2015

Pytracks: A Tool for Visualizing Fish Movement Tracks on Different Scales

Ross Fossum
Louisiana State University and Agricultural and Mechanical College

PYTRACKS: A TOOL FOR VISUALIZING FISH MOVEMENT TRACKS ON DIFFERENT SCALES

A Thesis

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Master of Computer Science

in

The Department of Computer Science and Engineering

by
Ross P. Fossum
B.S. in Coastal Environmental Science, Louisiana State University, 2013
May 2016

Acknowledgments

I would like to thank Dr. Bijaya Karki and Dr. Kenneth Rose for supervising me to carry out an interdisciplinary research work for my masters thesis bridged between the Oceanography and Coastal Sciences and Computer Science and Electrical Engineering departments of LSU. Their guidance and patience with me allowed me to finish this thesis. My last, but not least, committee member Dr. Anas "Nash" Mahmoud has helped me greatly with his guidance and experience. He gave me many ideas on how to effectively visualize this data. I would finally like to thank Dr. Konstantin "Costas" Busch for stepping in as the chair of advisory committee when Dr. Karki was not able to serve.

All of the support of my committee allowed me to create a thesis I would of never been able to do on my own.

I would also like to thank my family and friends who have also supported and helped me throughout my studies. I would not be here if it wasn't for the love and support they provide.

Contents

Abstract

A fundamental problem in conservation biology and fisheries management is the ability to make educated decisions based on the data collected. Fish populations and their spatial distributions need to be represented accurately for conversation efforts and management decisions. Methods such as modeling, surveying, and tracking can all be used to collect data on a particular fishery. To include the movement patterns in conservation and management, one needs to work with and process fish tracking data or data exported from fish movement simulation models. This data can often be difficult to process. This topic is becoming increasingly popular as technology to accurately track and log fish did not exist in the past. With all of this data being generated, real or simulated, tools need to be developed to efficiently process it all, as many do not exist. Pytracks attempts to fill a currently existing gap and help programmers who work with simulated and observed simulation data by allowing them to visualize and analyze their data more efficiently. Pytracks, as presented in this thesis, is a tool written in Python which wraps raw data files from field observations or simulation models with an easy to use API. This allows programmers to spend less time on trivial raw file processing and more time on data visualization and computation. The code to visualize sample data can also be much shorter and easier to interpret. In this thesis, pytracks was used to help solve a problem related to interpreting different movement algorithms. This work has a focus on fish movement models, but can also be relevant for any other type of animal if the data is compatible. Many examples have been included in this thesis to justify the effectiveness of pytracks. Additional online documentation has been written as well to show how to further utilize pytracks.

Chapter 1

Introduction

A trait which is common between almost all aquatic lifeforms is their ability to move (Diamond et al. 2013). They could move on their own accord, or be forced by an external force such as currents and other organisms (Campbell et al. 2011). When animals move, they usually always have reasons to move. The most basic reasons behind animal movement are to seek out more favorable conditions and avoid unfavorable ones (Crouse et al. 1987). Collectively, because every organism is moving around, they transport nutrients, biomass, and dynamic energy within ecosystems and across distinct ecosystems (Hussey et al. 2015).

To study this further, technology needs to be able to effectively track an animal. Recently developed aquatic telemetry technology allows us now to effectively and cheaply track animals (Creekmore 2011). Today, there are two main ways to track aquatic animals using observations, acoustic and satellite telemetry (Hussey et al. 2015). Animals which are tagged with acoustic devices are detected by receivers based on the ground, such as on a buoy or boat. Other devices can communicate via satellites overhead which then relay the data to a land based data collection facility (Xu et al. 2013). Along with the technology to effectively track animals, high resolution satellite imagery has allowed us to accurately capture and describe the environments these organisms live in (Yemane et al. 2009).

Individual based animal movement models usually have two components. The first compo-

nent is the grid, or the simulated environment. The second component is the movement model, which is then used to simulate the movements of individuals on the grid. An example of a grid could be a lake or a section of a sea. This grid will be broken up into cells, each containing data about the conditions in that cell (Khurana et al. 2012). For a model, the two most basic metrics or indicators of habitat quality describing a cell is the mortality factor, which describes how detrimental the cell is to the individual, and the growth potential, which describes how fruitful the cell is for the individuals which pass through it (Gu 2010). Together, these two metrics can be subtracted to achieve the habitat quality (Watkins & Rose 2014) which is the type of data used on the grids in this thesis:

(1) $HabitatQuality = Growth - Mortality$

The movement model portion of the simulation is more complex than the grid portion due to there being more information calculations involved (Xu et al. 2013). These models usually simulate and follow the entire lifetime of multiple tracks, with each track starting on the grid at a determined location. These simulations usually have very complex calculations in attempt to closely simulate what a real fish would do when put into a similar grid (Intel IT Center 2013). In these models, multiple formulas are used which attempt to simulate multiple aspects of a fish's life (SAS 2012). Formulas and rules need to be derived for components of the model such as the food uptake, biomass accumulation and predation. This is not the focus of this thesis. For further information, please refer to (Watkins & Rose 2014) which discuss the details of the movement models.

Chapter 2

Methods

2.1 Requirements

The goal of this thesis was to develop a tool which allows users to more easily deal with the grid and tracks data generated by fish movement models. The tool is easily adapted to more irregularly sampled tracks data from observations. Depending on the amount of details required, these files are often quite large and have a lot of multivariate data for the user to potentially utilize. If the proposed tool made wrapping this data in a format which allows a user to more easily use it, then this type of data can be more accessible to more people.

As the goal of this tool was to make the data from these models more accessible, the programming language Python was chosen. Along with being a fairly easy language to learn, Python is also very powerful. Due to its popularity, there are many libraries built using Python. Python is also widely used in the scientific community because of its ease of data calculation and visualization (Kumar 2015). Pytracks does this by building on what Python does already; making programming accessible to more people. Pytracks expresses the data raw text files exported by the modeling problems to the user in a pythonic way, saving the user effort and development time for their potential research.

The place that pytracks fits into a user's visualization research or project using data generated from a movement model can be shown in Figure 2.1 below.

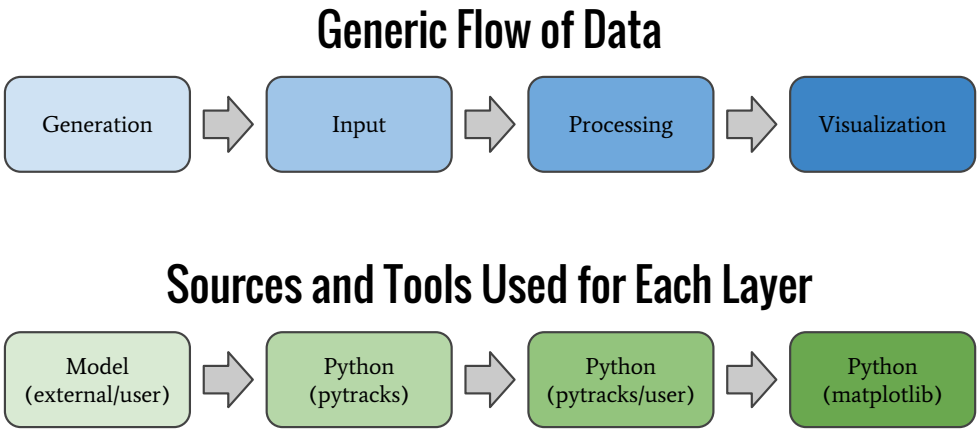

Figure 2.1: Four step flowchart describing steps required to visualize movement model data. On the top are the steps, and on the bottom is what users usually use and what was also used in this work. In the parenthesis shows that pytracks handles the input step and part of the processing step. If pytracks is not used, then the user would have to write all the code to input data and also some to process the data. This shows how much pytracks helps with the entire process.

