# Leaf Growth Pattern 

Raymond Yap<br>Casa di Mir Montessori School<br>Campbell, CA 95008, USA<br>mr.raymint@gmail.com


#### Abstract

Taking a walk around our neighborhood or in the park on a sunny day, if we look around, we will see lots of flowers and plants. So, we may wonder: "How is it that plants can maximize their photosynthesis?" In other words, "How can the leaves be best laid out to obtain the most amount of sunlight?". Intuitively, the plant with the leaves that are spread out most evenly would be the most efficient. Based on observations and research, different plants have their own constant leaf spacing angle. Utilizing MIT Scratch and Python, the author has programmed a "Leaf Calculator" to output standard deviation of leaf spacing given inputs of leaf rotation angle and leaf count. Iterations from rotation angle zero ( 0 ) to 180 degrees and leaf count one (1) to 30 were completed to produce a Tournament scoreboard, followed by a "Find Winner" program to discover the winner with the lowest cumulative standard deviation. The winner is the plant in which the leaf rotation angle grows such that the pattern is the most evenly spread out. A "Grow" program was also written to visualize the leaf growth at any angle and count. In the future, plants may be genetically engineered for maximum photosynthesis. Surprisingly, this work is related to the Golden Ratio.


## Keywords

Photosynthesis, Leaf Growth, Pattern Detection, MIT Scratch, Python, Standard Deviation.

## 1. Objective

### 1.1 Introduction

When we look at the question "How can plants maximize photosynthesis?" we can think about "what makes efficiency in plants?" The author did a little bit of research and found something called Phyllotaxis, which is the study of leaf arrangement in Botany. In Phyllotaxis, there are four types of leaf arrangements. One of them is a pattern with two leaves left and right, and then on top, another two and so on, as seen in arrangement i, Figure 1. The second type, is two leaves front and back, and then on top, two leaves left and right, and then on top, two leaves front and back, and so on, as seen in arrangement ii, Figure 1. The third type is a set of leaves, and then on top another set of leaves and so on, as seen in arrangement iii, Figure 1. Now the fourth type is a type of leaf growth that is a constant leaf growth angle as seen in arrangement iv, Figure 1. This means all of the leaves grow at the same angle.


Figure 1. Phyllotaxis: Illustration of four basic types of leaf arrangements in plants. Phyllotactic spirals form a distinctive class in nature. They can be found in many varieties of plants.

### 1.2 Scope

This project is focusing on type iv of leaf arrangement in Figure 1. We want to find out "What angle makes the leaves the most evenly spread out?" or "At what angle should the leaves grow to make them the most evenly spread out?

### 1.3 Tournament

Think of this like a tournament, where all leaf angles 0 through 180 degrees are competing to see which one is the most efficient. There are 30 rounds in this tournament, representing 30 leaf count. So, the winner of each round is the angle that is the most efficient for that leaf count. In this case, the most efficient angle for each round is determined by which angle has the lowest spacing deviation, which we have quantified using the standard deviation statistic. The standard deviation is a typical measure of variation within a "sample," and is used to represent a "population" which is normally distributed. We take this as a reasonable assumption because it is well known that so many phenomena quantified in nature follow a normal distribution. The lower the standard deviation, the more evenly spread out the leaves will be, because there will be less spacing variation relative to each other.

### 1.4 Scoreboard

Since hand calculating the standard deviation of all these angles and for all these leaves would be far too tedious, the author created programs in Python code and MIT Scratch. Python was used to create a scoreboard with all the scores of standard deviations on it, and MIT Scratch was used to create the data visualizations.

## 2. Procedure

The following steps are used to calculate the standard deviation of one angle for a certain quantity of leaves. Where indicated, computer program scripts were written to aid in the process:

1. Draw a leaf at starting position 0 degrees ( 12 o’clock) as shown in Figure 2.
2. Pick a constant rotation angle. Example: 123 degrees.
3. Draw the next leaf 123 degrees apart from the current leaf.
4. Mark the tip of each leaf with position (in degrees) and leaf number.


Figure 2. Leaf growth at constant rotation angle 123 degrees. The first two leaves are shown here, one at 0 degrees and the other at 123 degrees.
5. Repeat step 3-4 for the desired number of leaves. Example: Five.
6. Create a table to list leaf number and leaf position. See Figure 3.


