VALLERIE ALICIA TJOKRO

A P O R T O F O L I O

valeriealiciat@gmail.com
+62 81330659696
https://www.linkedin.com/in/valleriealiciatjokro/
https://github.com/valeriealiciat/

SEQUENCE OF FUNCTION

2020

Python, Jupyter Notebook

/ code for graphs /

```
import numpy as np
import matplotlib.pyplot as plt

#initialize ploting
fig = plt.figure(figsize=(15,15))
n=0

#loop to plot n from 0 to 6
for i in range(0,6):

    ax = fig.add_subplot(231+n,projection='3d')
    xx= np.linspace(-1,1)
    yy= np.linspace(-1,1)
    XX,YY = np.meshgrid(xx,yy)
    Z= evalFn(n,XX,YY)
    ax.plot_surface(XX, YY, Z)
    ax.set_ylabel('y')
    ax.set_xlabel('x')
    ax.set_zlabel('Fn')
    title=('F{} Surface'.format(n))
    ax.set_title(title)

    n+=1

plt.show()
```

Being one of my earliest python courseworks, it greatly revolves around the fundemental method to building an effective function: *looping*. Despite the simple task of plotting, this question has forever changed how I approach problems regarding data visualization.

The complications started when plotting numerous surface functions simultaneously.

Although the function can be plotted individually, it was very ineffective. Hence, by integrating recursions into my code, it made plotting notably more efficient especially when plotting graphs in large numbers.

This helped me to be proficient in plotting and also taught me how crucial recursions are in coding.

link to code:

https://github.com/valeriealiciat/.

/ Fn Sequence /

The sequence of functions $F_n(x, y)$ for $(n \geq 0)$ is defined by the recurrence formula

$$F_0(x, y) = 1, \quad F_1(x, y) = \frac{x}{2} - y^2$$

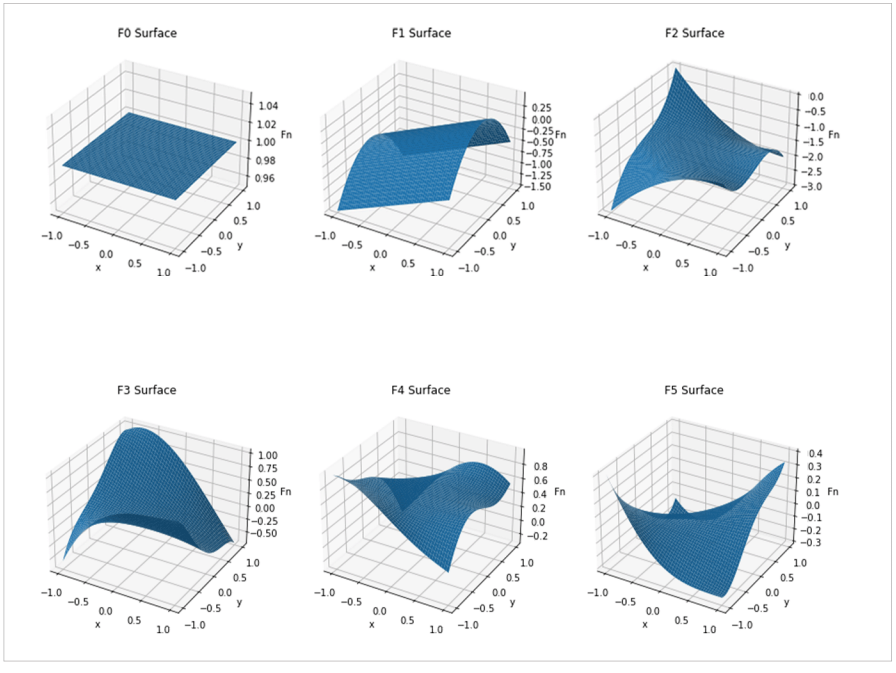
and

$$2n^2 F_{n+1}(x, y) = 2nxyF_n(x, y) - (2n + 1)F_{n-1}(x, y), \quad n \geq 1.$$

/ solution/

```
evalFn(3,np.linspace(0,1,3), np.linspace(0,1,3))
```

```
array([-0.    , -0.1875, -0.6875])
```



STATISTICAL REPORT: EFFECTS OF DELTA-9-THC (MARIJUANA)

2021

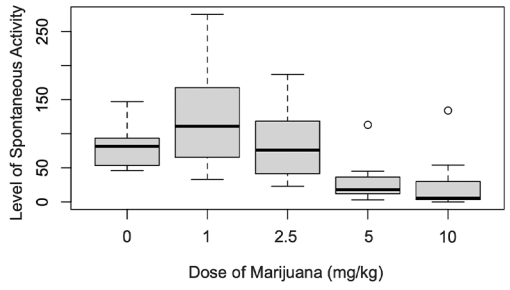
R Studio & Microsoft Word

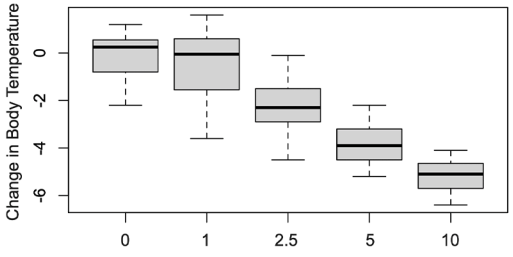
"Each data tells a story". As profound analysts, our job is to find connections and unravel the meanings hidden behind the numbers.