2.2 Pytracks Structure

As Python libraries should be, pytracks was coded in a modular fashion. There are 4 sub-modules and 7 main classes in pytracks, each wrapping a specific part of the raw data the user specified to read in. Figure 2.2 shows a flowchart of how the data is processed by pytracks. Table 2.1 gives a short description of each module and what classes that module contains.

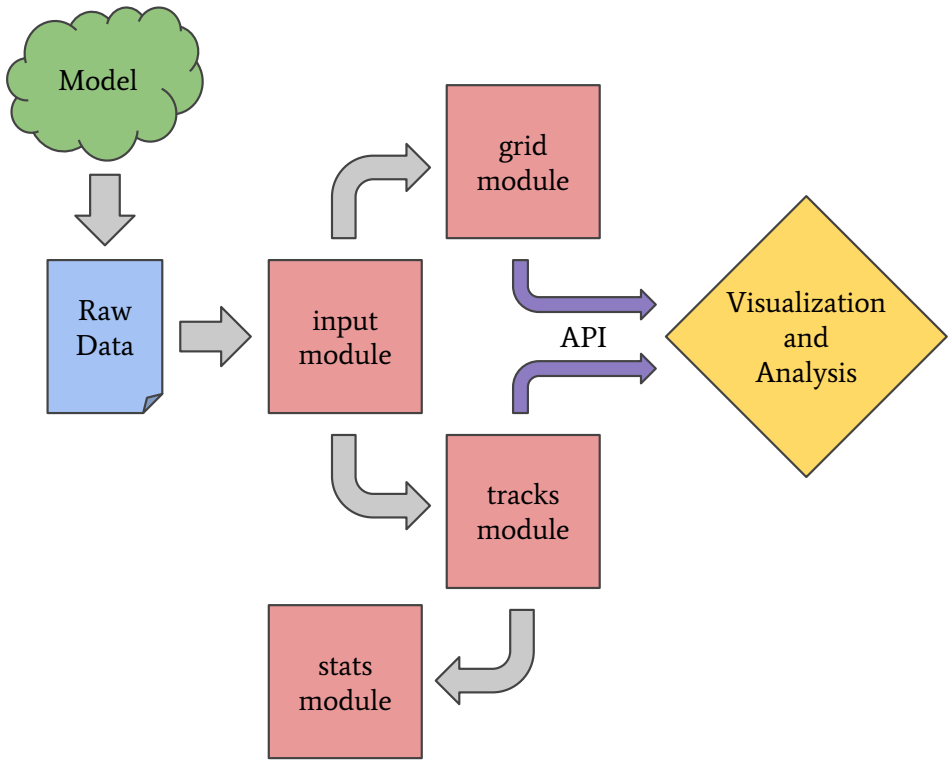

Figure 2.2: A flowchart which shows the movement of data through pytracks. Raw data is generated by a model and then passed to the input module. The input module then translates the raw data into usable objects for the track and grid modules. These modules can then be used to visualize the data. Also, the track module has the ability to pass data to the stats module to then allow the user to run simple statistical analysis on the data read in.

Table 2.1: The modular structure of pytracks.

Module	Class	Description
input	GridWrapper	Wraps a grid data file into a Grid object
input	TrackWrapper	Wraps a track data file into a TrackSet object
grid	Cell	Describes a single cell in the grid
grid	Grid	Holds Cell classes used to describe the grid
track	Track	Describes the entire lifetime of a single track
track	TrackSet	Holds Track classes with internal methods
stats	Stats	Generates a statistical overview of the track data

Chapter 3

Data

3.1 Source

In all of the examples shown, the data used was generated by movement models written by Katherine Shepard Watkins and Kenneth A. Rose. Further detail about the construction of this model can be found in (Watkins & Rose 2014). The data extracted for analysis in this thesis was the first grid and both resolutions described in the paper.

3.2 Background

Pytracks is built around accepting data from individual based models which use *super-individuals*. *Super-individuals* refers to a particle which represents some number of identical individuals (could be a school) with all of the individuals of a super-individual assumed to behave like the super-individual. (Barbaro et al. 2009). It is a method used by modelers to create more statistically viable data for their analysis but also keeping computational expense to a manageable level. This is why *worth* is included in the model's output. *Worth* is an intuitive name, and simply refers to the amount which that particular individual is worth. For example, if a super-individual represents 1000 individuals, then it's *worth* is 1000 whenever output statistics are computed about that individual. For example, the average

weight of two super-individuals with weights 10 and 20 and worths 1000 and 990 would be:

(2) $14.73 = (10x1000 + 20x900)/(1000 + 900)$

3.3 Raw Data Requirements

This section explains the format in which the data must be in to be usable by pytracks.

3.3.1 File Requirements

1. Raw text file
2. One entry per line
3. Tab or white-space delimited
4. File can hold multiple sets of data. See the *Sectioned* parameter in the input methods in `pytracks.input`.

Table 3.1 and Table 3.2 below show the format pytracks expects the data to be in by default. The values below are the default ids used in the program. Every column ID can be changed except *Section*, or *Sect*, which must be the first column and thus have an ID of 0. The user can have multiple extra data columns, which must be specified when wrapping the data file with the appropriate function.

3.3.2 Grid File

Table 3.1: The structure of a grid data file pytracks expects by default. This format can be modified when wrapping the file with *GridWrapper*.

Sect	X	Y	Extra
0	1	2	3-N

3.3.3 Tracks File

Table 3.2: The structure of a tracks data file pytracks expects by default. This format can be modified when wrapping the file with *TrackWrapper*.

Sect	Tick	Track	X	Y	Grow	Mort	Worth	Weight	Extra
0	1	2	3	4	5	6	7	8	9-N

Chapter 4

Example Usage

4.1 Introduction

As the major use of pytracks is to make it easier for people to use data exported by simulation models or collected from field observations, it has the potential to be used in a variety of ways. Users can take the data and run statistical analysis on it. They can also visualize the data using various well known Python visualization modules. Although pytracks has many uses besides just visualization, most of the examples which will be discussed below will only be centered around pytracks helping users visualize data.

One major module which is used by many people who use Python to visualize data is a module called matplotlib. This allows the user to visualize data similar to one would visualize data in the fairly well known program called MatLab. Pytracks is built to closely work with matplotlib to help users create visualizations more efficiently. Multiple examples were written in this work to show how one can use matplotlib with pytracks.

4.1.1 Note

The examples below will not go into much detail about the matplotlib code used. If one is to use pytracks, they need to be familiar with a visualization module or tool. The user can

then graph or process it using whichever utility they may desire. The focus of the examples below is to show pytrack's potential.

4.2 Grid Visualization

This example shows how pytracks makes it easier for a user to read raw data from a file and visualize a two-dimensional grid using matplotlib.

4.2.1 Data Input

Firstly, a user needs to read in the raw data. Pytracks has the built in ability to easily read in data which follows a flexible formatting scheme:

```
grid_wrapper = pytracks.input.GridWrapper("grid_data.out", extra_ids=[3, 4])
```

Check the Data chapter for the data format requirements. Pytracks allows the user to specify exactly the data they require in their script, as specified by the *extra_ids* parameter.

4.2.2 Data Processing

The previous method created a **GridWrapper** class, which allows us to access the data in raw form if one desires, or generate a **Grid** class::

```
grid = grid_wrapper.gen_grid()
```

Using the **Grid** class we generated, under the variable name *grid*, we can now generate data usable by matplotlib:

```
plot_data = numpy.zeros(grid.size)
for cell in grid.cells:
   plot_data[cell.y - 1][cell.x - 1] = (cell[0] - cell[1])
```

This generates data and stores it in the *plot_data* variable for use in the visualization code.

4.2.3 Visualization

The data generated earlier is used to generate a graph displaying the simulated grid using matplotlib:

```
axis.set_title("Grid Visualization")
colorbar = plot.colorbar(grid_image)
colorbar.set_ticks([-1, 0, 1])
colorbar.set_ticklabels([-1, 0, 1])
colorbar.set_label("Habitat Quality")

plot.savefig("export/grid.pdf", bbox_inches='tight', transparent=True)
plot.show()
```

4.2.4 Output

Output from this example can be seen in Figure 4.1.