Figure 3. Leaf number and position. It shows the angle in degrees in which each leaf is located.
7. Order leaf position from the least to the greatest.


Figure 4. Ordered leaf position, sorted in ascending order.
8. Measure the spacing between all adjacent leaves.


| loaf count | 1 | 2 | 3 | 4 | 5 |
| :--- | :---: | :---: | :---: | :---: | :---: |
| leaf xpaciner | 9 | 114 | 9 | 114 | 114 |

Figure 5. Leaf spacing. It shows the difference in degrees between the ordered angles.
9. Calculate the standard deviation of leaf spacing.

Table 1. Leaf spacing standard deviation. The detailed steps to calculate mean spacing of data set in Figure 5, the deviation from the mean, and standard deviation.

| Mean spocing | $(9+114+9+114+114) \div 5=72$ |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: |
| deviation from the mean | $\begin{gathered} 9-72= \\ -63 \end{gathered}$ | $\begin{gathered} 114-72= \\ 42 \end{gathered}$ | $\begin{aligned} & 9-72= \\ & -63 \end{aligned}$ | $\begin{aligned} & 114-72= \\ & 42 \end{aligned}$ | $\begin{aligned} & 114-72= \\ & 42 \end{aligned}$ |
| stomdard deviation | $\sqrt{\frac{(-63)^{2}+42^{2}+(.63)^{2}+42^{2}+42^{2}}{5}}=51.44$ |  |  |  |  |

10. Write a computer program to repeat step five (5) to nine (9) for leaf count one (1) to 30, leaf rotation angle zero (0) to 180 degrees, to generate a standard deviation table for 30 leaves

Table 2. Standard deviation scores. This table stores standard deviation calculated for a given leaf rotation angle and a given leaf count.

| rangle count | 1 | 2 | 3 | 4 | 5 | 6 | $\cdots$ | 29 | 30 |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| 0 |  |  |  |  |  |  | $\cdots$ |  |  |
| 1 |  |  |  |  |  |  | $\cdots$ |  |  |
| $\cdots$ |  |  |  |  |  |  | $\cdots$ |  |  |
| 123 | 0 | 57 | 4.24 | 46,4 | 51.44 |  | $\cdots$ |  |  |
| $\cdots$ |  |  |  |  |  |  | $\cdots$ |  |  |
| 180 |  |  |  |  |  |  | $\cdots$ |  |  |

11. Think of a sports tournament with competition rounds from one (1) to 30. Each round is each new leaf growth event (leaf count). Write a "Cumulative Score" program (Python) to create a scoreboard of cumulative standard deviation as leaf count increases.

Table 3. Cumulative scores. Based on Table 2, in this table, for each row (the leaf rotation angle), each column (leaf count) contains the sum of all the standard deviation scores from leaf count 1 to the current column (current leaf count).

| angle count | 1 | 2 | 3 | 4 | 5 | $\ldots$ | 30 |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| 0 |  |  |  |  |  | $\ldots$ |  |
| $\ldots$ | 0 | $0+57=$ <br> 57 | $57+4.24$ <br> 61.24 | $61.24+46,9$ <br> 108.14 | $108.14+51.44$ <br> $=159.58$ | $\ldots$ |  |
| 123 |  |  |  |  | 1 | $\cdots$ |  |
| $\ldots$ |  |  |  |  |  |  |  |

12. For each leaf count (each tournament), run the "Find Winner" program to find the lowest score, this is the most efficient rotation angle (the winner).

## 3. Results and Data Analysis

### 3.1 Standard Deviation Scores

The standard deviation table generated in step 10 produced a wide variety of winners (leaf rotation angles) at various tournaments (where leaf counts range from 1-30 per tournament). In other words, this method was unable to identify a clear winner(s)!

This result led the author to the concept of a cumulative scoreboard, wherein the intent was to improve the accuracy
and see the data converge on a specific winner. The winner for each tournament is the leaf rotation angle that produces the lowest sum of its leaf spacing standard deviation up to the tournament round (the leaf count).


Figure 6. Results of Standard Deviation scores from iteration of Step 10. For each tournament round (leaf count), find leaf rotation angle with the lowest standard deviation. These are the winners.

They are listed on the table and shown in the scatter plot.

