Thinking about Problems

Ryan Greenup & James Guerra

August 25, 2020

Contents

§ 1 Introduction

During preperation for this outline, an article published by the *Mathematical Association of America* caught my attention, in which mathematics is referred to as the *Science of Patterns* [5], this I feel, frames very well the essence of the research we are looking at in this project. Mathematics, generally, is primarily concerned with problem solving (that isn't, however, to say that the problems need to have any application[1]), and it's fairly obvious that different strategies work better for different problems. That's what we want to investigate, Different to attack a problem, different ways of thinking, different ways of framing questions.

[1]Although Hardy made a good defence of pure math in his 1940s Apology [6], it isn't rare at all for pure math to be found applications, for example much number theory was probably seen as fairly pure before RSA Encryption [12].

The central focus of this investigation will be with computer algebra and the various libraries and packages that exist in the free open source [2] space to solve and visualise numeric and symbolic problems, these include:

- Programming Languages and CAS
  - Julia
    - SymEngine
  - Maxima
    - Being the oldest there is probably a lot too learn
  - Julia
  - Reduce
  - Xcas/Gias
  - Python
    - Numpy
    - Sympy
- Visualisation
  - Makie
  - Plotly
  - GNUPlot

Many problems that look complex upon initial inspection can be solved trivially by using computer algebra packages and our interest is in the different approaches that can be taken to *attack* each problem. Of course however this leads to the question:

> Can all mathematical problems be solved by some application of some set of rules?

This is not really a question that we can answer, however, determinism with respect to systems is appears to make a very good area of investigation with respect to finding ways to deal with problems.

This is not an easy question to answer, however, while investigating this problem

Determinism

Are problems deterministic? can the be broken down into a step by step way? For example if we *discover all the rules* can we then simply solve all the problems?

chaos to look at patterns generally to get a deeper understanding of patterns and problems, loops and recursion generally.

To investigate different ways of thinking about math problems our investigation

laplaces demon

but then heisenberg,

but then chaos and meh.

¶ 1.1 Preliminary Problems

1.1.1 Recursion

1.1.2 Iteration and Recursion

To illustrate an example of different ways of thinking about a problem, consider the series shown in (1)[3] :

[2] Although proprietary software such as Magma, Mathematica and Maple is very good, the restrictive licence makes them undesirable for study because there is no means by which to inspect the problem solving tecniques implemented, build on top of the work and moreover the lock-in nature of the software makes it a risky investment with respect to time.

[3] This problem is taken from Project A (44) of Dr. Hazrat's *Mathematica: A Problem Centred Approach* [7]

$$g(k) = \frac{\sqrt{2}}{2} \cdot \frac{\sqrt{2+\sqrt{3}}}{3} \frac{\sqrt{2+\sqrt{3+\sqrt{4}}}}{4} \cdot \ldots \frac{\sqrt{2+\sqrt{3+\ldots+\sqrt{k}}}}{k} \tag{1}$$

let's modify this for the sake of discussion:

$$h(k) = \frac{\sqrt{2}}{2} \cdot \frac{\sqrt{3+\sqrt{2}}}{3} \cdot \frac{\sqrt{4+\sqrt{3+\sqrt{2}}}}{4} \cdot \ldots \cdot \frac{\sqrt{k+\sqrt{k-1+\ldots\sqrt{3+\sqrt{2}}}}}{k} \tag{2}$$

The function h can be expressed by the series:

$$h(k) = \prod_{i=2}^{k} \left(\frac{f_i}{i}\right) \quad : \quad f_i = \sqrt{i + f_{i-1}}, \;\; f_1 = 1$$

Within *Python*, it isn't difficult to express h, the series can be expressed with recursion as shown in listing 1, this is a very natural way to define series and sequences and is consistent with familiar mathematical thought and notation. Individuals more familiar with programming than analysis may find it more comfortable to use an iterator as shown in listing 2.

```python
from sympy import *
def h(k):
    if k > 2:
        return f(k) * f(k-1)
    else:
        return 1

def f(i):
    expr = 0
    if i > 2:
        return sqrt(i + f(i -1))
    else:
        return 1
```

Listing 1: Solving (2) using recursion.

```python
from sympy import *
def h(k):
    k = k + 1 # OBOB
    l = [f(i) for i in range(1,k)]
    return prod(l)

def f(k):
    expr = 0
    for i in range(2, k+2):
        expr = sqrt(i + expr, evaluate=False)
    return expr/(k+1)
```

Listing 2: Solving (2) by using a `for` loop.

Any function that can be defined by using iteration, can always be defined via recursion and vice versa, [4, 3] see also [11, 8] there is, however, evidence to suggest that recursive functions are easier for people to understand [2] . Although independent research has shown that the specific language chosen can have a bigger effect on how well recursive as opposed to iterative code is understood [10].