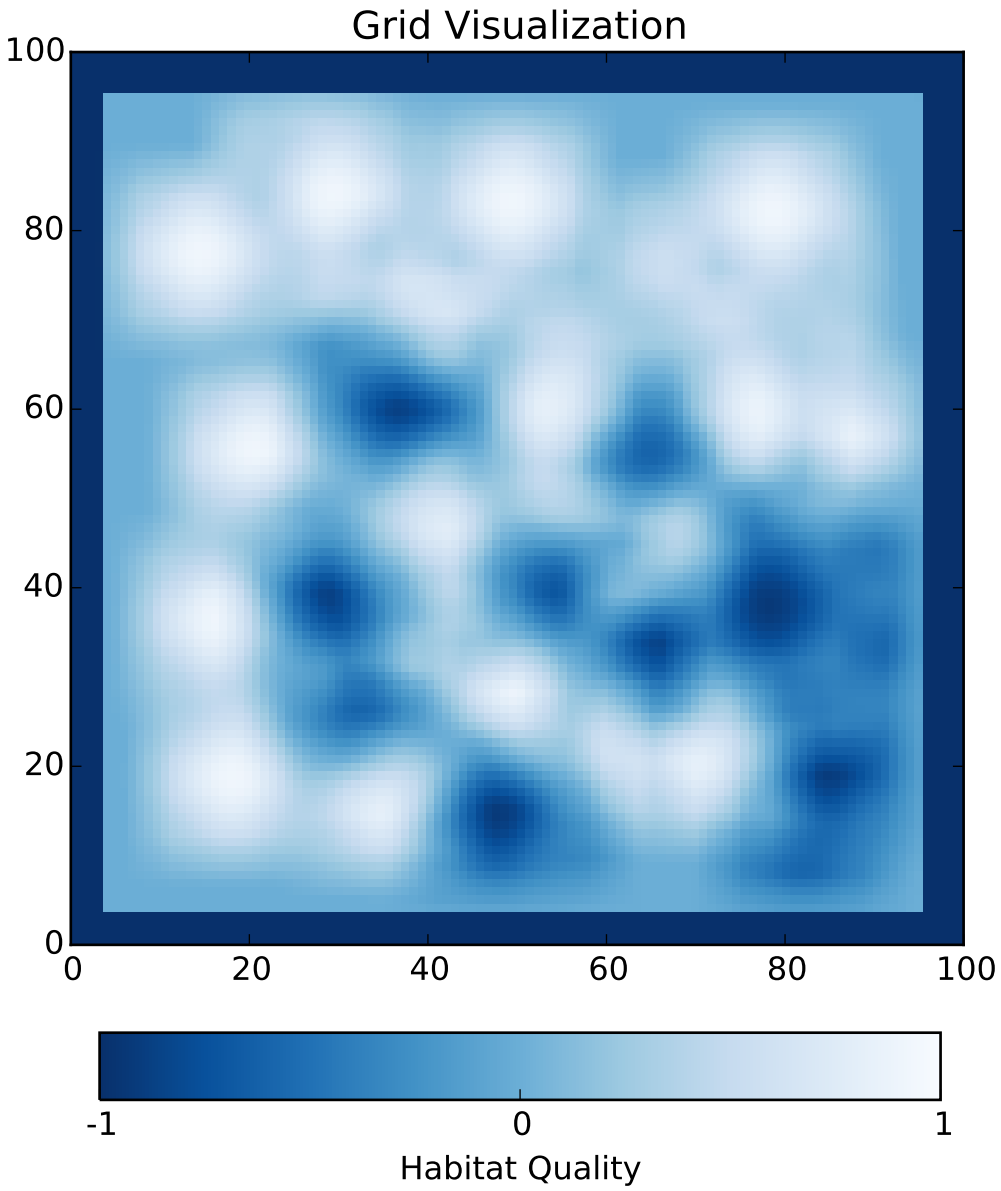

Figure 4.1: Example of visualizing a grid using pytracks. The data above was taken from simulated movement data generated by the model in (Watkins & Rose 2014) and shows the habitat quality in each cell using the color map technique. The habitat quality can range from (-1) to 1, as shown on the colorbar. A habitat quality of 1 is favorable, and a habitat quality of (-1) is unfavorable. This means that fish will avoid the dark blue areas and try to move to the white areas. All of the further examples will use track data generated from a model running on this grid.

4.3 Track Visualization

This example shows how a user can use pytracks to extract data from a grid to show how well a individual does over its lifetime. The *biomass* of each individual is encoded along the fish tracks using the color-map technique. The brown shading relates to the biomass amount as it moves through the grid. Darker means lower biomass, and lighter means higher biomass. The green and red dots refer to the start and end points respectfully.

4.3.1 Setup Functions

To create the lines which track the biomass magnitude, we need two functions which help us with that. One function creates the line using arrays of coordinates and the data, while the other creates the segments:

```
def colorline(x, y, data, normalize=plot.Normalize(0.0, 1.0)):
    z = numpy.asarray(data)
    segments = make_segments(x, y)
    lc = LineCollection(segments, array=z, cmap=plot.get_cmap('copper'),
         norm=normalize)
    ax = plot.gca()
    ax.add_collection(lc)
    return lc


def make_segments(x, y):
    points = numpy.array([x, y]).T.reshape(-1, 1, 2)
    segments = numpy.concatenate([points[:-1], points[1:]], axis=1)
    return segments
```

4.3.2 Data Input

As before, the user needs to read in the raw data and generate the pytracks classes.

```
grid_wrapper = pytracks.input.GridWrapper("event_25/grid.out", extra_ids=[3, 4])
tracks_wrapper = pytracks.input.TrackWrapper("event_25/Event_5.out", id_column=2,
                 weight_column=3, worth_column=4, x_column=5, y_column=7, g_column=14,
                 m_column=15)

grid = grid_wrapper.gen_grid()
trackset = tracks_wrapper.gen_trackset()
```

4.3.3 Grid Visualization

As done in the previous example, the **Grid** needs to be visualized again:

```
plot_data = numpy.zeros(grid.size)
for cell in grid.cells:
    plot_data[cell.y - 1][cell.x - 1] = (cell[0] - cell[1])
```

4.3.4 Figure Setup

The user needs to initialize the figure so it can be drawn on:

```
figure, axis = plot.subplots(figsize=(6, 10))
```

4.3.5 Fetching a Sample Track

The following line fetches a random track from the **TrackSet** created with the data read in. We then get the only track in the set.

```
newset = trackset.get_tracks_random(1)

track = newset[0]
```

4.3.6 Drawing Start and End Points

We then need to indicate where the **Track** started and where it ended:

```
area = numpy.pi * (5)**2 # dot radius of 5

plot.scatter(track.x[0]/25, track.y[0]/25, c="green", s=area, zorder=3)

plot.scatter(track.x[-1]/25, track.y[-1]/25, c="red", s=area, zorder=3)
```

4.3.7 Drawing the Path

Then, we will draw the path and save the figure:

```
path = Path(numpy.column_stack([track.x/25, track.y/25]))

verts = path.interpolated(steps=3).vertices

x, y = verts[:, 0], verts[:, 1]

data = numpy.true_divide(track.biomasses, max_biomass)


max_ratio = max(data)

min_ratio = min(data)


axis.add_collection(colorline(x, y, data, plot.Normalize(min_ratio, max_ratio)))


axis.set_title("Lifetime - Biomass")

axis.set_xlim([0, 100])

axis.set_ylim([0, 100])


figure.subplots_adjust(bottom=0.235)
```

```
colorbar_axis = figure.add_axes([0.15, .22, .73, .05])


grid_image = axis.imshow(plot_data, interpolation='none', origin="lower",
             cmap=plot.get_cmap("Blues_r"), vmin=-1, vmax=1,
             extent=[0, 100, 0, 100], aspect="equal")
colorbar_hq = plot.colorbar(grid_image, cax=colorbar_axis, orientation='horizontal')
colorbar_hq.set_ticks([-1, 0, 1])
colorbar_hq.set_ticklabels([-1, 0, 1])
colorbar_hq.set_label("Habitat Quality")


colorbar_tq_axis = figure.add_axes([0.15, .09, .73, .05])
colorbar_tq = ColorbarBase(ax=colorbar_tq_axis, cmap=plot.get_cmap('copper'),
              norm=matplotlib.colors.Normalize(0, 1), orientation='horizontal')
colorbar_tq.set_ticks([0, 1])
colorbar_tq.set_ticklabels(["Low", "High"])
colorbar_tq.set_label("Biomass")


plot.savefig("export/track_lifetime.pdf", bbox_inches='tight', transparent=True)
plot.show()
```

4.3.8 Output

Output from this example can be seen in Figure 4.2.