### 3.2 Cumulative Scores

The cumulative standard deviation table (the cumulative score) generated in step 11 produced the top two winners at rotation angles of 137 and 138 degrees, winning 12 and 9 tournament rounds, respectively (Figure 7)

The scatter plot of winners appear to be converging around 137 and 138 degrees. Step 11 was performed again for rotation angles between 137 to 138 degrees in 0.1 degree resolution and once more at 0.001 degree resolution (See Figures 8 and 9)


| angle count | 1 | 2 | 3 | 4 | 5 | $\cdots$ | 30 |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| 0 |  |  |  |  | $\ldots$ |  |  |
| $\ldots$ |  |  |  |  |  | $\cdots$ |  |
| 123 | 0 | $0+57=$ <br> 57 | $57+4.24=$ <br> 61.24 | $61.24+46.9$ <br> -108.14 | $108.14+51.44$ <br> 159.58 | $\cdots$ |  |
| $\ldots$ |  |  |  | 1 | $\cdots$ |  |  |




Figure 7. Results of identifying top two winners from iteration of Step 11. For each tournament round (leaf count), find leaf rotation angle with the lowest cumulative standard deviation.

The top two winners winning multiple tournament round are 137 and 138.

Finer resolutions produced scatter plot with winner converging at approximately 137.5 degrees as leaf count increases. (Figure 8 and 9)


Figure 8. Winners between angle 137 and 138 at one tenth of inch resolution. As leaf count increases, the scatter plot is converging at approximately 137.5 degrees.

Proceedings of the International Conference on Industrial Engineering and Operations Management Bogota, Colombia, October 25-26, 2017


Figure 9. Winners between angle 137 and 138 at one thousandth of inch resolution. At this resolution it shows smoother convergence to approximately 137.5 degrees.

## 4. Discussion and Conclusion

The scatter plot of scoreboard for angles $0-180$ and leaf counts 1-30 surprisingly look very much like that of the Fibonacci series plot, shown in Figure 10 . The winning rotation angle converging at around 137.5 satisfies the equation:
$\mathrm{b} / \mathrm{a}=(\mathrm{a}+\mathrm{b}) / \mathrm{b}=\Phi$,
where $\mathrm{a}=137.5, \mathrm{~b}=222.5, \mathrm{a}+\mathrm{b}=360$.
Reverse calculation of $b=a+b / \Phi=360 / \Phi$,
with $\Phi=1.6180339887498948482045868343656$, yields $b=222.49223594996214535365126037163$.
Thus, $\mathrm{a}=360-\mathrm{b}=137.50776405003785464634873962837$ or rounded to the thousandth place, 137.508.
This should be the most efficient rotation angle for leaf growth.

© IEOM Society International

Figure 10. The cumulative score scatter plot resembles the Golden Ratio plot and the winning angle 137.5 degrees satisfies the Golden Ratio equation $b / a=(a+b) / b$.

## 5. Summary of Program Computations and Data Visualization Results

Visualization of all 30 leaf counts for the winning angle 137.508 degrees. is shown in Figure 11. It appears to be the most evenly spread out when compared to other nearby angles such as: 137, 137.25, 137.75, and 138 degrees.


Figure 11. Visualization of the winning angle 137.508 compared to $137,137.25,137.75,138$. It produces the most evenly spread out leaf arrangement.

In contrast, a leaf by leaf comparison for up to 20 leaf counts of a much less efficient angle of 123 degrees versus the winning angle of 137.508 degrees. shows the winning angle produces a more evenly spread out, and less overlapping, leaf arrangement which is therefore more efficient for absorbing sunlight.


Figure 12. Leaf growth comparison between angle 123 degrees versus 137.508 degrees. In this leaf by leaf growth, the winning angle 137.508 degrees shows minimal overlapping in every leaf count compared to 123 degrees which starts to have tight overlapping starting leaf count 4.

Below are the programs scripts (Figure 13) written in MIT Scratch for leaf pattern visualization and scoreboard calculations and colormaps. A similar set of programs was written in Python to generate the scoreboard tables (Figure 14).