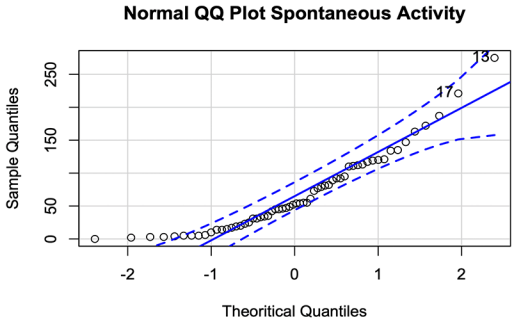
This coursework introduced me to *R studio* and how to *analyse* and *interpret statistical data* using exploratory methods such as *statistical summary* and *graphical techniques* to ultimately create an elaborate report regarding the relationship between marijuana, spontaneous activity, and temperature in mice.

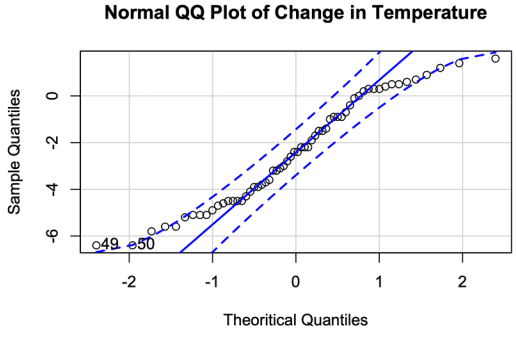
While learning the language of R studio is tough on its own, it is even more challenging to properly *understand* and translate abstract data and graphs into comprehensive paragraphs which will integrate everything neatly, and finally draw the various data and findings to a conclusion.

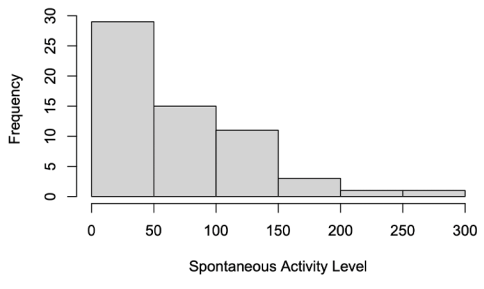
Nevertheless, with meticulous analysis, research, a lot of practice, and the help of prior experiences, I was able to find connections between all the data and altogether conclude that marijuana does indeed, cause fewer reactions and lower body temperature in mice.

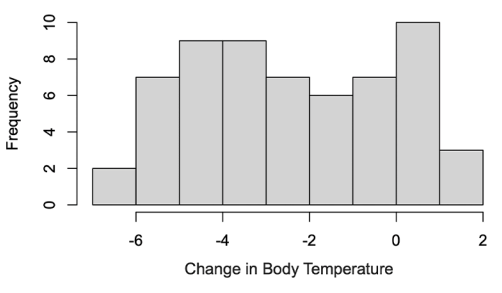












link to report and code:
https://github.com/valeriealiciat/

SYSTEM SOLVER 2021

Python, Spyder

/ Partial pivoting /

```python
def partial_pivoting(A,b,n,c):

    M=np.hstack((A,b))

    #Steps are similar to no_pivoting

    for i in range(0, c):  #we use c instead to prematurely stop the function
        for j in range(i+1, n):

            p = find_max(M,n,i)  #use find_max to detect the row with absolute maximum
            M[[i,p],:] = M[[p,i],:]  #we switch the current row with the p-th row
            factor = M[j,i] / M[i,i]

            for k in range(i, n+1):
                M[j,k] = M[j,k] - factor * M[i,k] #row operation

    return M


def partial_pivoting_solve (A ,b , n ):

    M = partial_pivoting(A,b,n,n)
    return bw.backward_substitution(M, n) #we use backward.py to solve the matrix after
                    #Gaussian elimination with partial pivotiing
```

/ Gauss Seidel /

```python
def Gauss_Seidel(A,b,n,x0,tol,maxits):

    #lower triangular matrix from A
    L = np.tril(A)

    #upper triangular matrix
    U = A - L

    #to initialize while loop
    it=1

    #we stop when we reach maxits
    while it<=maxits: #we stop when we reach maxits

        #apply the formula for gauss seidel
        x0 = np.dot(np.linalg.inv(L), b - np.dot(U, x0))
        it+=1

        if np.max(np.abs(b-np.matmul(A,x0))) < tol:
            return x0

    if np.max(np.abs(b-np.matmul(A,x0))) > tol:
        x0=(" Desired tolerance not reached after maxits iterations have been performed.")
        return x0
```

The system solver coursework consists of functions that solve augmented matrices such as Gauss Seidel, parrtial pivoting, no pivoting and Doolittle factorization.