The relevant question is which method is often more appropriate, generally the process for determining which is more appropriate is to the effect of:

1. Write the problem in a way that is easier to write or is more appropriate for demonstration
2. If performance is a concern then consider restructuring in favour of iteration
   - For interpreted languages such ***R*** and *Python*, loops are usually faster, because of the overheads involved in creating functions [11] although there may be exceptions to this and I'm not sure if this would be true for compiled languages such as *Julia*, *Java*, ***C*** etc.

Some Functions are more difficult to express with Recursion in

Attacking a problem recursively isn't always the best approach, consider the function $g(k)$ from (1):

$$\begin{aligned} g(k) &= \frac{\sqrt{2}}{2} \cdot \frac{\sqrt{2+\sqrt{3}}}{3} \frac{\sqrt{2+\sqrt{3+\sqrt{4}}}}{4} \cdot \ldots \frac{\sqrt{2+\sqrt{3+\ldots+\sqrt{k}}}}{k} \\ &= \prod_{i=2}^{k} \left(\frac{f_i}{i}\right) \quad : \quad f_i = \sqrt{i + f_{i+1}} \end{aligned}$$

Observe that the difference between (1) and (2) is that the sequence essentially *looks* forward, not back. To solve using a `for` loop, this distinction is a non-concern because the list can be reversed using a built-in such as `rev`, `reversed` or `reverse` in *Python*, ***R*** and *Julia* respectively, which means the same expression can be implemented.

To implement recursion however, the series needs to be restructured and this can become a little clumsy, see (3):

$$g(k) = \prod_{i=2}^{k} \left(\frac{f_i}{i}\right) \quad : \quad f_i = \sqrt{(k-i) + f_{k-i-1}} \tag{3}$$

Now the function could be performed recursively in *Python* in a similar way as shown in listing 3, but it's also significantly more confusing because the f function now has k as a parameter and this is only made significantly more complicated by the variable scope of functions across common languages used in Mathematics and Data science such as `bash`, *Python*, ***R*** and *Julia* (see section 1.1.3).

If however, the `for` loop approach was implemented, as shown in listing 4, the function would not significantly change, because the `reversed()` function can be used to flip the list around.

What this demonstrates is that taking a different approach to simply describing this function can lead to big differences in the complexity involved in solving this problem.

1.1.3 Variable Scope of Nested Functions

§ 2 Outline

1. Intro Prob
2. Variable Scope
3. Problem Showing Recursion
   - All Different Methods
     - Discuss all Different Methods
     - Discuss Vectorisation

```python
from sympy import *
def h(k):
    if k > 2:
        return f(k, k) * f(k, k-1)
    else:
        return 1

def f(k, i):
    if k > i:
        return 1
    if i > 2:
        return sqrt((k-i) + f(k, k - i -1))
    else:
        return 1
```

Listing 3: Using Recursion to Solve (1)

```python
from sympy import *
def h(k):
    k = k + 1 # OBOB
    l = [f(i) for i in range(1,k)]
    return prod(l)

def f(k):
    expr = 0
    for i in reversed(range(2, k+2)):
        expr = sqrt(i + expr, evaluate=False)
    return expr/(k+1)
```

Listing 4: Using Iteration to Solve (1)

    - Is this needed in Julia
    - Comment on Faster to go column Wise

4. Discuss Loops

5. Show Rug

6. Fibonacci

   - The ratio of fibonacci converges to ϕ
   - Golden Ratio
     - If you make a rectangle with the golden ratio you can cut it up under recursion to get another one, keep doing this and eventually a logarithmic spiral pops out, also the areas follow a fibonacci sequence.

7. Discuss isomorphisms for recursive Relations

8. Jump to Lorenz Attractor

9. Now Talk about Morphogenesis

10. Fractals

    - Many Occur in Nature
      - Mountain Ranges, compare to MandelBrot
      - Sun Flowers
      - Show the golden Ratio
    - Fractals are all about recursion and iteration, so this gives me an excuse to look at them
      - Show MandelBrot
        - Python
          - Sympy Slow
          - Numpy Fast
        - Julia brings Both Benefits
          - Show Large MandelBrot
        - Show Julia Set
          - Show Julia Set Gif

11. Things I'd like to show

    - Simulate stripes and animal patterns
    - Show some math behind spirals in Nautilus Shells
    - Golden Rectangle
      - Throw in some recursion
      - Watch the spiral come out
      - Record the areas and show that they are Fibonacci
    - That the ratio of Fibonacci Converges to Phi
    - What on Earth is the Reimann Sphere
    - Lorrenz Attractor
      - How is this connected to the lorrenz attractor
    - What are the connections between discrete iteration and continuous systems such as the julia set and the lorrenz attractor

12. Things I'd like to Try (in order to see different ways to approach Problems)

    - Programming Languages and CAS

  - Julia
    * SymEngine
  - Maxima
  - Julia
- Visualisation
  - Makie
  - Plotly
  - GNUPlot