Table 4. Program names, their inputs, their functional descriptions of what they do and their outputs.

| Program Name | Inputs | Description |
| :--- | :--- | :--- |
| Leaf Position | count, <br> angle | Returns a list of leaf positions in degrees. Procedure step 6. |
| Order It | leaves | Returns a sorted list of the given leaves (list of leaf positions). <br> Procedure step 7. |
| Measure Spacing | leaves | Returns a list of spacing between adjacent leaves. Procedure step 8. |
| Standard Deviation | items | Returns standard deviation of the given list of items (list of leaf spacing <br> angles). Procedure step 9. |
| Tournament | count, <br> angle | Iterates and returns a table of standard deviation of the given leaf count <br> (1 to count) and angle (0 to angle). Procedure step 10. |
| Cumulative Score | score | Returns a cumulative score table from the given standard deviation <br> score input. Procedure step 11 |
| Find Winner | score | Returns a list of winner for each tournament round (each leaf count) |
| Grow | angle, <br> count | Visualizes leaves arrangement for the given angle and count. |
| Leaf Calculator | angle, <br> count | Returns a single standard deviation of leaf spacing for the given input <br> of leaf rotation angle and leaf count. |



Figure 13. Structure of the programs and their data flow. At the core of this structure is the Leaf Calculator function which calls Standard Deviation which does the standard deviation, obtains inputs by calling Measure Spacing, Order It, and Leaf Position. Tournament function produces scoreboard by iterating from leaf count 1 to 30 and leaf rotation angle zero (0) to 180 calling Leaf Calculator in each iteration. Cumulative Score produces the cumulative scoreboard taking standard deviation scoreboard produced by Tournament. Find Winner function returns a list of winner, those with lowest score in each tournament round (each leaf count)


Figure 14. Summary of MIT Scratch program scripts developed and used in this project.
l/usr/bin/python
l/usr/bin/python
l/usr/bin/python
l/usr/bin/python
import math
import math
import math
import math
\& (ef grow(count-0, angle=0, sorted-False):
\& (ef grow(count-0, angle=0, sorted-False):
\& (ef grow(count-0, angle=0, sorted-False):
\& (ef grow(count-0, angle=0, sorted-False):
leaves - [1 (
leaves - [1 (
leaves - [1 (
leaves - [1 (
lol
lol
lol
lol
f sorted: (%)
f sorted: (%)
f sorted: (%)
f sorted: (%)
leaves.sort()
leaves.sort()
leaves.sort()
leaves.sort()
lef order_leaves(leaves-[]):
lef order_leaves(leaves-[]):
lef order_leaves(leaves-[]):
lef order_leaves(leaves-[]):
- leaves[:]
- leaves[:]
- leaves[:]
- leaves[:]
1.sort()
1.sort()
1.sort()
1.sort()
lef measure_space(leaves-[1):
lef measure_space(leaves-[1):
lef measure_space(leaves-[1):
lef measure_space(leaves-[1):
spacings = [] (len(leaves)):
spacings = [] (len(leaves)):
spacings = [] (len(leaves)):
spacings = [] (len(leaves)):
in}$$
\begin{array}{l}{\mathrm{ in range(len(leaves)}}\\{=-1en(leaves)-1:}\\{=-360-1eaves[1]}
        in}\begin{array}{l}{\mathrm{ in range(len(leaves)}}\\{=-1en(leaves)-1:}\\{=-360-1eaves[1]}
        in}\begin{array}{l}{\mathrm{ in range(len(leaves)}}\\{=-1en(leaves)-1:}\\{=-360-1eaves[1]}
        in}\begin{array}{l}{\mathrm{ in range(len(leaves)}}\\{=-1en(leaves)-1:}\\{=-360-1eaves[1]}
        = 360-leaves[ [1 
        = 360-leaves[ [1 
        = 360-leaves[ [1 
        = 360-leaves[ [1 
        s-leaves[i+1]_ leaves[i]
        s-leaves[i+1]_ leaves[i]
        s-leaves[i+1]_ leaves[i]
        s-leaves[i+1]_ leaves[i]
        spacings.append(s)
        spacings.append(s)
        spacings.append(s)
        spacings.append(s)
    det sum(items=[1)
    det sum(items=[1)
    det sum(items=[1)
    det sum(items=[1)
    total =0
    total =0
    total =0
    total =0
        cotal - total + i
        cotal - total + i
        cotal - total + i
        cotal - total + i
    ef mean(items-[1):
    ef mean(items-[1):
    ef mean(items-[1):
    ef mean(items-[1):
    ef mean(items-[1):
    ef mean(items-[1):
    ef mean(items-[1):
    ef mean(items-[1):
    total = sum(items=1tems)
    total = sum(items=1tems)
    total = sum(items=1tems)
    total = sum(items=1tems)
    return avg
    return avg
    return avg
    return avg
    jef squared_difference(items-[1):
    jef squared_difference(items-[1):
    jef squared_difference(items-[1):
    jef squared_difference(items-[1):
    avg - mean(items-items
    avg - mean(items-items
    avg - mean(items-items
    avg - mean(items-items
        sag_diff = [1 m
        sag_diff = [1 m
        sag_diff = [1 m
        sag_diff = [1 m