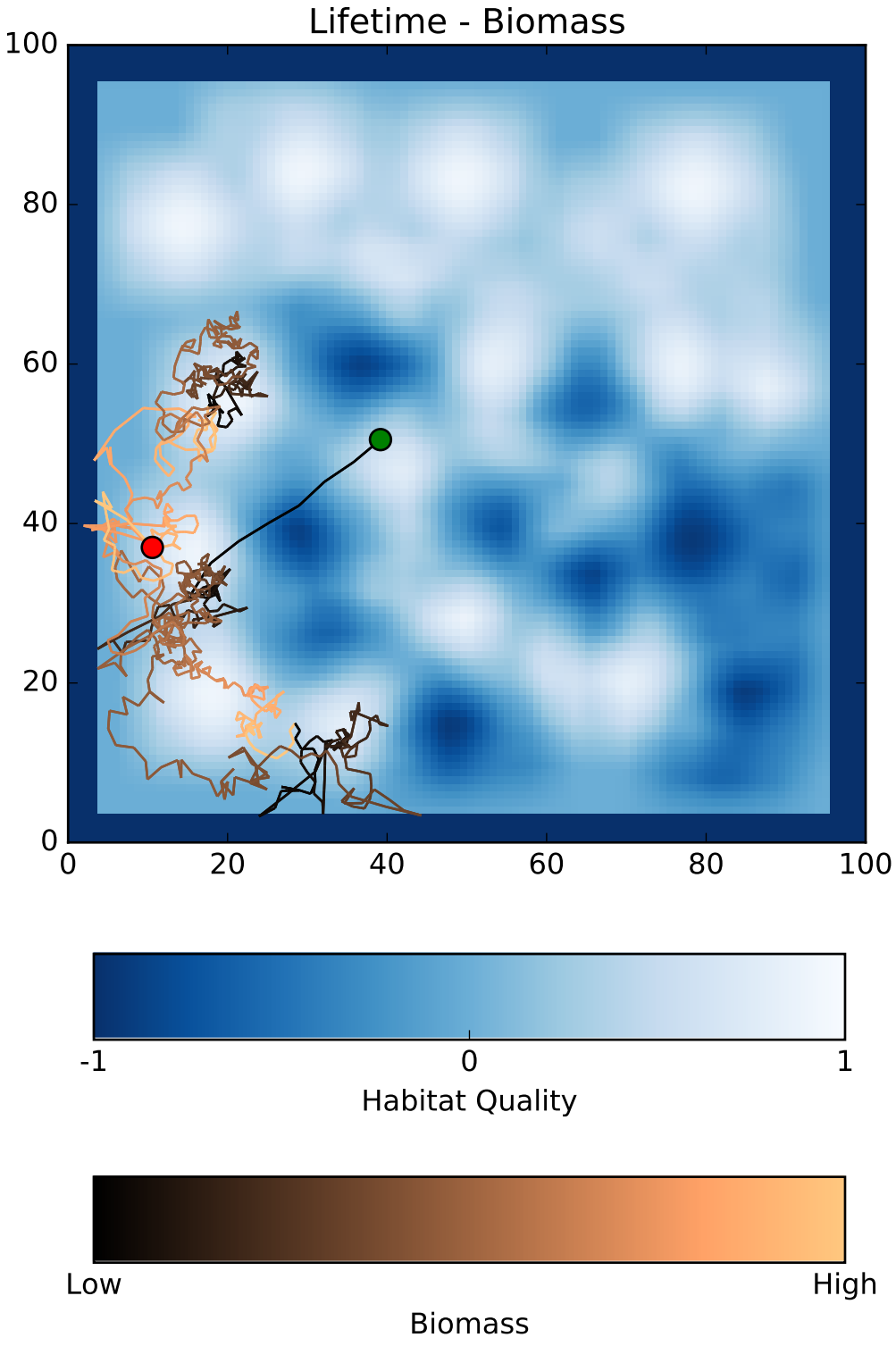

Figure 4.2: Example of visualizing a grid and the biomass over time using pytracks. The start and end points of the track are shown with a green and red dot respectively.

4.4 Heat-map

This example looks at the data in an Eulerian point of view. Below we create a graph which shows four snapshots of the overall biomass distribution at specific ticks of the simulation.

4.4.1 Initialization Code

As before, most of the code to initialize and prepare for processing is the same:

```
grid_wrapper = pytracks.input.GridWrapper("HMRC_100/grid.out", extra_ids=[3, 4])
tracks_wrapper = pytracks.input.TrackWrapper("HMRC_100/HMRC_20.out", id_column=2,
                 x_column=5, y_column=7, extra_ids=[10, 11])

grid = grid_wrapper.gen_grid()
trackset = tracks_wrapper.gen_trackset()

figure, axlist = plot.subplots(nrows=2, ncols=2, sharex="col", sharey="row",
                 figsize=(6, 7))
```

4.4.2 Initializing a Array for Data

This code creates a length 4 array which will allow us to create maps for each plot. In this code we also prepare the titles and select the specific ticks we want to look at.

```
titles = ["Biomass - Tick ", "Biomass - Tick ", "Biomass - Tick ", "Biomass - Tick "]
plots_data = numpy.array([numpy.zeros(grid.size) for _ in range(4)])
tl = len(trackset[0]) - 1
ticks = [0, int(round(tl * 0.33)), int(round(tl * 0.66)), tl]
```

4.4.3 Calculations

This code calculates the biomass density for each cell and stores it in the variables we initialized earlier. We also set the titles to the tick number.

```
for i in range(4):
    for track in trackset:
        x_coord = math.floor(track.x[ticks[i]]/100)
        y_coord = math.floor(track.y[ticks[i]]/100)
        plots_data[i][y_coord - 1][x_coord - 1] += track.biomasses[ticks[i]]
    titles[i] += str(ticks[i])
```

4.4.4 Visualization Code

The below code has a line which finds the maximum biomass of every cell, to assure that the colorbar range is valid for each plot. To make it so the lower concentrations are visible with the color, we choose the 99.75 percentile, so the extreme top end values are excluded. Other than that, typical matplotlib code is below.

```
max_biomass = int(round(numpy.percentile(plots_data.flatten(), 99.75)))

for axis, plot_data, title in zip(axlist.flat, plots_data, titles):
    axis.set_xlim([0, 25])
    axis.set_ylim([0, 25])
    axis.set_title(title)
    cbar = axis.imshow(plot_data, interpolation='none', origin="lower",
           cmap=plot.get_cmap("afmhot"), vmin=0, vmax=max_biomass,
           extent=[0, 25, 0, 25], aspect="equal")

figure.subplots_adjust(bottom=0.235)
colorbar_axis = figure.add_axes([0.15, .12, .73, .05])
```

```
colorbar = plot.colorbar(cbar, cax=colorbar_axis, orientation="horizontal")
colorbar.set_ticks([0, max_biomass])
colorbar.set_ticklabels(["Low", "High"])
colorbar.set_label("Biomass Concentration")


plot.savefig("export/ticks_heatmap.pdf", bbox_inches='tight', transparent=True)
plot.show()
```

4.4.5 Output

Output from this example can be seen in Figure 4.3.