§ 3 Download RevealJS

So first do `M-x package-install ox-reveal` then do `M-x load-library` and then look for `ox-reveal`

```
(load "/home/ryan/.emacs.d/.local/straight/build/ox-reveal/ox-reveal.el")
```

Download Reveal.js and put it in the directory as `./reveal.js`, you can do that with something like this:

```
# cd /home/ryan/Dropbox/Studies/2020Spring/QuantProject/Current/Python-Quant/Outline/
wget https://github.com/hakimel/reveal.js/archive/master.tar.gz
tar -xzvf master.tar.gz && rm master.tar.gz
mv reveal.js-master reveal.js
```

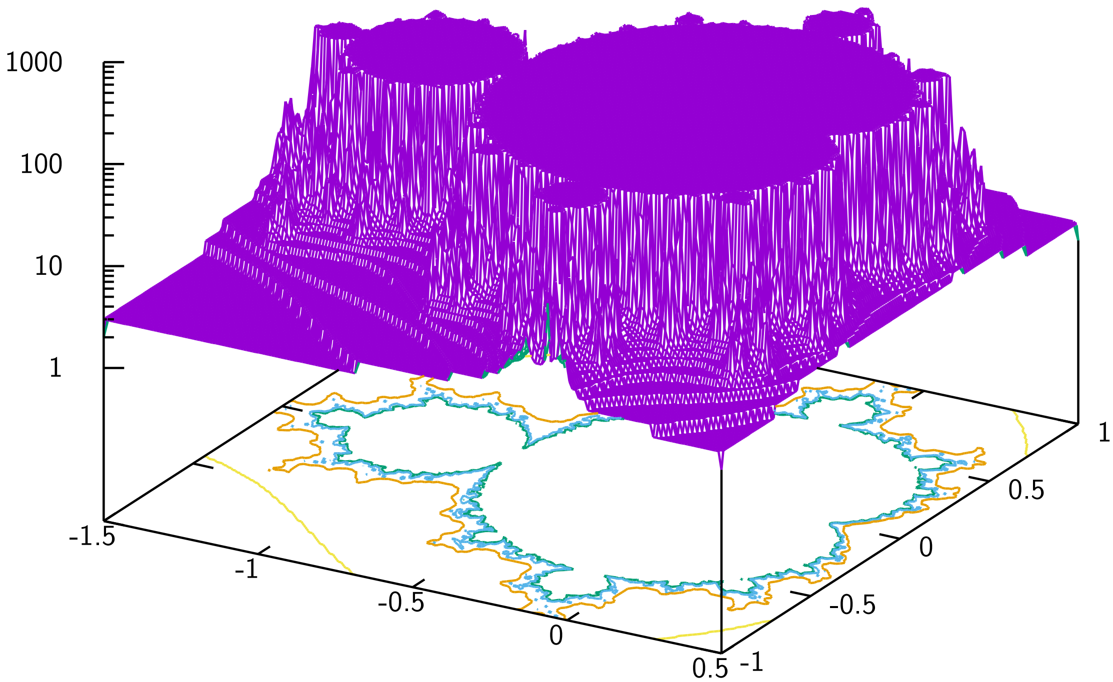
Then just do `C-c e e R R` to export with RevealJS as opposed to PHP you won't need a fancy server, just open it in the browser.

§ 4 GNU Plot

limit of recursion is 250

```
complex(x,y) = x*{1,0}+y*{0,1}
mandel(x,y,z,n) = (abs(z)>2.0 || n>=200) ? \
                  n : mandel(x,y,z*z+complex(x,y),n+1)

set xrange [-1.5:0.5]
set yrange [-1:1]
set logscale z
set isosample 200
set hidden3d
set contour
splot mandel(x,y,{0,0},0) notitle
```