    ef variance(items=[1):
    ef variance(items=[1):
    ef variance(items=[1):
    ef variance(items=[1):
    sq_diff - squared_difference(items-items)
    sq_diff - squared_difference(items-items)
    sq_diff - squared_difference(items-items)
    sq_diff - squared_difference(items-items)
    l
    l
    l
    l
        def standard_deviation(items-[1):
        def standard_deviation(items-[1):
        def standard_deviation(items-[1):
    var - variance(itens-items)
std_dev - round(math. sqrt(var).8)
return std_dev
    var - variance(itens-items)
std_dev - round(math. sqrt(var).8)
return std_dev
    var - variance(itens-items)
std_dev - round(math. sqrt(var).8)
return std_dev
lef stdev_list(count-1, angle-0):
lef stdev_list(count-1, angle-0):
lef stdev_list(count-1, angle-0):
    sto_dey \(=[1\)
for \(n\) in range ( 1, count +1\()\)
    sto_dey \(=[1\)
for \(n\) in range ( 1, count +1\()\)
    sto_dey \(=[1\)
for \(n\) in range ( 1, count +1\()\)
        leaves - grow(count-n, angle-angle, sorted-True)
        leaves - grow(count-n, angle-angle, sorted-True)
        leaves - grow(count-n, angle-angle, sorted-True)
        leaves - grow(count-n, angle-angle. sorted-True)
mordered leaves - order leaves (leaves-leaves)
"spacings - measure.space(leaves-ordered_leaves)
        leaves - grow(count-n, angle-angle. sorted-True)
mordered leaves - order leaves (leaves-leaves)
"spacings - measure.space(leaves-ordered_leaves)
        leaves - grow(count-n, angle-angle. sorted-True)
mordered leaves - order leaves (leaves-leaves)
"spacings - measure.space(leaves-ordered_leaves)
        "spacings = measure -space(leaves-ordered_leaves
spacings - measure_space (leaves-leaves)
        "spacings = measure -space(leaves-ordered_leaves
spacings - measure_space (leaves-leaves)
        "spacings = measure -space(leaves-ordered_leaves
spacings - measure_space (leaves-leaves)
        spacings = measure_space(leaves-leaves)
std_dev.append(standard_deviation(items-spacings))
return std_dev
        spacings = measure_space(leaves-leaves)
std_dev.append(standard_deviation(items-spacings))
return std_dev
        spacings = measure_space(leaves-leaves)
std_dev.append(standard_deviation(items-spacings))
return std_dev
def stdev_cumulative(stdev=[1):
def stdev_cumulative(stdev=[1):
def stdev_cumulative(stdev=[1):
    stdev_cumn - [1
    stdev_cumn - [1
    stdev_cumn - [1
        for 2 in range(len(stdev)):
        for 2 in range(len(stdev)):
        for 2 in range(len(stdev)):
            stdev_cumm.append(stdev[i]
            stdev_cumm.append(stdev[i]
            stdev_cumm.append(stdev[i]
            se: - cum.append(stdev[i])
            se: - cum.append(stdev[i])
            se: - cum.append(stdev[i])
        stdev_cumm.append(stdev_cum \([i-1]+\operatorname{stdev}[1])\)
stidev_curmm
        stdev_cumm.append(stdev_cum \([i-1]+\operatorname{stdev}[1])\)
stidev_curmm
        stdev_cumm.append(stdev_cum \([i-1]+\operatorname{stdev}[1])\)
stidev_curmm
    def print_header(topleft-'angle\count', count=0, format-'xas'):
    def print_header(topleft-'angle\count', count=0, format-'xas'):
    def print_header(topleft-'angle\count', count=0, format-'xas'):