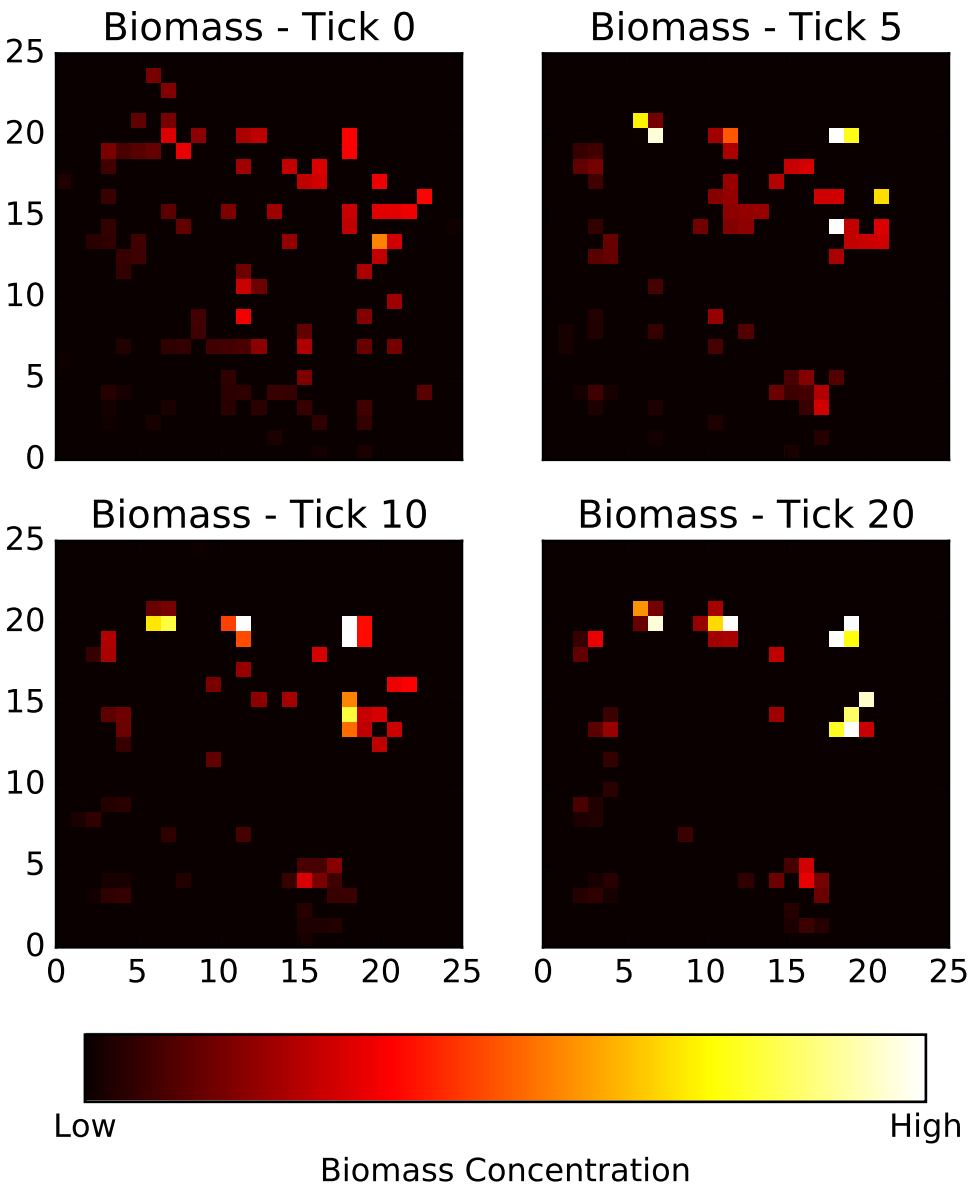

Figure 4.3: Example of visualizing the biomass density in the grid at specific ticks.

Chapter 5

Analysis

5.1 Introduction

As described earlier, properly simulating the movement of organisms is an important topic and component of modeling. For each individual to decide where to move next, an algorithm needs to be used. There are many different algorithms used now. Some have been developed for other applications and adapted for fish movement (K. A. Rose, Sable, et al. 2015). Others have been specifically developed for fish movement models. Either way, the goal is to find an algorithm which is the most effective. Often this can not be determined just by looking at tracks graphed on a grid and having a person analyze them. One may need to analyze the tracks in other ways to discover the differences between the movement algorithms.

The data for this analysis was taken from the runs which used the 25x25 cell grid.

5.2 Algorithms

In the paper (Watkins & Rose 2014) five movement algorithms were used. Their names are as follows, in no particular order:

1) Restricted-area Search
2) Kinesis
3) Event based
4) Run and Tumble (RT)
5) HMRC

Each movement algorithm is described in more detail in the (Watkins & Rose 2014) paper.

Each algorithm uses a different method to attempt to find favorable conditions for the individual. In this analysis we will see if and how these algorithms differ.

5.3 Population Success

The metric which shows how well a population did overall is the total biomass of the population. For each movement algorithm, the total biomass is shown in Figure 5.1.

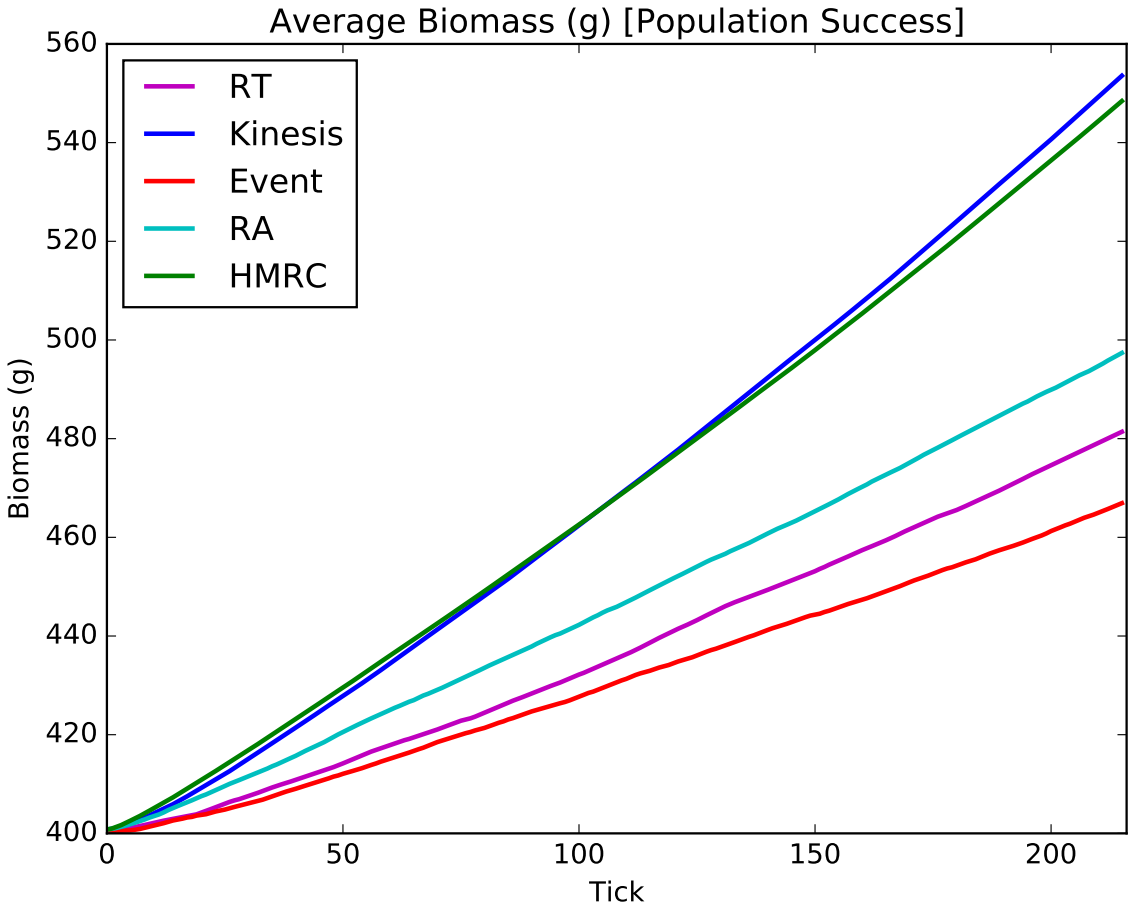

Figure 5.1: The total biomass of each movement algorithm's population over the entire run.