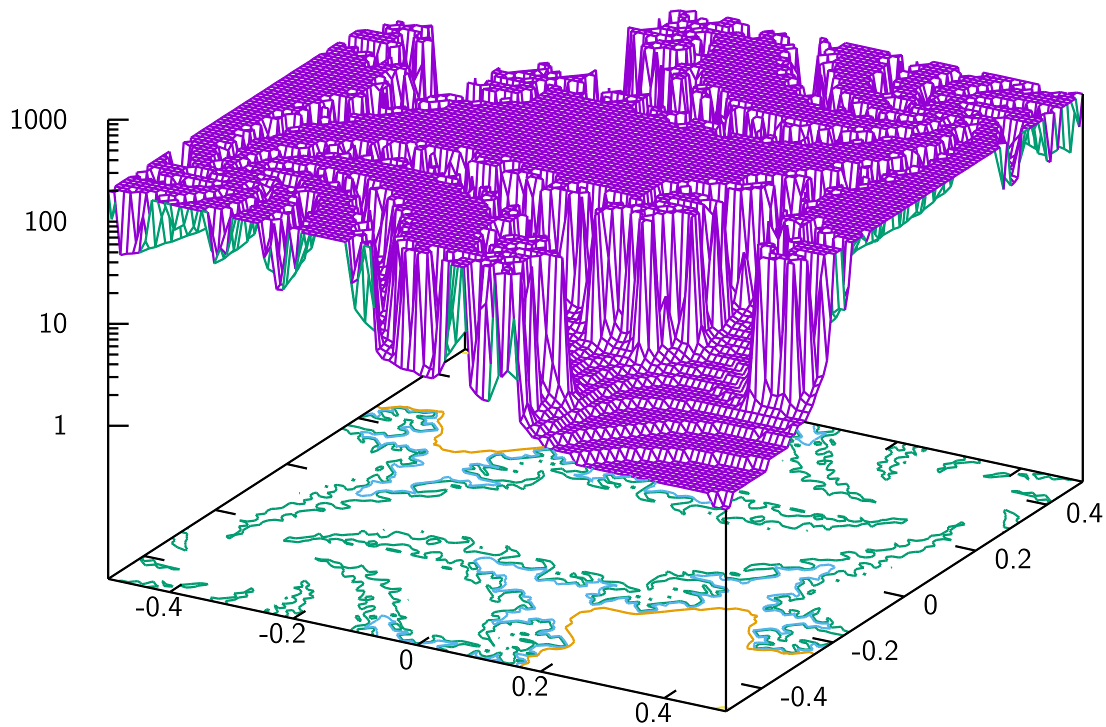


reference for image

,#+begin$_{\text{src}}$ gnuplot

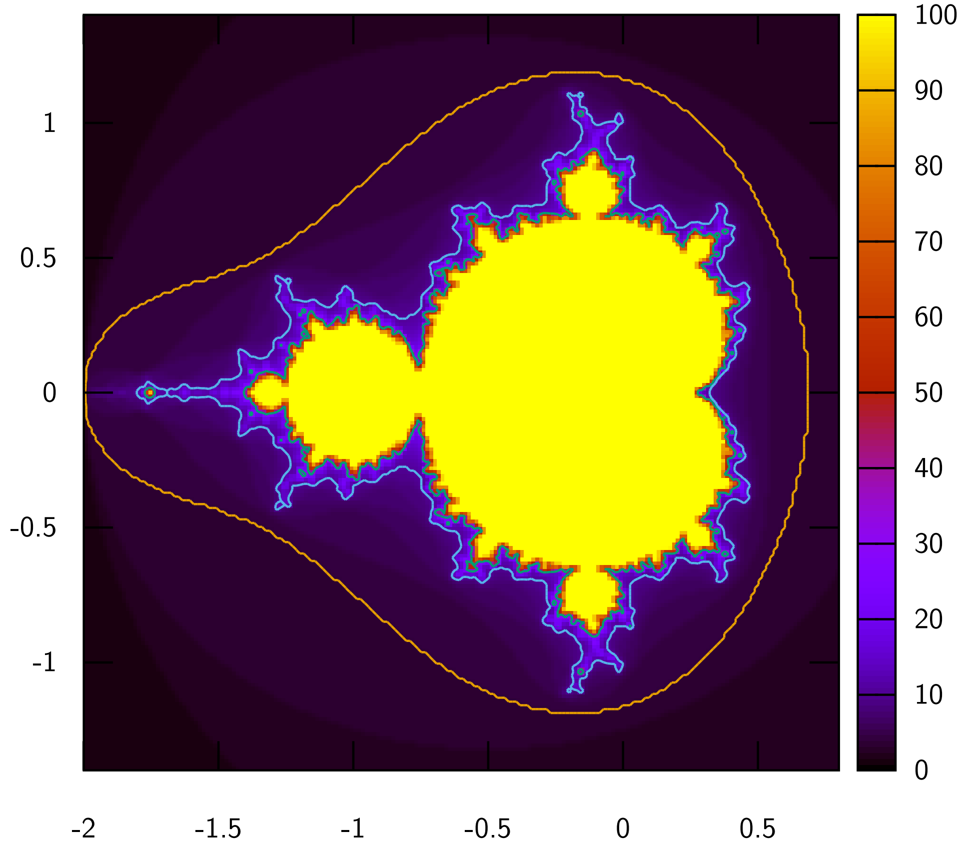
```
complex(x,y) = x*{1,0}+y*{0,1}
mandel(x,y,z,n) = (abs(z)>2.0 || n>=200) ? \
                  n : mandel(x,y,z*z+complex(x,y),n+1)

set xrange [-0.5:0.5]
set yrange [-0.5:0.5]
set logscale z
set isosample 100
set hidden3d
set contour
a= -0.37
b= -0.612
splot mandel(a,b,complex(x,y),0) notitle
```



reference

```
rmax = 2
nmax = 100
complex (x, y) = x * {1, 0} + y * {0, 1}
mandelbrot (z, z0, n) = n == nmax || abs (z) > rmax ? n : mandelbrot (z ** 2 + z0, z0, n + 1)
set samples 200
set isosamples 200
set pm3d map
set size square
splot [-2 : .8] [-1.4 : 1.4] mandelbrot (complex (0, 0), complex (x, y), 0) notitle
```