    for 1 in range( 1 .count +1 ):
    for 1 in range( 1 .count +1 ):
    for 1 in range( 1 .count +1 ):
        if i < count: . \(\%\).
        if i < count: . \(\%\).
        if i < count: . \(\%\).
        print format \(\$_{1} \ldots\)
        print format \(\$_{1} \ldots\)
        print format \(\$_{1} \ldots\)
        olse: print format \(\%\) i.
        olse: print format \(\%\) i.
        olse: print format \(\%\) i.
def print_list(list=[]. format='*s'):
def print_list(list=[]. format='*s'):
def print_list(list=[]. format='*s'):
    for it in range( len(1ist))
    for it in range( len(1ist))
    for it in range( len(1ist))
        if i \& leng(ist) -1 : ):
        if i \& leng(ist) -1 : ):
        if i \& leng(ist) -1 : ):
        print format \(x\) list \([i]+\)
        print format \(x\) list \([i]+\)
        print format \(x\) list \([i]+\)
        else: format \(\times\) list[i] +
print format \(\times\) list[i].
        else: format \(\times\) list[i] +
print format \(\times\) list[i].
        else: format \(\times\) list[i] +
print format \(\times\) list[i].
lef leaves(count=0, sorted-False):
lef leaves(count=0, sorted-False):
lef leaves(count=0, sorted-False):
    print, header (count -count)
    print, header (count -count)
    print, header (count -count)
        or angle in range(181)
        or angle in range(181)
        or angle in range(181)
        print "x5s. " \% angle.
        print "x5s. " \% angle.
        print "x5s. " \% angle.
        for i in range(lent (leaves)):
        for i in range(lent (leaves)):
        for i in range(lent (leaves)):






        if i \(<\) len(leaves)-1:
print '\%3s,' \% leaves[i],
else: \(\quad\) print '\%3s' \% leaves[i].
print
        if i \(<\) len(leaves)-1:
print '\%3s,' \% leaves[i],
else: \(\quad\) print '\%3s' \% leaves[i].
print
    *(%)
    *(%)
    *(%)
    *(%)
    \({ }_{\text {print }}^{\text {pr }}\)
    \({ }_{\text {print }}^{\text {pr }}\)
def leaves_spacing (angle \(=0\), count \(=0\) ) :
def leaves_spacing (angle \(=0\), count \(=0\) ) :
    def leaves_spacing(angle \(=0\), count \(=0\) ):
print_header(topleft \(=\prime\) stdev' , count \(=\) count, format \(=' \% 5 s^{\prime}\) )
    def leaves_spacing(angle \(=0\), count \(=0\) ):
print_header(topleft \(=\prime\) stdev' , count \(=\) count, format \(=' \% 5 s^{\prime}\) )
    print_header(topleft=' \({ }^{\text {stdev }}{ }^{\prime}\),
for c in range( 1 , count+1):
    print_header(topleft=' \({ }^{\text {stdev }}{ }^{\prime}\),
for c in range( 1 , count+1):
        leaves \(=\) grow(count \(=c\). angle \(=\) angle, sorted \(=T\) rue)
spacings \(=\) measure
        leaves \(=\) grow(count \(=c\). angle \(=\) angle, sorted \(=T\) rue)
spacings \(=\) measure
        spacings = measure_space(leaves=leaves)
        spacings = measure_space(leaves=leaves)
        sd = standard_deviation(items=spacings)
        sd = standard_deviation(items=spacings)
        print '\%6.2f,' \% sd,
        print '\%6.2f,' \% sd,
def std_leaves(count=0, cumulative=False):
def std_leaves(count=0, cumulative=False):
def std_leaves (count \(=0\), cumu
print_header (count \(=\) count \()\)
def std_leaves (count \(=0\), cumu
print_header (count \(=\) count \()\)
    print_header(count=count)
for angle in range(181):
    print_header(count=count)
for angle in range(181):


