma 8 the following conjecture is stronger than Conjecture 14, and therefore harder to prove:

Conjecture 15. *[Geometric version of Conjecture 1] κ-(geometric) well conditioned computation allows only $O(\log(\kappa))$ speedup of FFT.*

It is interesting to study algebraic condition number in its own right as a measure of numerical stability. Consider the following related question: Lemma 8 tells us that if an algorithm is algebraically κ well conditioned then it is also (geometrically) κ-well conditioned. The converse is clearly not true, but perhaps some weak form of the converse should hold. In the extreme case, for example, note that an algorithm that is (geometrically) 1-well conditioned *must also be* algebraically 1-conditioned, because such an algorithm can only use constants 1 and -1. Could we generalize this observation to obtain some kind of converse for Lemma 8?

Some more noteworthy open problems:

- Extending the results to the extra-memory (non in-place) model.
- Obtaining computational lower bounds for computing a Johnson-Lindenstrauss transform. There has been work showing that a JL transform can be sped up using the Fourier transform ([4]), but computational lower bounds for Fourier do

[6] Note that $\Psi(M_\Delta^{(m)})\Psi(A)$ is not the same as $\Psi(M_\Delta^{(m)}A)$, due to modular wrap-around of monomials. But the argument is still correct.

[7] We also need to show that the preconditioner $\Psi(A)$ can affect the entropy of $\Psi(M_\Delta^{(m)})$ by at most $o(LN \log(LN))$, which we omit here for the sake of simplicity, and since this section is used for explaining intuitive evidence.

not imply computational lower bounds for JL. Note that there are tight lower bounds for dimensionality parameters of JL ([12]), but they are not computational. We also note the question of whether this line of work can be used to show lower bounds for the *sparse* Fourier transform ([8,10,11]).

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A. The general case: algorithms that are not integral

In this section we show how to extend the main result in this work to the case in which the algorithm $\mathcal{A}$ is integral with respect to no Δ.

A.1. Extension of 'well conditioned' definition to the general case

Fix an algorithm $\mathcal{A} = (\mathrm{Id} = M^{(0)}, \ldots, M^{(m)})$ in $\mathbb{R}^n$. Let

$$\Delta_1 < \Delta_2 < \cdots < \Delta_q < \ldots$$

be an infinite, increasing sequence of numbers in the open interval $\left(\frac{2}{3}, 1\right)$, tending to 1 in the limit. For each index q we define another algorithm $\mathcal{A}_q = (M_q^{(0}, \ldots, M_q^{(m)})$ inductively, as follows. First, $M_q^{(0)} = \mathrm{Id}$. For $t \geq 1$, $M_q^{(t)}$ is obtained from $M_q^{(t-1)}$ depending on whether $M^{(t)}$ is obtained from $M^{(t-1)}$ by a rotation or a constant gate. In case of rotation, $M_q^{(t)}$ is obtained by applying the same rotation to $M_q^{(t-1)}$. In case of a constant gate c acting on row i. The matrix $M_q^{(t)}$ is obtained from $M_q^{(t-1)}$ by multiplying the i'th row by

$$c_q := \Delta_q^{\lceil \log_{\Delta_q} c \rceil}.$$

The resulting algorithm $\mathcal{A}_q$ is clearly integral with respect to Δ_q. Additionally, it is not hard to see by standard limit calculus and by induction, that tor each $t = 0 \ldots m$, $\lim_{q\to\infty} M_q^{(t)} = M^{(t)}$. We are now ready to extend Definition 6 to the non integral case.

Definition 16. $\mathcal{A}$ is algebraically κ-well conditioned if for some sequence $\Delta_1 < \Delta_2 < \cdots$ as above, there exists a sequence $\kappa_1, \kappa_2, \ldots$ tending to κ, such that $\mathcal{A}_q$ is κ_q-well conditioned for all q (per Definition 6).

It should be noted that if $\mathcal{A}$ is algebraically κ-well conditioned, then any sequence $\Delta_1 < \Delta_2 < \cdots$ (and not just one) will serve as a witness for the definition. It should also be noted that Definition 16 coincides with Definition 6 for algorithms $\mathcal{A}$ that are Δ-integral for some Δ. This can be seen, for example, by choosing $\Delta_q = \Delta^{1/q}$ and $\kappa_q = \kappa$ for all q.