§ 5 Heres a Gif

So this is a very big Gif that I'm using:

How did I make the Gif??

https://dl.dropboxusercontent.com/s/rbu25urfg8sbwfu/out.gif?dl=0

§ 6 Give a brief Sketch of the project

```
code /home/ryan/Dropbox/Studies/QuantProject/Current/Python-Quant/ & disown
```

Here's what I gatthered from the week 3 slides

¶ 6.1 Topic / Context

We are interested in the theory of problem solving, but in particular the different approaches that can be taken to attacking a problem.

Essentially this boils down to looking at how a computer scientist and mathematician attack a problem, although originally I thought there was no difference, after seeing the odd way Roozbeh attacks problems I see there is a big difference.

¶ 6.2 Motivation

¶ 6.3 Basic Ideas

- Look at FOSS CAS Systems
  - Python (Sympy)
  - Julia
    * Sympy integration
    * symEngine
    * Reduce.jl
    * Symata.jl
- Maybe look at interactive sessions:
  - Like Jupyter
  - Hydrogen
  - TeXmacs
  - org-mode?

After getting an overview of SymPy let's look at problems that are interesting (chaos, morphogenesis and order from disarray etc.)

¶ 6.4 Where are the Mathematics

- Trying to look at the algorithms underlying functions in Python/Sympy and other Computer algebra tools such as Maxima, Maple, Mathematica, Sage, GAP and Xcas/Giac, Yacas, Symata.jl, Reduce.jl, SymEngine.jl
  - For Example Recursive Relations
- Look at solving some problems related to chaos theory maybe
  - Mandelbrot and Julia Sets
- Look at solving some problems related to Fourier Transforms maybe

AVOID DETAILS, JUST SKETCH THE PROJECT OUT.

¶ 6.5 Don't Forget we need a talk

6.5.1 Slides In Org Mode

- Without Beamer
- With Beamer

§ 7 Undecided

7.0.1 Determinant

Computational thinking can be useful in problems related to modelling, consider for example some matrix $n \times n$ matrix B_n described by (4) :

$$b_{ij} = \begin{cases} \frac{1}{2j-i^2}, & \text{if } i > j \\ \frac{i}{i-j} + \frac{1}{n^2-j-i}, & \text{if } j > i \\ 0 & \text{if } i = j \end{cases} \tag{4}$$

Is there a way to predict the determinant of such a matrix for large values?
From the perspective of linear algebra this is an immensely difficult problem and there isn't really a clear place to start.
From a numerical modelling perspective however, as will be shown, this a fairly trivial problem.

Create the Matrix

Using *Python* and `numpy`, a matrix can be generated as an `array` and by iterating through each element of the matrix values can be attributed like so:

```python
import numpy as np
n = 2
mymat = np.empty([n, n])
for i in range(mymat.shape[0]):
    for j in range(mymat.shape[1]):
        print("(" + str(i) + "," + str(j) + ")")
```

```
(0,0)
(0,1)
(1,0)
(1,1)
```

and so to assign the values based on the condition in (4), an `if` test can be used:

```python
def BuildMat(n):
    mymat = np.empty([n, n])
    for i in range(n):
        for j in range(n):
            # Increment i and j by one because they count from zero
            i += 1; j += 1
            if (i > j):
                v = 1/(2*j - i**2)
            elif (j > i):
                v = 1/(i-j) + 1/(n**2 - j - i)
            else:
                v = 0
            # Decrement i and j so the index lines up
            i -= 1; j -= 1
            mymat[j, i] = v
    return mymat

BuildMat(3)
```

```
array([[ 0.        , -0.5       , -0.14285714],
       [-0.83333333,  0.        , -0.2       ],
       [-0.3       , -0.75      ,  0.        ]])
```

Find the Determinant

Python, being an object orientated language has methods belonging to objects of different types, in this case the `linalg` method has a `det` function that can be used to return the determinant of any given matrix like so:

```python
def detMat(n):
    ## Sympy
    # return Determinant(BuildMat(n)).doit()
    ## Numpy
    return np.linalg.det(BuildMat(n))
detMat(3)
```

Listing 5: Building a Function to return the determinant of the matrix described in (4)

```
-0.11928571428571424
```

