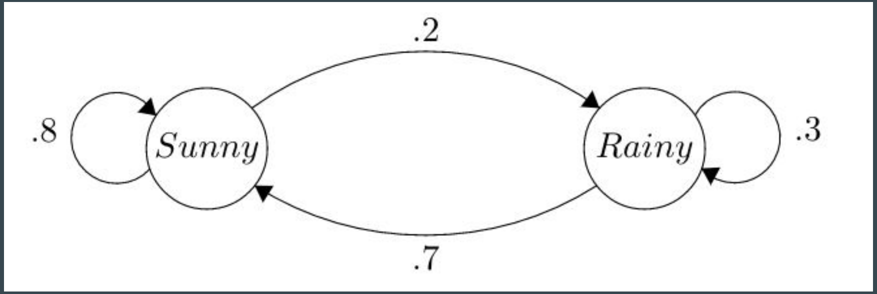
Evolutionary Optimization: Markov Chain Stationary Vector System

March 5th, 2020

Maddox Johnston, Colin Frazier, Jolie Even
College of Charleston

Markov Chain

A **Markov Chain** is a stochastic model that describes a sequence of events where the probability of going to a particular state depends solely on the state of the previous event.



Each Markov Chain can be represented by a **transition probability matrix P.**

Markov Chain

Conditions for P:

- All entries must be between 0 and 1.
- Rows must sum to 1.
- Must be a square matrix.

Two states a and b are said to **communicate** if it’s possible to get from a to b and it’s possible to get from b to a. A **communicating class** is a set of states that all communicate, so it’s possible to start in one state and get to any other state in finite time.

Stationary Distribution

We want to find the stationary distribution π of a given Markov process, such that $\pi = P\pi$, where π is a (1, n) row vector, and P is a square (nxn) probability transition matrix.

We note that π is a unique stationary vector if P is irreducible (one closed communicating class).

Algorithmic Music Composition

- Uses software to implement Markov Chains and generate music
  - Csound, Max, and SuperCollider
  - When at a certain note, which next notes will sound best?
- As the generated song progresses, what notes should we expect to appear most often? The vector π will answer this.

$$\begin{array}{c|ccc} & A & C\# & B\flat \\ \hline A & 0.1 & 0.6 & 0.3 \\ C\# & 0.25 & 0.05 & 0.7 \\ B\flat & 0.7 & 0.3 & 0 \end{array}$$