A.2. Proof of Lemma 8 for the general case

Assume $\mathcal{A} = (M^{(0)}, \ldots, M^{(m)})$ is algebraically κ-well conditioned (per the general Definition 16). Let $\Delta_1 < \Delta_2 < \cdots$ and $\kappa_1, \kappa_2, \ldots$ be as guaranteed to exist by the definition. Then by Lemma 8 (established for the integral case), the algorithm $\mathcal{A}_q$ is geometrically κ_q-well conditioned for all q. But the geometric condition number of a matrix is a continuous invariant in the domain of nonsingular matrices, because both spectral norm and matrix inverse are continuous functions over matrices. Therefore, for all $t = 0 \ldots m$, $M^{(t)}$ (which is also the limit of $M_q^{(t)}$) is geometrically κ-well conditioned.

A.3. Proof of Theorem 9 for the general case

Assume $\mathcal{A} = (M^{(0)}, \ldots, M^{(m)})$ is algebraically κ-well conditioned (per the general Definition 16). Let $\Delta_1 < \Delta_2 < \cdots$ and $\kappa_1, \kappa_2, \ldots$ be as guaranteed to exist by the definition. Let $F_q = M_q^{(m)}$, namely, the resulting matrix computed by $\mathcal{A}_q$. First we note that $\lim F_q = F$. For each q, we now want to use the statement of the theorem in the integral case, which we have proved. The problem is that the resulting matrix F_q does not necessarily have the two key properties of Fourier transform used in the theorem: (1) unitarity and (2) high potential $\Phi(F_q)$.

In order to adjust the proof for the case of Δ_q-integral algorithm $\mathcal{A}$ computing F_q (instead of F), we will first assume that q is large enough so that

$$\forall i, j \in [n] \quad |(F_q)_{i,j} - F_{i,j}| \leq |F_{i,j}|/2\,. \tag{A.1}$$

This is achievable, because $F_{i,j} \neq 0$ for all i, j, for both types of Fourier transform.[8] In particular, (A.1) implies:

$$\sum_{i,j} F_{i,j}(F_q)_{i,j} = \Theta(n) \qquad \Phi_F(F_q) := -\sum_{i,j} F_{i,j}(F_q)_{i,j} \log(F_{i,j}(F_q)_{i,j}) \geq \Omega(n \log n)\ . \tag{A.2}$$

The trick is now to use preconditioners A_q, B_q as defined in Section 5.

$$A_q = \mathrm{Id}(1 + \Delta_q z^{-1} + \Delta_q^2 z^{-2} + \Delta_q^3 z^{-3} + \cdots + \Delta_q^{\rho_q+\ell_q} z^{-\rho_q-\ell_q}) \quad B_q = ((M_q)^{(m)}_{\Delta_q})^* \sum_{i=\rho_q}^{\rho_q+\ell_q} z^{-i} F\ ,$$

where ρ_q is $\frac{\log \kappa_q}{2 \log \Delta_q^{-1}}$ and ℓ_q is the smallest integer such that $\Delta_k^{\ell} \leq 1/2$. Note that in the definition of B_q we used F and not F_q. This is important, because we need Claim 12 and Lemma 13 to work (they rely on B being a product of a unitary matrix, a paraunitary matrix and a polynomial over scalars). Lemma 13 now reads as

$$\Phi_{A_q,B_q}((M_q)^{(m)}_{\Delta}) = \Omega\left(\frac{\ell_q n \log n}{\sqrt{\kappa_q}}\right) \tag{A.3}$$

$$\Phi_{A_q,B_q}(\mathrm{Id}) = o\left(\frac{\ell n \log n}{\sqrt{\kappa_q}}\right) \tag{A.4}$$

For the proof of Lemma 13, the only thing worth noting is that the expression in line (12) becomes

$$\sum_{k=-\rho_q-\ell_q}^{-\rho_q} \sum_{i,j} h\left((\Delta_q^{-k}(F_q)_{i,j}) \cdot (F_{i,j})\right)\ ,$$

that is, one copy of $F_{i,j}$ (coming from the preconditioner A_q) is replaced by $(F_q)_{i,j}$, and the other remains untouched. The remainder of the proof of the lemma proceeds in an obvious way, taking advantage of (A.2).