In most cases, working with matrices means that we are dealing with a large number of data at the same time, which can be extremely complex to handle.

As expected, I struggled with operating that much data effectively. But I figured that to increase the efficiency of the code, *loops integrated with array indexing* should be used to operate the matrix. With deliberate visualization, I was able to tackle the complications regarding matrix operations.

Through this coursework, I developed strong 2D visualization and also became proficient in working with indexing and matrix iterations.

/ outputs /

```
Gauss Seidel 2 test output:
[[12.99917603]
 [ 3.99979401]
 [ 6.9999485 ]]
```

```
Partial Pivoting 1 test output:
[[ 10.    0.   20.    6. ]
 [  0.   -5.   -1.    6.4]
 [  0.   10.  -11.    1. ]]
```

```
Partial Pivoting Solve test output:
[[ 2.72307692]
 [-1.06769231]
 [-1.06153846]]
```

```
No Pivoting 2 test output:
[[  1.   -5.    1.    7. ]
 [  0.   50.   10.  -64. ]
 [  0.    0.  -13.   13.8]]
```

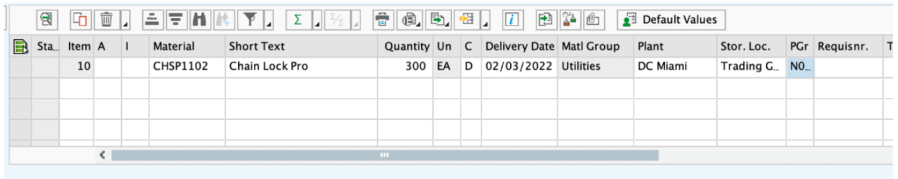
```
No Pivoting Solve test output:
[[ 2.72307692]
 [-1.06769231]
 [-1.06153846]]
```

```
U:                  L:
[[ 1.   1.   0.] [[1. 0. 0.]
 [ 0.  -1.  -1.]  [2. 1. 0.]
 [ 0.   0.   0.]] [0. 1. 1.]]
```

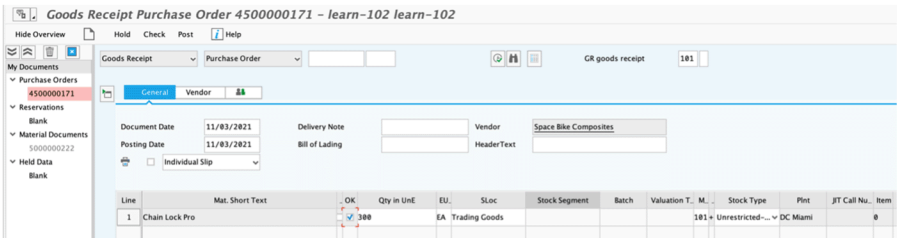
link to code:
https://github.com/valeriealiciat/

SAP (ERP) & Microsoft Word

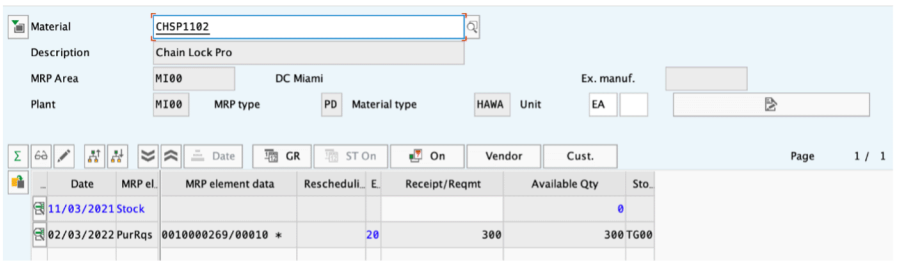
/ purchase requisition /



/ goods receipt /



/ goods specification /



link to report:
https://github.com/valeriealiciat/

Realizing the complexity of coding, it is an arduous task for a single person to create an intricate software. In order to achieve such great feats, a team is needed.