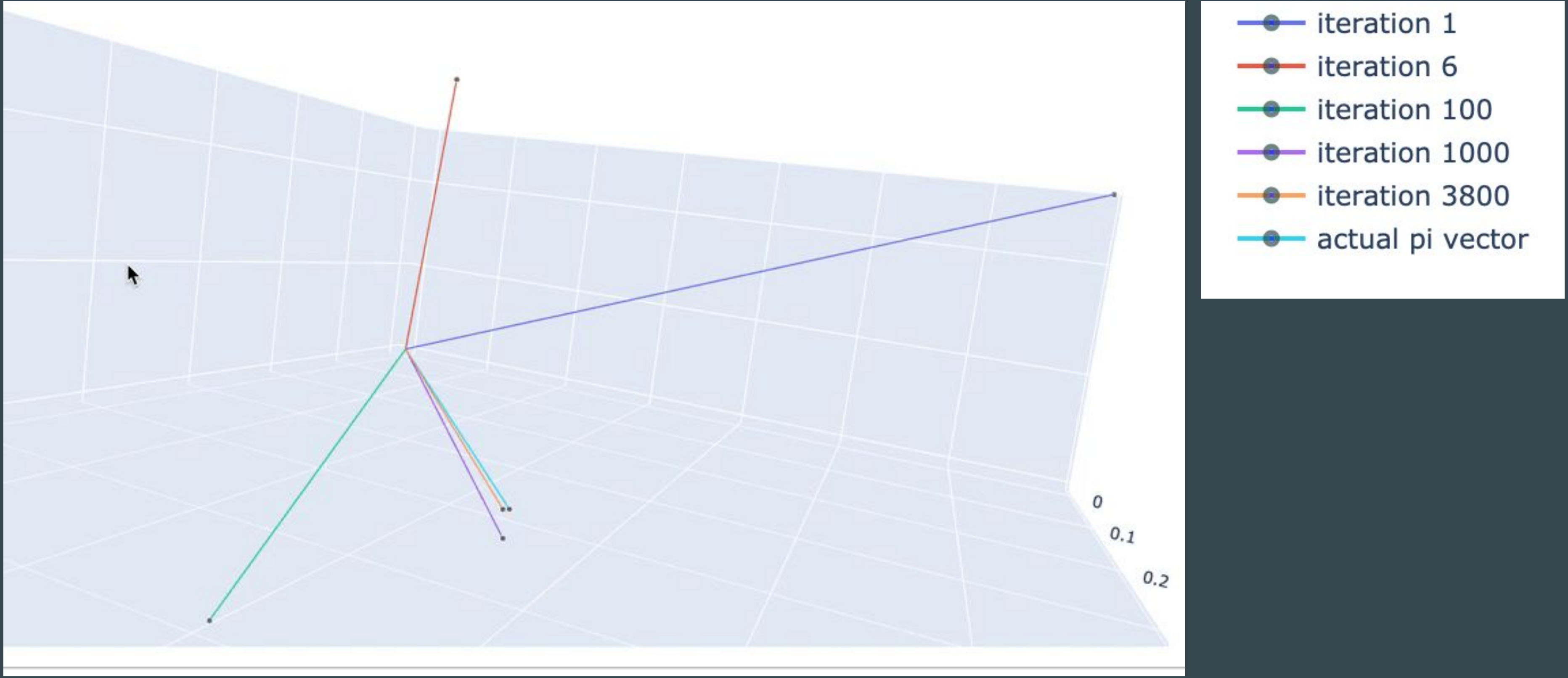
Fitness Function and Stopping Criteria

Our fitness function can be understood the norm of (π - Pπ), with a precise solution when F=0., i.e. the norm is minimized.

$$\mathrm{F} = \left|\left|\pi - P\pi\right|\right|_2$$

$$= \sqrt{\left(\sum_{i=1}^{n} (\pi_i - P\pi_i)^2\right)}$$

Visual Representation: Approaching Stationary Vector



Mating and Mutating Algorithms

***Weighted Addition:** Add, and divide by sum

$$[0.4, 0.6] + [0.1, 0.9]$$
$$= [0.5, 1.5] \rightarrow [.25, .75]$$

Stochastic Weighted "Subtraction": Subtract, take absolute value of resultant, and divide by sum