Find the Determinant of Various Values

To solve this problem, all that needs to be considered is the size of the n and the corresponding determinant, this could be expressed as a set as shown in (**??**):

$$\left\{ \det\left(M(n)\right) \mid M \in \mathbb{Z}^{+} \leq 30 \right\} \tag{5}$$

where:

- M is a function that transforms an integer to a matrix as per (4)

Although describing the results as a set (5) is a little odd, it is consistent with the idea of list and set comprehension in *Python* [1] and *Julia* [9] as shown in listing 6

Generate a list of values Using the function created in listing 5, a corresponding list of values can be generated:

```python
def detMat(n):
    return abs(np.linalg.det(BuildMat(n)))

# We double all numbers using map()
result = map(detMat, range(30))

# print(list(result))
[round(num, 3) for num in list(result)]
```

Listing 6: Generate a list using list-comprehension

```
[1.0,
 0.0,
 0.0,
```

```
0.119,
0.035,
0.018,
0.013,
0.01,
0.008,
0.006,
0.005,
0.004,
0.004,
0.003,
0.003,
0.002,
0.002,
0.002,
0.002,
0.001,
0.001,
0.001,
0.001,
0.001,
0.001,
0.001,
0.001,
0.001,
0.001,
0.001]
```

```python
import pandas as pd

data = {'Matrix.Size': range(30),
        'Determinant.Value': list(map(detMat, range(30)))
}



df = pd.DataFrame(data, columns = ['Matrix.Size', 'Determinant.Value'])

print(df)
```

Create a Data Frame

```
Matrix.Size  Determinant.Value
0             0            1.000000
1             1            0.000000
2             2            0.000000
3             3            0.119286
4             4            0.035258
5             5            0.018062
6             6            0.013023
7             7            0.009959
8             8            0.007822
9             9            0.006288
10           10            0.005158
11           11            0.004304
```

12	12	0.003645
13	13	0.003125
14	14	0.002708
15	15	0.002369
16	16	0.002090
17	17	0.001857
18	18	0.001661
19	19	0.001494
20	20	0.001351
21	21	0.001228
22	22	0.001121
23	23	0.001027
24	24	0.000945
25	25	0.000872
26	26	0.000807
27	27	0.000749
28	28	0.000697
29	29	0.000650

Plot the Data frame Observe that it is necessary to use `copy`, *Julia* and *Python* **unlike** *Mathematica* and ***R*** only create links between data, they do not create new objects, this can cause headaches when rounding data.

```
from plotnine import *
import copy

df_plot = copy.copy(df[3:])
df_plot['Determinant.Value'] = df_plot['Determinant.Value'].astype(float).round(3)
df_plot

(
    ggplot(df_plot, aes(x = 'Matrix.Size', y = 'Determinant.Value')) +
        geom_point() +
        theme_bw() +
        labs(x = "Matrix Size", y = "|Determinant Value|") +
        ggtitle('Magnitude of Determinant Given Matrix Size')

)
```

```
<ggplot: (8770001690691)>
```

In this case it appears that the determinant scales exponentially, we can attempt to model that linearly using `scikit`, this is significantly more complex than simply using ***R***. ^lrpy

```python
import numpy as np
import matplotlib.pyplot as plt  # To visualize
import pandas as pd  # To read data
from sklearn.linear_model import LinearRegression

df_slice = df[3:]

X = df_slice.iloc[:, 0].values.reshape(-1, 1)  # values converts it into a numpy array
Y = df_slice.iloc[:, 1].values.reshape(-1, 1)  # -1 means that calculate the dimension
↪  of rows, but have 1 column
linear_regressor = LinearRegression()  # create object for the class
linear_regressor.fit(X, Y)  # perform linear regression
Y_pred = linear_regressor.predict(X)  # make predictions



plt.scatter(X, Y)
plt.plot(X, Y_pred, color='red')
plt.show()
```

```
array([5.37864677])
```

Log Transform the Data

The `log` function is actually provided by `sympy`, to do this quicker in `numpy` use `np.log()`

```python
# # pyperclip.copy(df.columns[0])
# #df['Determinant.Value'] =
# #[ np.log(val) for val in df['Determinant.Value']]

df_log = df

df_log['Determinant.Value'] = [ np.log(val) for val in df['Determinant.Value'] ]
```

In order to only have well defined values, consider only after size 3

```python
df_plot = df_log[3:]
df_plot
```

	Matrix.Size	Determinant.Value
3	3	-2.126234
4	4	-3.345075
5	5	-4.013934
6	6	-4.341001
7	7	-4.609294
8	8	-4.850835
9	9	-5.069048
10	10	-5.267129
11	11	-5.448099
12	12	-5.614501

```
13              13           -5.768414
14              14           -5.911529
15              15           -6.045230
16              16           -6.170659
17              17           -6.288765
18              18           -6.400347
19              19           -6.506082
20              20           -6.606547
21              21           -6.702237
22              22           -6.793585
23              23           -6.880964
24              24           -6.964704
25              25           -7.045094
26              26           -7.122390
27              27           -7.196822
28              28           -7.268592
29              29           -7.337885
```

A limitation of the *Python* `plotnine` library (compared to *Ggplot2* in ***R***) is that it isn't possible to round values in the `aesthetics` layer, a further limitation with `pandas` also exists when compared to ***R*** that makes rounding data very clusy to do.