The populations are slightly different in their outcome, but some populations are too close to each other to fully see if they are different. Now we need to look at the energistics of the algorithms to show the differences more clearly.

5.4 Energy Input

In this model and these various movement algorithms, we can define energy input as the growth rate experienced (K. A. Rose, Fiechter, et al. 2015). For each movement algorithm, Figure 5.2 shows the average growth rate experienced. As specified before, the grid in which this model was ran on is made up of cells which each contain a growth and mortality factor. Each factor affects how well the individual will do while it is in that particular cell. The algorithms are designed to find the cells which have the best growth factor.

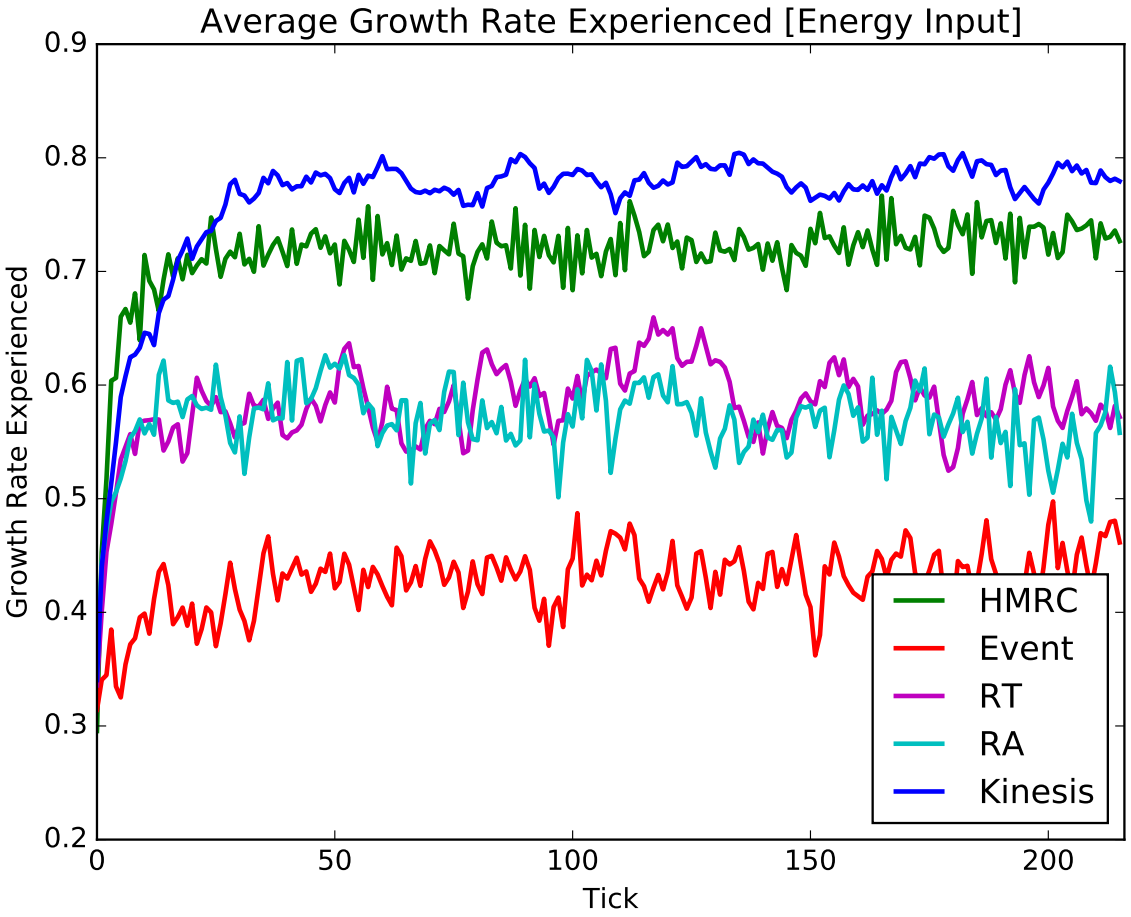

Figure 5.2: The average growth rate experienced for each tick of each movement algorithm's population over the entire run.

As we can relate the growth factor to the overall quality of the algorithm, the analysis of the energy input is relevant. We can see that the Kinesis algorithm performs the best at finding and staying in cells with a high growth factor. On the opposite side, we see that

Kinesis does a poor job at finding and staying in cells with a high growth factor. This confirms out findings in our first analysis of biomass, which shows that total biomass and the growth factor experienced by the individual is directly related.

5.5 Energy Output

Energy output in this model can be defined as the distance traveled. This is shown in Figure 5.3. This makes sense, as it takes energy to travel and on average the more time an individual spends moving, the more energy they will expend (Rose et al. 2010).

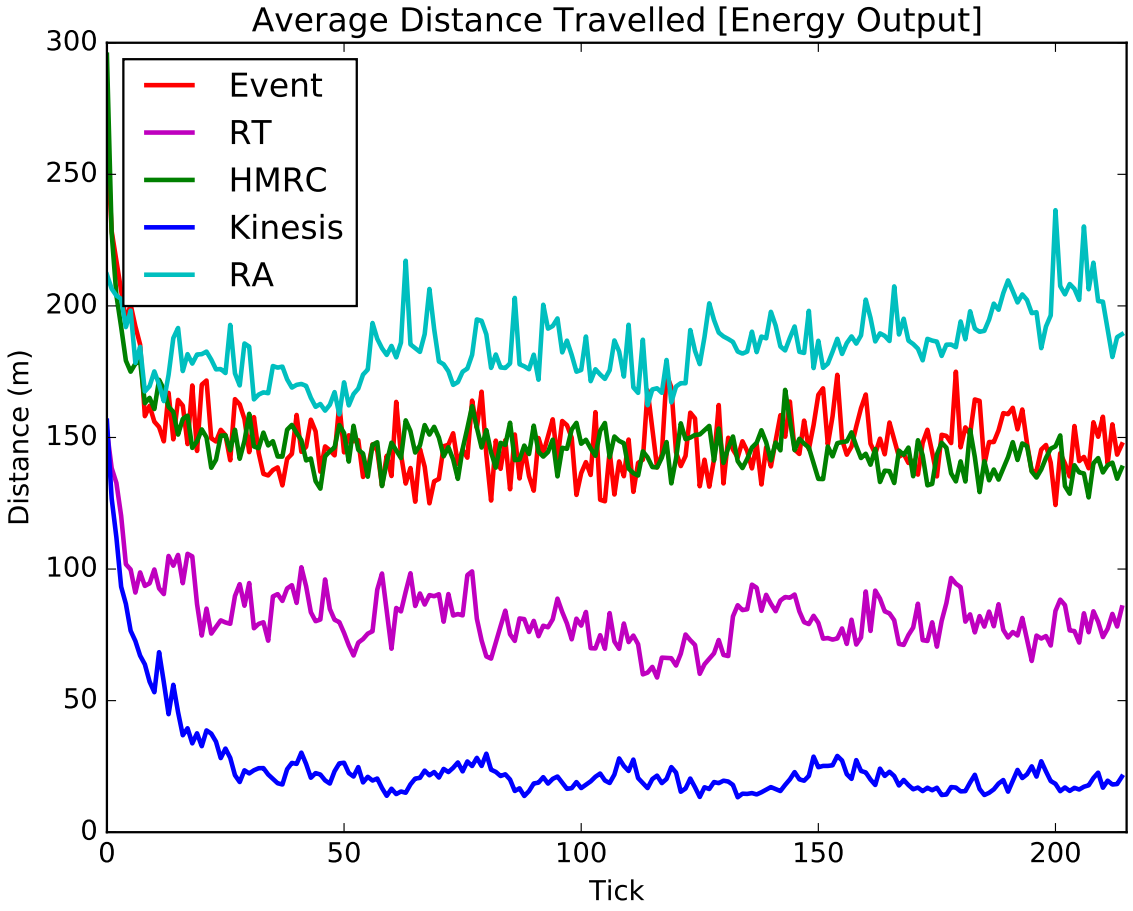

Figure 5.3: The total distance traveled by tick for each movement algorithm's population over the entire run.