        if cumulative:
        if cumulative:
        list = stdev_cumulative(stdev=list)
        list = stdev_cumulative(stdev=list)
        print \%5s, \% angle,
for i in range(len(list)):
if \(i<1\) len(list) 1 :
        print \%5s, \% angle,
for i in range(len(list)):
if \(i<1\) len(list) 1 :
        if \(i\) in range(len(list)):
        if \(i\) in range(len(list)):
        if \(i<1\) en(list)
print
.1:
\(\% 7.2 f\).
\(\%\) list[i].
        if \(i<1\) en(list)
print
.1:
\(\% 7.2 f\).
\(\%\) list[i].
        print '\%7.2f.' \% list[i].
else:
print \(\times \% 7.2 f\) ' \% list[i]
        print '\%7.2f.' \% list[i].
else:
print \(\times \% 7.2 f\) ' \% list[i]
def find_winners(scoreboard=\{
keys \(=\) scoreboard.keys ()
def find_winners(scoreboard=\{
keys \(=\) scoreboard.keys ()
    keys =- scoreboard.keys()
    keys =- scoreboard.keys()
    keys. sort ()
kinners = []
    keys. sort ()
kinners = []
    Winners = []
winners_cumu_sdev = []
for i in range(len(scor
    Winners = []
winners_cumu_sdev = []
for i in range(len(scor
        for \(i\) in range(len(scoreboard[keys[0]])):
        for \(i\) in range(len(scoreboard[keys[0]])):
        winners. append (0)
winners_cumu_sdev.append ( 10
        winners. append (0)
winners_cumu_sdev.append ( 10
        winners_cumu_sdev.append (1000000)
        winners_cumu_sdev.append (1000000)
    for \(i\) in range(len(scoreboard[keys[0]])):
for \(k\) in keys:
    for \(i\) in range(len(scoreboard[keys[0]])):
for \(k\) in keys:
        if scoreboard[k][i] < winners_cumu_sdev[i]
        if scoreboard[k][i] < winners_cumu_sdev[i]


        winners \([i]=k\)
        winners \([i]=k\)
    print winners
print winners_cumu_sdev
    print winners
print winners_cumu_sdev
    print wi
sum \(=0\)
    print wi
sum \(=0\)
    sum \(=0\)
for \(i\) in
    sum \(=0\)
for \(i\) in
    sum \(=0\)
for \(i\) in winners:
sum \(+=i\)
    sum \(=0\)
for \(i\) in winners:
sum \(+=i\)
    \(\begin{array}{l}\text { sum } \\ \text { print } \\ \text { average }\end{array}=\) i s. \(^{\prime}\) \% (sum/len(winners) \()\)
    \(\begin{array}{l}\text { sum } \\ \text { print } \\ \text { average }\end{array}
$$=\) i s. $^{\prime}$ \% (sum/len(winners) $)$

Figure 15. Summary of Python program scripts developed and used in this project.

## 6. Future Work

Future projects may include various pattern and trend analysis in disciplines such as geology, medicine, sociology, politics, and economics, wherein the Fibonacci series may be applied. Utilizing this programming and data visualization approach to process and analyze large data sets for pattern detection can be a powerful tool for understanding the current state of problems within these disciplines, and for making future state predictions.

## Acknowledgements

The author would like to thank his advisor, Patrick Giuliano for his guidance, and IEOM leadership for their timely and thoughtful consideration of this work.

## References

Campbell, S. C., and Campbell, R. P., Growing patterns: Fibonacci numbers in nature. Honesdale, PA: Boyds Mills Press, 2010.
Devlin, K. The Man of Numbers. London: Bloomsbury, pp. 147, 2014
Mathisfun.com, Standard Deviation and Variance. Available: http://www.mathsisfun.com/data/standarddeviation.html, 16 Jan. 2013.
Knott, R., and the Plus team, The Life and Numbers of Fibonacci, Available: https://plus.maths.org/ content/life-and-numbers-fibonacci. 17 Jan. 2013.
Fratini, R. M., Sciteneg Genetics, Fibonacci Digital Simulation of Spiral Phyllotaxis, Available: http://www.sciteneg.com/ PhiTaxis/PHYLLOTAXIS.htm, 3 Dec 2012

## Biography

Raymond Yap is a 7th grader at Casa di Mir Montessori school. He participated in Montessori Model United Nations (MMUN) and was elected to speak at the United Nations General Assembly in New York, USA in March 2017 about Nuclear Disarmament and Prevention of An Arms Race In Outer Space. As a First Lego League Robotics competitor, he successfully led his team in competition for 3 consecutive years, to win the Robot Design and Robot Performance Award in 2016. He is currently at an 8th grade level in Mandarin Chinese, and is fluent in both verbal and written forms. He is currently a level 6 pianist with 7 years of experience, and he has received the Music Teachers’ Association of California (MTAC) Branch Honors award twice in 2014 and 2015. He is a junior black belt in Aikido martial arts. Raymond is also an experienced MIT Scratch programmer, and his research interests include architecture, computer technology, digital music composition, and NBA basketball.