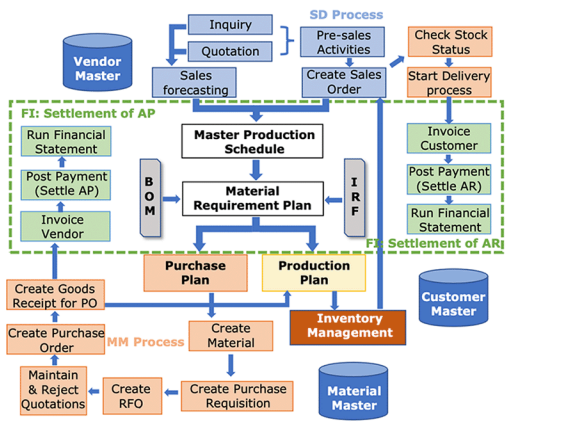
This report is a compilation of the three business processes: material procurement (MM), delivery of goods to consumers (SD) and financial accounting (FI) using SAP(ERP), which was not a small task and thus, a team was made to complete the report.

Asides the difficulties of familiarizing ourselves with the ERP system, we also realize that team coordination was a struggle as we were in different locations around the world and each have different opinions and distinct skill sets.

However, with proper communication, understanding and cooperation, we were able to help one another, cover for what each other lacked, work effectively and ultimately settle conflicts and resolve problems.

Asides from learning how to learn and adapt to the system of SAP(ERP) quickly, I learned the importance of a team, how members complement each other's strengths, and how working together immensely improves efficiency. After all, it is true that teamwork makes the dream work.

1. Flow Chart of SD, MM and FI processes integration in SAP



POLYNOMIAL INTERPOLATION 2022

Python, Spyder

Polynomial interpolation: a curve fitting method

Without doubt, this scientific computation coursework about different kinds of polynomial interpolations (uniform, non-uniform, piecewise) remains one of the most challenging as it requires rigorous *problem solving* as well as *critical thinking*.

The first challenge arises when composing nodal interpolation points which should be non-uniformly spaced without any guidance.

To solve this, I unexpectedly had to use the *linear formula* M(x)= cx+d, where c and d are to be determined, to map the intervals.

Considering the given code execution time limit of *1 minute* to evaluate *2000 points n times* to compute the error (with p=10), the second challenge quickly emerges.

The first trial took more than *2 hours* to execute. Thus, in order to immensely cut the computation time, I decided to use *list comprehensions* for effective recursions and *arrays* to store the required data.

This method successfully reduced the run time drastically by almost *120x*, from *2 hours to less than 1 minute*.

Indeed, a lot of solutions to a code are out of the ordinary. Nevertheless, that is exactly why I find the art of coding to be so compelling.

/ non uniform polynomial interpolation /

```
def nonuniform_poly_interpolation(a,b,p,n,x,f,produce_fig):

    #Create empty lists to contain results of evaluation.
    L=[]

    #Use given xhat formula.
    xh = [ np.cos( ( (2*k+1) / (2*(p+1)) ) * np.pi ) for k in range (0,p+1) ]

    #Calculate mapped xhat
    xhat = [ ( (b-a) / 2 ) * x + ( (a+b) / 2 ) for x in xh]

    #Store lagrange polynomial results
    lp = lagrange_poly(p, xhat, n, x, 1e-10)[0]

    #Evaluate F by values in xhat
    F = [ f(x) for x in xhat ]


    #Start loops
    for j in range( n ):

        #po value is resetted to 0 every j loop
        po=0


        #Calculate interpolant
        for i in range( p+1 ):

            #Use interpolation formula
            po += F[ i ] * lp[ i,j ]

        #Store results
        L.append(po)

    nu_interpolant = np.array(L)
```

/ compute errors /

```
def compute_errors(a,b,n,P,f):

    #Compute evaluation points
    data = np.linspace(a,b,2000)


    #Calculate f(x)
    mat = [f(d) for d in data]


    #Calculate error for points in 'data' for both uniform and non-uniform polynomial
    #interpolation
    eu = [ max ( np.abs( mat -
                        uniform_poly_interpolation(a,b,P[i],2000,data,f,False)[0] ) )
                                for i in range(n) ]
    enu = [ max( np.abs( mat -
                        nonuniform_poly_interpolation(a,b,P[i],2000,data,f,False)[0] ) )
                       for i in range(n) ]

    #Combine arrays containing calculated errors
    error_matrix= np.vstack((eu,enu))


    #Initialize plotting
    fig = plt.figure()

    #set plot title, labels, and legend
    plt.title('Lagrange Interpolation Errors')

    plt.xlabel('p')

    plt.ylabel('error')

    #plot using plt.semilogy
    plt.semilogy(P , error_matrix[0,:], label='Uniform Polynomial Interpolation')

    plt.semilogy(P , error_matrix[1,:], label='Non Uniform Polynomial Interpolation')

    plt.legend()

    return error_matrix, fig
```

link to code:
https://github.com/valeriealiciat/