We can see that Kinesis very quickly finds cells that it likes, further proving its adeptness as an excellent movement algorithm. Restricted-area search spends a lot of time wandering

around though, as demonstrated in Figure 5.4. It also will more often than not decide that the cell that it currently inhabits is not good enough, so it endlessly searches for cells with a more favorable environment, unlike Kinesis.

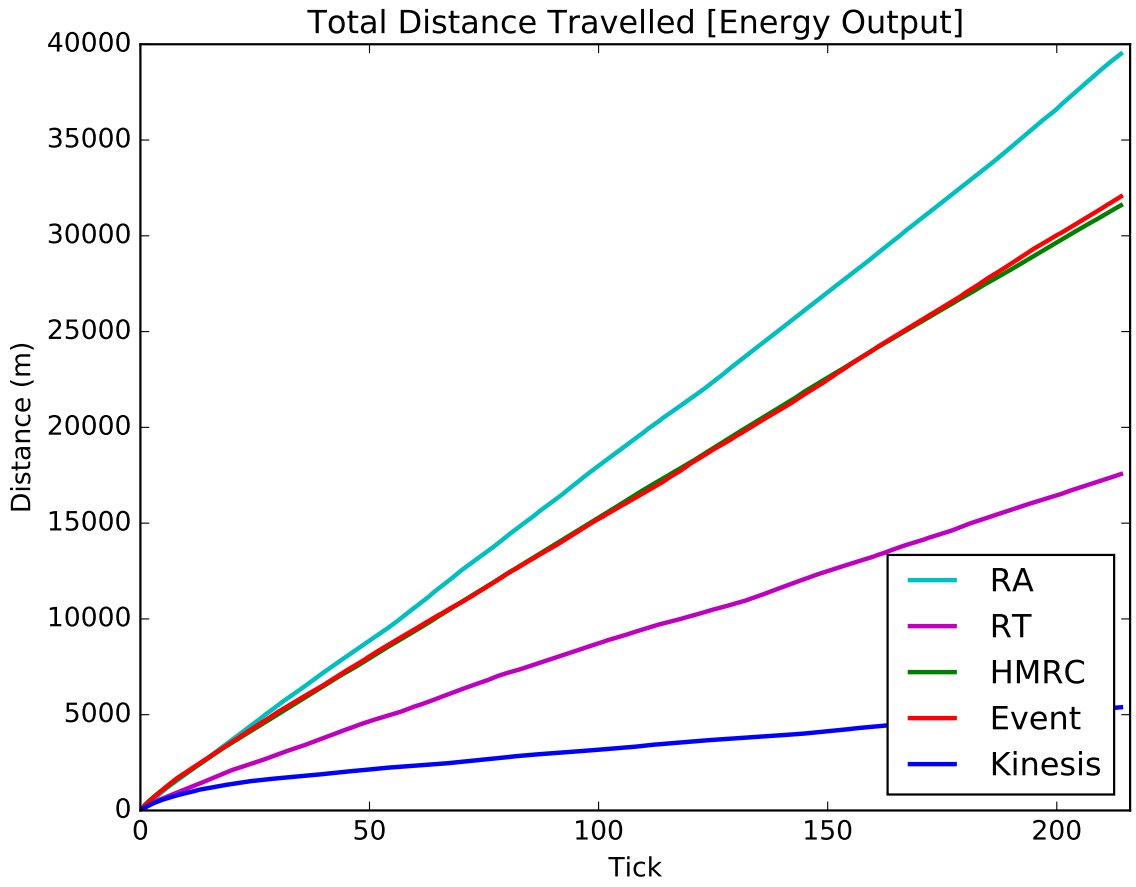

Figure 5.4: The average distance traveled by tick for each movement algorithm's population over the entire run.

5.6 Energy Efficiency

Efficiency in this model is defined by the average biomass of each tick divided by the average distance traveled by the individuals in that tick. This gives us a metric which shows how efficient with energy each movement algorithm was.

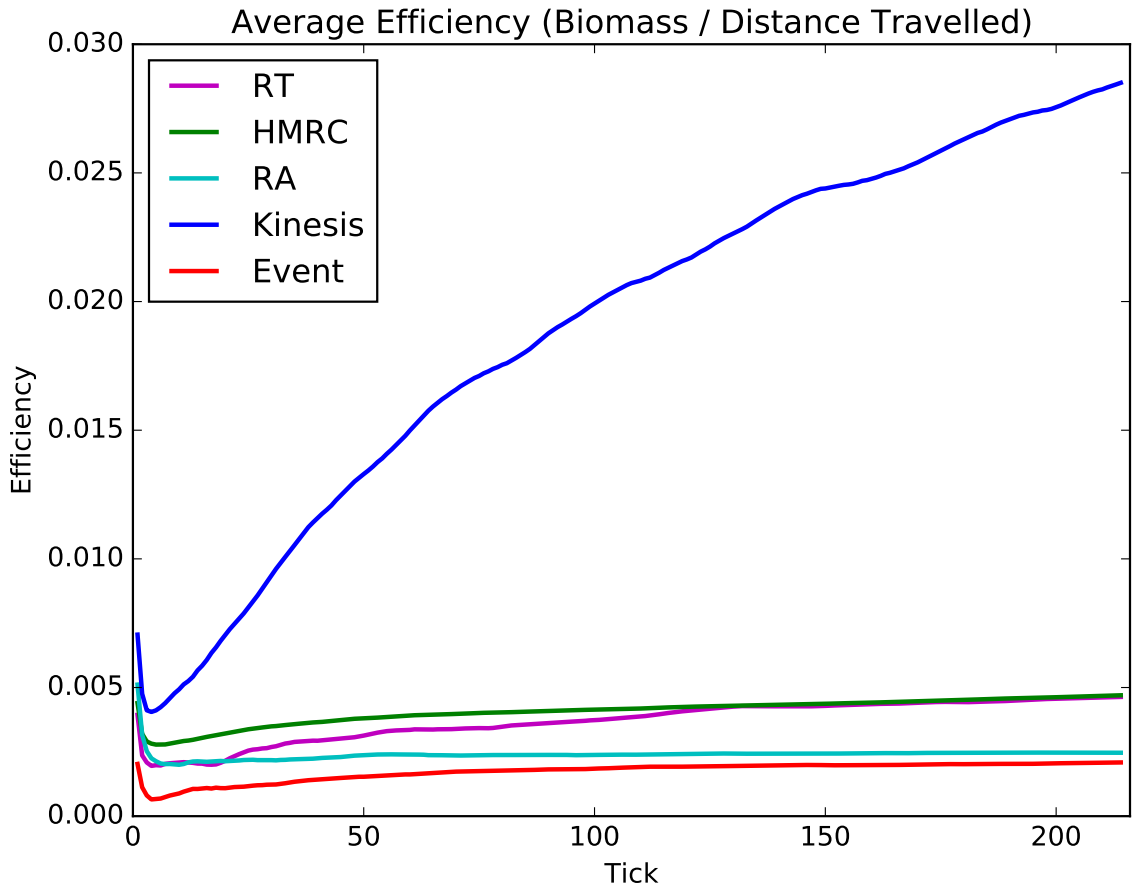

Figure 5.5: A metric which shows us how efficient each movement algorithm is.