In order to round data use the `numpy` library:

```
import pandas as pd
import numpy as np
df_plot['Determinant.Value'] = df_plot['Determinant.Value'].astype(float).round(3)
df_plot
```

```
    Matrix.Size  Determinant.Value
3             3             -2.126
4             4             -3.345
5             5             -4.014
6             6             -4.341
7             7             -4.609
8             8             -4.851
9             9             -5.069
10           10             -5.267
11           11             -5.448
12           12             -5.615
13           13             -5.768
14           14             -5.912
15           15             -6.045
16           16             -6.171
17           17             -6.289
18           18             -6.400
19           19             -6.506
20           20             -6.607
21           21             -6.702
22           22             -6.794
23           23             -6.881
24           24             -6.965
25           25             -7.045
26           26             -7.122
27           27             -7.197
28           28             -7.269
```

29 29 -7.338

```python
from plotnine import *


(ggplot(df_plot[3:], aes(x = 'Matrix.Size', y = 'Determinant.Value')) +
   geom_point(fill= "Blue") +
   labs(x = "Matrix Size", y = "Determinant Value",
        title = "Plot of Determinant Values") +
   theme_bw() +
   stat_smooth(method = 'lm')
)
```

<ggplot: (8770002281897)>

```python
from sklearn.linear_model import LinearRegression

df_slice = df_plot[3:]

X = df_slice.iloc[:, 0].values.reshape(-1, 1)  # values converts it into a numpy array
Y = df_slice.iloc[:, 1].values.reshape(-1, 1)  # -1 means that calculate the dimension
↪  of rows, but have 1 column
linear_regressor = LinearRegression()  # create object for the class
linear_regressor.fit(X, Y)  # perform linear regression
Y_pred = linear_regressor.predict(X)  # make predictions



plt.scatter(X, Y)
plt.plot(X, Y_pred, color='red')
plt.show()
```

```python
m = linear_regressor.fit(X, Y).coef_[0][0]
b = linear_regressor.fit(X, Y).intercept_[0]

print("y = " + str(m.round(2)) + "* x" + str(b.round(2)))
```

y = -0.12* x-4.02

So the model is:

$$\text{abs}(\text{Det}(M)) = -4n - 0.12$$

where:

- n is the size of the square matrix

Largest Percentage Error

To find the largest percentage error for $n \in [30, 50]$ it will be necessary to calculate the determinants for the larger range, compressing all the previous steps and calculating the model based on the larger amount of data:

```
import pandas as pd

data = {'Matrix.Size': range(30, 50),
        'Determinant.Value': list(map(detMat, range(30, 50)))
}
df = pd.DataFrame(data, columns = ['Matrix.Size', 'Determinant.Value'])
df['Determinant.Value'] = [ np.log(val) for val in df['Determinant.Value']]
df
from sklearn.linear_model import LinearRegression


X = df.iloc[:, 0].values.reshape(-1, 1)  # values converts it into a numpy array
Y = df.iloc[:, 1].values.reshape(-1, 1)  # -1 means that calculate the dimension of
↪ rows, but have 1 column
linear_regressor = LinearRegression()  # create object for the class
linear_regressor.fit(X, Y)  # perform linear regression
Y_pred = linear_regressor.predict(X)  # make predictions

m = linear_regressor.fit(X, Y).coef_[0][0]
b = linear_regressor.fit(X, Y).intercept_[0]

print("y = " + str(m.round(2)) + "* x" + str(b.round(2)))
```

```
y = -0.05* x-5.92
```

```
Y_hat = linear_regressor.predict(X)
res_per = (Y - Y_hat)/Y_hat
res_per
```

```
array([[-5.41415364e-03],
       [-3.51384602e-03],
       [-1.90798428e-03],
       [-5.74487234e-04],
       [ 5.06726599e-04],
       [ 1.35396448e-03],
       [ 1.98395424e-03],
       [ 2.41201322e-03],
       [ 2.65219545e-03],
       [ 2.71742022e-03],
       [ 2.61958495e-03],
       [ 2.36966444e-03],
       [ 1.97779855e-03],
       [ 1.45336983e-03],
       [ 8.05072416e-04],
       [ 4.09734813e-05],
       [-8.31432011e-04],
       [-1.80517224e-03],
```

```
[-2.87375452e-03],
[-4.03112573e-03]])
```

```python
max_res = np.max(res_per)
max_ind = np.where(res_per == max_res)[0][0] + 30

print("The Maximum Percentage error is " + str(max_res.round(4) * 100) + "% which
 ↪ corresponds to a matrix of size " + str(max_ind))
```

```
The Maximum Percentage error is 0.27% which corresponds to a matrix of size 39
```

§ 8 What we're looking for

- Would a reader know what the project is about?
- Would a reader become interested in the upcoming report?
- Is it brief but well prepared?
- Are the major parts or phases sketched out

References

[1] *5. Data Structures Python 3.8.5 Documentation*. URL: https://docs.python.org/3/tutorial/datastructures.html (visited on 08/24/2020) (cit. on p. 12).