$$[0.4, 0.6] - [0.1, 0.9]$$
$$= [|0.3, -0.3|] \rightarrow [0.5, 0.5]$$

Weighted "Multiplication": Inner product, and divide by sum

$$[0.4, 0.6] \cdot [0.1, 0.9]$$
$$= [0.4, 0.54] \rightarrow [0.425, 0.575]$$

Flip: Reverse the vector

$$[0.2, 0.5, 0.3] \rightarrow [0.3, 0.5, 0.2]$$

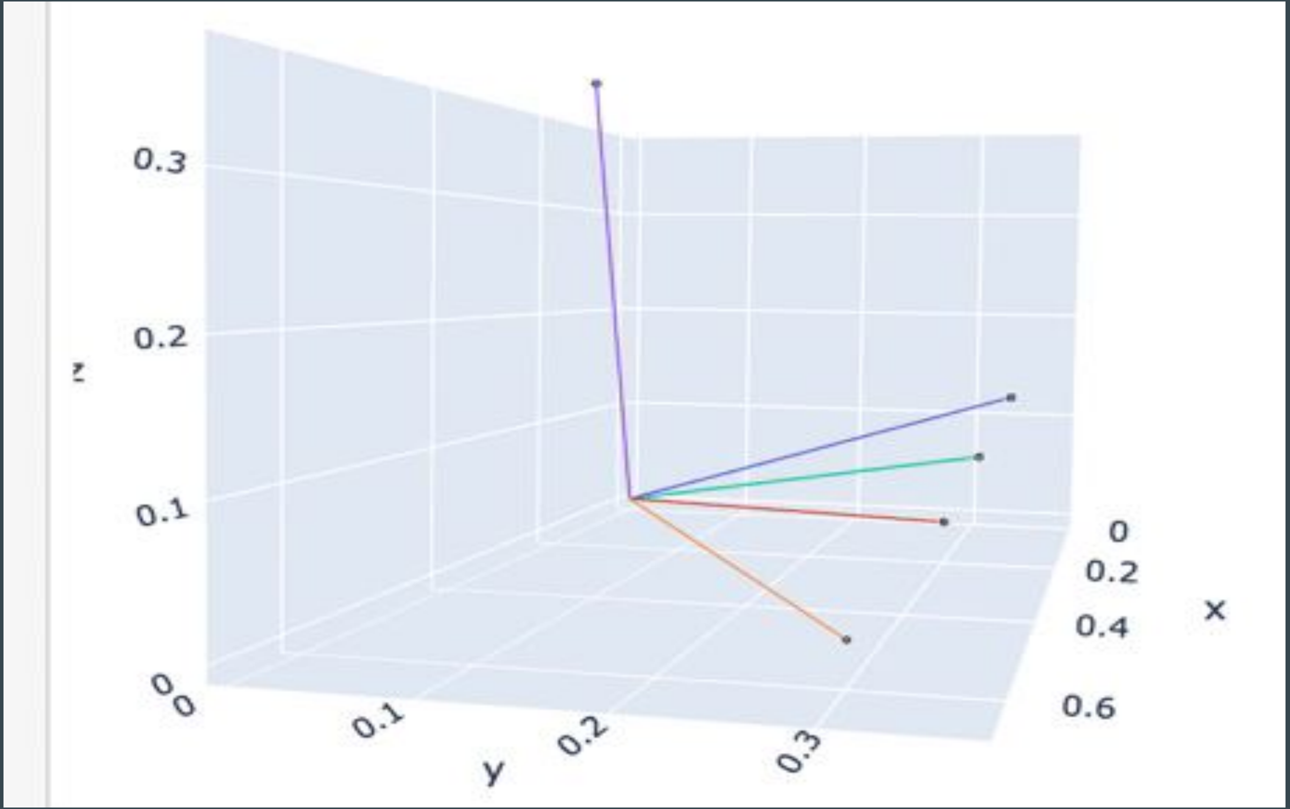
Index Swap: Swap two indices

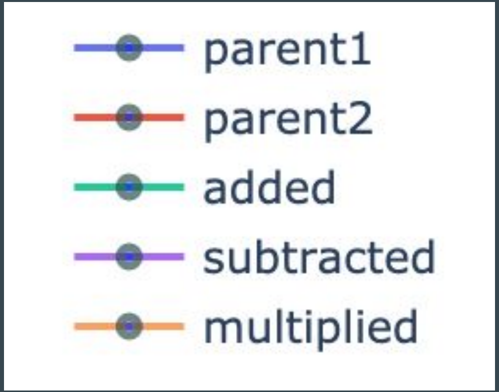
$$[0.2, 0.5, 0.3] \rightarrow [0.5, 0.2, 0.3]$$

***Random Stochastic Shuffle:** Replace with a new random, stochastic vector

$$[0.2, 0.5, 0.3] \rightarrow [0.43, 0.24, 0.33]$$

Visual Representation of Mating Algorithms





Methods

Through testing with different mutation and mating splits, we noticed that certain splits performed better as the size of our vector changed.

In general, as **n increased**, a solution could be found quicker if **more mating algorithms** were applied.

It often proved beneficial to **mutate** the **illest fit parent**, and to replace an ill fit parent with the **mate** of an **ill-fit parent** and a **well-fit parent**.