The resulting lower bound for the number m of steps that the algebraically κ_q-well conditioned algorithm $\mathcal{A}_q$ needs to compute F_q is

$$C \frac{\Phi_F(F_q)}{\sqrt{\kappa_q}}\ ,$$

for sufficiently large q, for a global C. Taking q to infinity and recalling that $\kappa_q \to \kappa$ we conclude Theorem 9's proof for the general case.

Appendix B. Proof of Lemma 10

Without loss of generality, assume the rotation is applied to rows 1 and 2 of M, so

$$M' = \begin{pmatrix} \cos\theta & \sin\theta & & & \\ -\sin\theta & \cos\theta & & & \\ & & 1 & & \\ & & & \ddots & \\ & & & & 1 \end{pmatrix} M \tag{B.1}$$

for some $\theta \in [0, 2\pi)$.[9] It suffices to consider only the change in the contribution of rows 1 and 2 to the potential. The contribution to the pre-rotation potential coming from the first two rows, which we denote $\Phi_{1,2}$, is

$$\Phi_{1,2} = \sum_k \sum_{j=1}^{n} \sum_{i=1}^{2} h\left(\mathrm{coeff}_k((MA)_{i,j})\overline{\mathrm{coeff}_k((MB)_{i,j})}\right).$$

Similarly, the contribution to the post-rotation potential coming from the first two rows, denoted $\Phi'_{1,2}$, is

$$\Phi'_{1,2} = \sum_k \sum_{j=1}^{n} \sum_{i=1}^{2} h\left(\mathrm{coeff}_k((M'A)_{i,j})\overline{\mathrm{coeff}_k((M'B)_{i,j})}\right).$$

[8] We could also make due with zero valued matrix elements, but for simplicity we avoid this technicality here.

[9] To be technically precise, we also need to take into consideration reflections. But these can be composed as a product of a matrix as in (B.1) and a matrix with ± 1 on the diagonal. It is trivial that such a matrix makes no difference to the potential.

By the triangle inequality:

$$|\Phi_{1,2} - \Phi'_{1,2}| \leq \sum_k \sum_{j=1}^{n} \Bigg| \underbrace{\sum_{i=1}^{2} h\left(\mathrm{coeff}_k((MA)_{i,j})\overline{\mathrm{coeff}_k((MB)_{i,j})}\right) - h\left(\mathrm{coeff}_k((M'A)_{i,j})\overline{\mathrm{coeff}_k((M'B)_{i,j})}\right)}_{\delta_{k,j}} \Bigg| .$$

To avoid clutter, let

$$\alpha_{k,i,j} = \mathrm{coeff}_k((MA)_{i,j}),\ \beta_{k,i,j} = \mathrm{coeff}_k((MB)_{i,j}),\ \alpha'_{k,i,j} = \mathrm{coeff}_k((M'A)_{i,j}),\ \beta'_{k,i,j} = \mathrm{coeff}_k((M'B)_{i,j}) .$$

For each k, j, let

$$\rho_{k,j} := \sqrt{|\alpha_{k,1,j}|^2 + |\alpha_{k,2,j}|^2} = \sqrt{|\alpha'_{k,1,j}|^2 + |\alpha'_{k,2,j}|^2} \tag{B.2}$$

$$\sigma_{k,j} := \sqrt{|\beta_{k,1,j}|^2 + |\beta_{k,2,j}|^2} = \sqrt{|\beta'_{k,1,j}|^2 + |\beta'_{k,2,j}|^2} , \tag{B.3}$$

where the right hand equality in both (B.2) and (B.3) is from the fact that a rotation preserves inner products (and in particular norms) in $\mathbb{C}^2$. By the definition of h,