[2] A. C Benander, B. A Benander, and Janche Sang. "An Empirical Analysis of Debugging Performance Differences between Iterative and Recursive Constructs". In: *Journal of Systems and Software* 54.1 (Sept. 30, 2000), pp. 17–28. ISSN: 0164-1212. DOI: 10.1016/S0164-1212(00)00023-6. URL: http://www.sciencedirect.com/science/article/pii/S0164121200000236 (visited on 08/24/2020) (cit. on p. 2).

[3] Corrado Böhm. "Reducing Recursion to Iteration by Algebraic Extension: Extended Abstract". In: *ESOP 86*. Ed. by Bernard Robinet and Reinhard Wilhelm. Red. by G. Goos et al. Vol. 213. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 1986, pp. 111–118. ISBN: 978-3-540-16442-5 978-3-540-39782-3. DOI: 10.1007/3-540-16442-1_8. URL: http://link.springer.com/10.1007/3-540-16442-1_8 (visited on 08/24/2020) (cit. on p. 2).

[4] Corrado Böhm. "Reducing Recursion to Iteration by Means of Pairs and N-Tuples". In: *Foundations of Logic and Functional Programming*. Ed. by Mauro Boscarol, Luigia Carlucci Aiello, and Giorgio Levi. Red. by G. Goos et al. Vol. 306. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 1988, pp. 58–66. ISBN: 978-3-540-19129-2 978-3-540-39126-5. DOI: 10.1007/3-540-19129-1_3. URL: http://link.springer.com/10.1007/3-540-19129-1_3 (visited on 08/24/2020) (cit. on p. 2).

[5] Michael Fried. "Mathematics as the Science of Patterns - Mathematics as the Science of Patterns | Mathematical Association of America". In: *Convergence* (Aug. 2010). URL: https://www.maa.org/press/periodicals/convergence/mathematics-as-the-science-of-patterns-mathematics-as-the-science-of-patterns (visited on 08/24/2020) (cit. on p. 1).

[6] G. H Hardy. *A Mathematician's Apology*. OCLC: 1172189203. 2012. ISBN: 978-1-107-29559-9 978-1-107-60463-6. URL: http://dx.doi.org/10.1017/CBO9781107295599 (visited on 08/24/2020) (cit. on p. 1).

[7] Roozbeh Hazrat. *Mathematicaő: A Problem-Centered Approach*. 2nd ed. 2015. Springer Undergraduate Mathematics Series. Cham: Springer International Publishing : Imprint: Springer, 2015. 1 p. ISBN: 978-3-319-27585-7. DOI: 10.1007/978-3-319-27585-7 (cit. on p. 2).

[8] *Iteration vs. Recursion - CS 61A Wiki*. Dec. 19, 2016. URL: https://www.ocf.berkeley.edu/~shidi/cs61a/wiki/Iteration_vs._recursion (visited on 08/24/2020) (cit. on p. 2).

[9] *Multi-Dimensional Arrays ů The Julia Language*. URL: https://docs.julialang.org/en/v1/manual/arrays/#man-comprehensions-1 (visited on 08/24/2020) (cit. on p. 12).

[10] A.P. Sinha and I. Vessey. "Cognitive Fit: An Empirical Study of Recursion and Iteration". In: *IEEE Transactions on Software Engineering* 18.5 (May 1992), pp. 368–379. ISSN: 00985589. DOI: 10.1109/32.135770. URL: http://ieeexplore.ieee.org/document/135770/ (visited on 08/24/2020) (cit. on p. 2).

[11] S Smolarski. *Math 60 – Notes A3: Recursion vs. Iteration*. Feb. 9, 2000. URL: http://math.scu.edu/~dsmolars/ma60/notesa3.html (visited on 08/24/2020) (cit. on p. 2).

[12] V. Anton Spraul. *How Software Works: The Magic behind Encryption, CGI, Search Engines, and Other Everyday Technologies*. San Francisco: No Starch Press, 2015. 198 pp. ISBN: 978-1-59327-666-9. URL: https://learning.oreilly.com/library/view/how-software-works/9781457189968/ (visited on 08/24/2020) (cit. on p. 1).