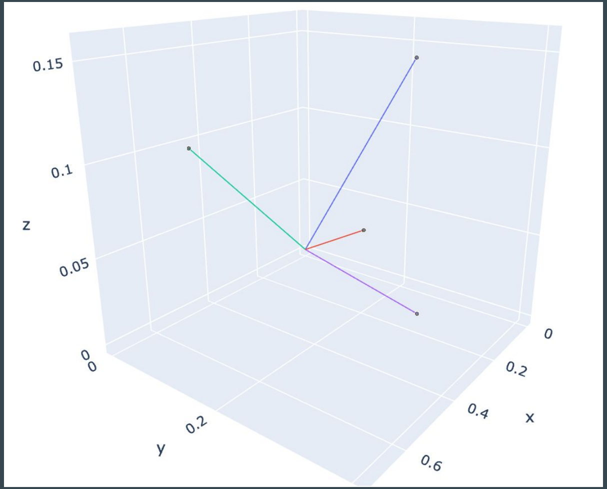
Implementation

We implemented the code using both Python and R. Despite slightly different techniques, we noticed strikingly similar trends:

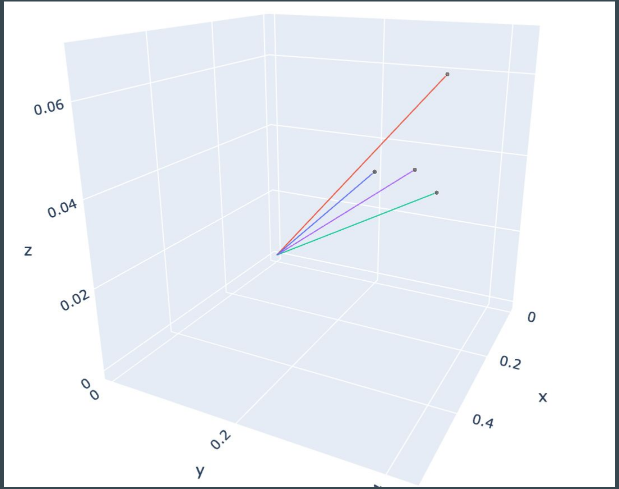
-The first hundred or so iterations would bring us very close to the solution, quite quickly and efficiently.

-If the program did not solve π within the first 0-200 iterations, it would often take many more thousands of iterations, with little variability between the two.

3 Parent Vectors and π (purple) after 3 iterations, t= ~25ms



3 Parent Vectors and π (purple) after 100 iterations t= ~800ms.



Pseudo Code (Pre-Algorithm)

Get P and population vectors.

Define fitness function and mating/mutating algorithms.

Determine tolerance.

Determine mating/mutating percentage.

Pseudo Code (Running the Algorithm)

Generate a random number between 1 and 100. This determines whether to use a mating algorithm or mutating algorithm.

Generate a random number between 1 and 3. This determines which mating or mutation algorithm to use.

In either case if the offspring is better, we'll replace the offspring with the (worst fitting) parent in our population.

If new offspring is not within the predetermined tolerance, we'll go through the algorithm again.

Results

- Solved for π in the music example
- Tolerance: F < 0.01
- π=[0.347, 0.323, 0.33]
- Number of iterations: n=83
- Run time: < 1 second

$$\begin{array}{c|ccc} & A & C\# & B\flat \\ A & 0.1 & 0.6 & 0.3 \\ C\# & 0.25 & 0.05 & 0.7 \\ B\flat & 0.7 & 0.3 & 0 \end{array}$$

Future Work and Improvements

- Mean run times/iterations for different percentages of mating/mutating.
- Mean run times/iterations for greater n.

References

- Karlin, Samuel, and Howard M. Taylor. *A First Course in Stochastic Processes.* Acad. Press, 2011.
- K McAlpine; E Miranda; S Hoggar (1999). "Making Music with Algorithms: A Case-Study System". *Computer Music Journal.*

Thank You.
Any Questions?