As the reader can see, Kinesis once again is exceptional. It has the ability to use the least energy possible to achieve a particular goal biomass. This is due to its excellent path finding movement algorithm, which reflects in the total distance traveled. Once the Kinesis algorithm finds a particular cell it likes, it tends to stick there. This works well for grids with multiple good areas, but what happens if the grid is different?

5.7 Verdict

As we have shown above, Kinesis seems to be the best movement algorithm. As questioned before though, is it really the best overall? Kinesis may possibly not be the best movement algorithm for other grids due to it's safe style of searching. It's characteristic of staying in a cell once it finds a cell which is deemed good enough may be detrimental in other situations. To determine if that is the case, further analysis of other grids can be done. That was not done so in this analysis, as this requires much more data than is available to come to a solid conclusion.

Every movement algorithm is different and will perform differently when put on a different grid (Rose 2012). Another movement algorithm might out-perform Kinesis on a different grid. Many different grids would need to be analyzed to see which movement algorithm performs best or most life-like overall. This is why the post analysis, which pytracks attempts to help with, needs to be as easy to possible.

Chapter 6

Conclusion

6.1 Summary

Pytracks, as presented in this thesis, is a tool which took a lot of learning to create. It took a lot of research to be able to create a tool which was both easy to use, and thus simple, but also intricate and expandable. The requirements specified in the methods were achieved. It allowed the users to write code which visualizes data which is much shorter and easier to read and interpret than without it.

The major limit of pytracks is its scope of usability is quite small in the grand scheme of things. The problem is that for something to be effective at what it does, it has to be specific and tailored to solving one problem. The questions that were attempted to answer were centered around using only two dimensional grids, which is what pytracks is limited to at the moment. Pytracks has the potential to solve many other problems, but that takes extra effort by another developer to implement solutions for that.

After using several examples and answering a research question, pytracks has been shown to be effective in solving the problem at hand. It was able to allow developers to create visualizations much easier than if they were just trying to use the file without this library. The ultimate goal of pytracks was to let it do the work of wrapping these raw data files,

which saves a developer's time and energy. All this leaves is the visualization work left for the user to do. As shown above, pytracks fills this gap, which means it achieved its goal.

6.2 Future Work

The question asked in this thesis and the tool developed for it can be expanded further in multiple ways. First, as I feel examples are the most useful for other users to learn how to use a tool, more could be written to demonstrate the usefulness of the tool. Second, more built in flexibility for reading and inputting the raw data input files and processing methods could be developed as well. This is where many people could contribute a library of their own methods to the pytracks code-base, which then simply just increases pytracks's usefulness. Lastly, the library could be expanded to actually visualize the data and not just wrap it. This is a good idea, but it also shifts pytracks from being a library and development tool to an application. This way may not be a good direction for pytracks. Currently pytracks is only accessible as a library and the user must program to use it. This keeps pytracks flexible, which was the goal from the beginning. An alternative to this is to keep pytracks as a library, but an application could be developed which wraps it and makes specific visualizations and calculations easier to the end user. This approach keeps the pytracks library flexible, and also makes it more accessible to more users. This hypothetical application may not require the user to program at all, which greatly expands its potential user-base.

Appendix 1: Source Code

The source code for pytracks can be found on Github: https://github.com/rosspf/pytracks

To run the code, a user can learn from the examples listed in this paper or look at the Quick Start Guide in the additional online documentation. A link to the online documentation is in Appendix 2.

Appendix 2: Further Documentation

Sphinx 1.3.1 (http://sphinx-doc.org/) was used to to document pytracks further. The HTML based documentation can be found here: http://pytracks.readthedocs.org/en/latest/

References

Barbaro, A., Einarsson, B. & Sigurosson, S., 2009. Modelling and simulations of the migration of pelagic fish. *ICES J. Mar. Sci.*, 66(5), pp.826–838.

Campbell, M.D. et al., 2011. Individual-based modeling of an artificial reef fish community: Effects of habitat quantity and degree of refuge. *Ecological Modelling*, 222(23-24), pp.3895–3909.

Creekmore, S.B., 2011. Modeling the population effects of hypoxia on Atlantic croaker (Micropogonias undulatus) in the northwestern Gulf of Mexico. *LSU Masters Thesis*, (August), p.73.

Crouse, D., Crowder, L. & Caswell, H., 1987. A Stage-Based Population Model for Loggerhead Sea Turtles and Implications for Conservation. *Ecology*, 68(5), pp.1412–1423.

Diamond, S.L., Murphy, C.a. & Rose, K.a., 2013. Simulating the effects of global climate change on Atlantic croaker population dynamics in the mid-Atlantic Region. *Ecological Modelling*, 264, pp.98–114.

Gu, S., 2010. Component-Based Isosurface Extraction for Multiple Dataset Visualization. *Journal of Winter School of Computer Graphics (WSCG)*, 18, pp.97–104.

Hussey, N.E. et al., 2015. Aquatic animal telemetry: A panoramic window into the underwater world. *Science (New York, N.Y.)*, 348(6240), p.1221.

Intel IT Center, 2013. Big Data Visualization : Turning Big Data Into Big Insights. *White Paper on Big Data Visualization [White Paper]*, (March).

Khurana, S. et al., 2012. Multi Scale Color Coding of Fluid Flow Mixing Indicators along Integration Lines. *Proceeding of: WSCG2012 - 20-th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision, June 25-28, 2012, Plzen, Czech Republic*.

Kumar, R., 2015. Future For Scientific Computing Using Python. *Conference: International Journal of Engineering Technologies and Management*, 2(July), pp.30–41.

Rose, K.A., 2012. End-to-end models for marine ecosystems: Are we on the precipice of a

significant advance or just putting lipstick on a pig? *Scientia Marina*, 76(1), pp.195–201.

Rose, K.A. et al., 2010. End-To-End Models for the Analysis of Marine Ecosystems: Challenges, Issues, and Next Steps. *Marine and Coastal Fisheries*, 2(1), pp.115–130.

Rose, K.A. et al., 2015. Demonstration of a fully-coupled end-to-end model for small pelagic fish using sardine and anchovy in the California Current. *Progress in Oceanography*.

Rose, K.A. et al., 2015. Proposed best modeling practices for assessing the effects of ecosystem restoration on fish. *Ecological Modelling*, 300, pp.12–29.

SAS, 2012. Data Visualization : Making Big Data Approachable and Valuable (white paper)., (August), pp.1–4.

Watkins, K.S. & Rose, K.A., 2014. The Effects of Spatial and Temporal Resolution in Simulating Fish Movement in Individual-Based Models. *Transactions of the American Fisheries Society*, 143(5), pp.1143–1160.

Xu, Y. et al., 2013. Environmental influences on the interannual variation and spatial distribution of Peruvian anchovy (Engraulis ringens) population dynamics from 1991 to 2007: A three-dimensional modeling study. *Ecological Modelling*, 264, pp.64–82.

Yemane, D., Shin, Y.-j. & Field, J.G., 2009. Exploring the effect of Marine Protected Areas on the dynamics of fish communities in the southern Benguela: an individual-based modelling approach. *ICES Journal of Marine Science*, 66(1995), pp.378–387.

Vita

Ross Fossum was born in Houston, Texas. He then moved to New Orleans, Louisiana and after Katrina moved to Baton Rouge, Louisiana. He then went to Mckinley Senior High School in Baton Rouge. After high school, he entered the Coastal Environmental Science Program at Louisiana State University. After graduating with a bachelors of Coastal Environmental Science, he then entered the Graduate School at Louisiana State University to pursue a Masters of Computer Science in Louisiana State University's Division of Computer Science and Engineering.