$$\begin{aligned} h(\alpha_{k,1,j}\overline{\beta_{k,1,j}}) &= -\alpha_{k,1,j}\overline{\beta_{k,1,j}} \log |\alpha_{k,1,j}\overline{\beta_{k,1,j}}| \\ &= \alpha_{k,1,j}\overline{\beta_{k,1,j}} \log \left| \rho_{k,j}\sigma_{k,j} \frac{\alpha_{k,1,j}\overline{\beta_{k,1,j}}}{\rho_{k,j}\sigma_{k,j}} \right| \\ &= \underbrace{-\alpha_{k,1,j}\overline{\beta_{k,1,j}} \log |\rho_{k,j}\sigma_{k,j}|}_{(*)} \underbrace{-\rho_{k,j}\sigma_{k,j} \frac{\alpha_{k,1,j}\overline{\beta_{k,1,j}}}{\rho_{k,j}\sigma_{k,j}} \log \left| \frac{\alpha_{k,1,j}\overline{\beta_{k,1,j}}}{\rho_{k,j}\sigma_{k,j}} \right|}_{(**)} \end{aligned}$$

where we simply multiplied and divided by $\rho_{k,j}\sigma_{k,j}$ inside the logarithm to get the second line, then used logarithm properties and multiplied and divided by $\rho_{k,j}\sigma_{k,j}$ to get the third. Similarly:

$$h(\alpha_{k,2,j}\overline{\beta_{k,2,j}}) = \underbrace{-\alpha_{k,2,j}\overline{\beta_{k,2,j}} \log |\rho_{k,j}\sigma_{k,j}|}_{(*)} \underbrace{-\rho_{k,j}\sigma_{k,j} \frac{\alpha_{k,2,j}\overline{\beta_{k,2,j}}}{\rho_{k,j}\sigma_{k,j}} \log \left| \frac{\alpha_{k,2,j}\overline{\beta_{k,2,j}}}{\rho_{k,j}\sigma_{k,j}} \right|}_{(**)}$$

$$h(\alpha'_{k,1,j}\overline{\beta'_{k,1,j}}) = \underbrace{-\alpha'_{k,1,j}\overline{\beta'_{k,1,j}} \log |\rho_{k,j}\sigma_{k,j}|}_{(*)} \underbrace{-\rho_{k,j}\sigma_{k,j} \frac{\alpha'_{k,1,j}\overline{\beta'_{k,1,j}}}{\rho_{k,j}\sigma_{k,j}} \log \left| \frac{\alpha'_{k,1,j}\overline{\beta'_{k,1,j}}}{\rho_{k,j}\sigma_{k,j}} \right|}_{(**)}$$

$$h(\alpha'_{k,2,j}\overline{\beta'_{k,2,j}}) = \underbrace{-\alpha'_{k,2,j}\overline{\beta'_{k,2,j}} \log |\rho_{k,j}\sigma_{k,j}|}_{(*)} \underbrace{-\rho_{k,j}\sigma_{k,j} \frac{\alpha'_{k,2,j}\overline{\beta'_{k,2,j}}}{\rho_{k,j}\sigma_{k,j}} \log \left| \frac{\alpha'_{k,2,j}\overline{\beta'_{k,2,j}}}{\rho_{k,j}\sigma_{k,j}} \right|}_{(**)}$$

Now to estimate $\delta_{k,j} = h(\alpha_{k,1,j}\overline{\beta_{k,1,j}}) + h(\alpha_{k,2,j}\overline{\beta_{k,2,j}}) - h(\alpha'_{k,1,j}\overline{\beta'_{k,1,j}}) - h(\alpha'_{k,2,j}\overline{\beta'_{k,2,j}})$, we first note that the contribution coming from the (*) expressions vanishes. Indeed, rotation preserves complex inner product, and hence

$$\alpha_{k,1,j}\overline{\beta_{k,1,j}} + \alpha_{k,2,j}\overline{\beta_{k,2,j}} = \alpha'_{k,1,j}\overline{\beta'_{k,1,j}} + \alpha'_{k,2,j}\overline{\beta'_{k,2,j}} .$$

To estimate the contribution coming from the (**) expressions, we simply note that the four fractions $\frac{\alpha_{k,1,j}\overline{\beta_{k,1,j}}}{\rho_{k,j}\sigma_{k,j}}$, $\frac{\alpha_{k,2,j}\overline{\beta_{k,2,j}}}{\rho_{k,j}\sigma_{k,j}}$, $\frac{\alpha'_{k,1,j}\overline{\beta'_{k,1,j}}}{\rho_{k,j}\sigma_{k,j}}$, $\frac{\alpha'_{k,2,j}\overline{\beta'_{k,2,j}}}{\rho_{k,j}\sigma_{k,j}}$ are at most 1 in absolute value. This implies that $|\delta_{k,j}| = O(\rho_{k,j}\sigma_{k,j})$. Summing up over all k and j, we get

$$\sum_{k,j} |\delta_{k,j}| \leq O\left(\sum_{k,j} \rho_{k,j}\sigma_{k,j}\right) \leq O\left(\sqrt{\left(\sum_{k,j} \rho_{k,j}^2\right)\left(\sum_{k,j} \sigma_{k,j}^2\right)}\right)$$

$$= O\left(\sqrt{\left(\|(MA)_{1,:}\|^2 + \|(MA)_{2,:}\|^2\right)\left(\|(MB)_{1,:}\|^2 + \|(MB)_{2,:}\|^2\right)}\right)$$
$$= O\left(\|MA\|_{2,\infty} \cdot \|MB\|_{2,\infty}\right).$$

This concludes the proof.

References

[1] N. Ailon, A lower bound for Fourier transform computation in a linear model over 2x2 unitary gates using matrix entropy, Chic. J. Theor. Comput. Sci. (2013).
[2] N. Ailon, Tighter Fourier transform lower bounds, in: Automata, Languages, and Programming - 42nd International Colloquium, Proceedings, Part I, ICALP 2015, Kyoto, Japan, July 6–10, 2015, 2015, pp. 14–25.
[3] N. Ailon, An omega((n log n)/R) lower bound for Fourier transform computation in the r-well conditioned model, ACM Trans. Comput. Theory 8 (1) (2016) 4.
[4] N. Ailon, B. Chazelle, The fast Johnson–Lindenstrauss transform and approximate nearest neighbors, SIAM J. Comput. 39 (1) (2009) 302–322.
[5] N. Ailon, G. Yehuda, The complexity of computing a Fourier perturbation, CoRR, arXiv:1604.02557 [abs], 2016.
[6] J.W. Cooley, J.W. Tukey, An algorithm for the machine computation of complex Fourier series, J. Am. Math. Soc. (1964) 297–301.
[7] N. Dunford, J. Schwartz, Linear Operators, Vol. I, Interscience, New York, 1958.
[8] A.C. Gilbert, P. Indyk, M. Iwen, L. Schmidt, Recent developments in the sparse Fourier transform: a compressed Fourier transform for big data, IEEE Signal Process. Mag. 31 (5) (2014) 91–100.
[9] L. Hales, S. Hallgren, An improved quantum Fourier transform algorithm and applications, in: Proceedings of the 41st Annual Symposium on Foundations of Computer Science, 2000, pp. 515–525.
[10] P. Indyk, M. Kapralov, Sample-optimal Fourier sampling in any constant dimension, in: 55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18–21, 2014, 2014, pp. 514–523.
[11] P. Indyk, M. Kapralov, E. Price, (Nearly) sample-optimal sparse Fourier transform, in: Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5–7, 2014, 2014, pp. 480–499.
[12] K.G. Larsen, J. Nelson, The Johnson-Lindenstrauss lemma is optimal for linear dimensionality reduction, in: 43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11–15, 2016, Rome, Italy, 2016, pp. 82:1–82:11.
[13] V.Y. Pan, The trade-off between the additive complexity and the asynchronicity of linear and bilinear algorithms, Inf. Process. Lett. 22 (1) (1986) 11–14.
[14] C.H. Papadimitriou, Optimality of the fast Fourier transform, J. ACM 26 (1) (Jan. 1979) 95–102.
[15] P.P. Vaidyanathan, Multirate Systems and Filter Banks, 1st edition, Prentice Hall P T R, Englewood Cliffs, NJ, 1993.
[16] S. Winograd, On computing the discrete Fourier transform, Proc. Nat. Assoc. Sci. 73 (4) (1976) 1005–1006.
[17] S. Winograd, Some bilinear forms whose multiplicative complexity depends on the field of constants, Math. Syst. Theory 10 (1976) 169.