Zewail City of Science and Technology
University of Science and Technology
Math 404 - fall 2019

Report 2

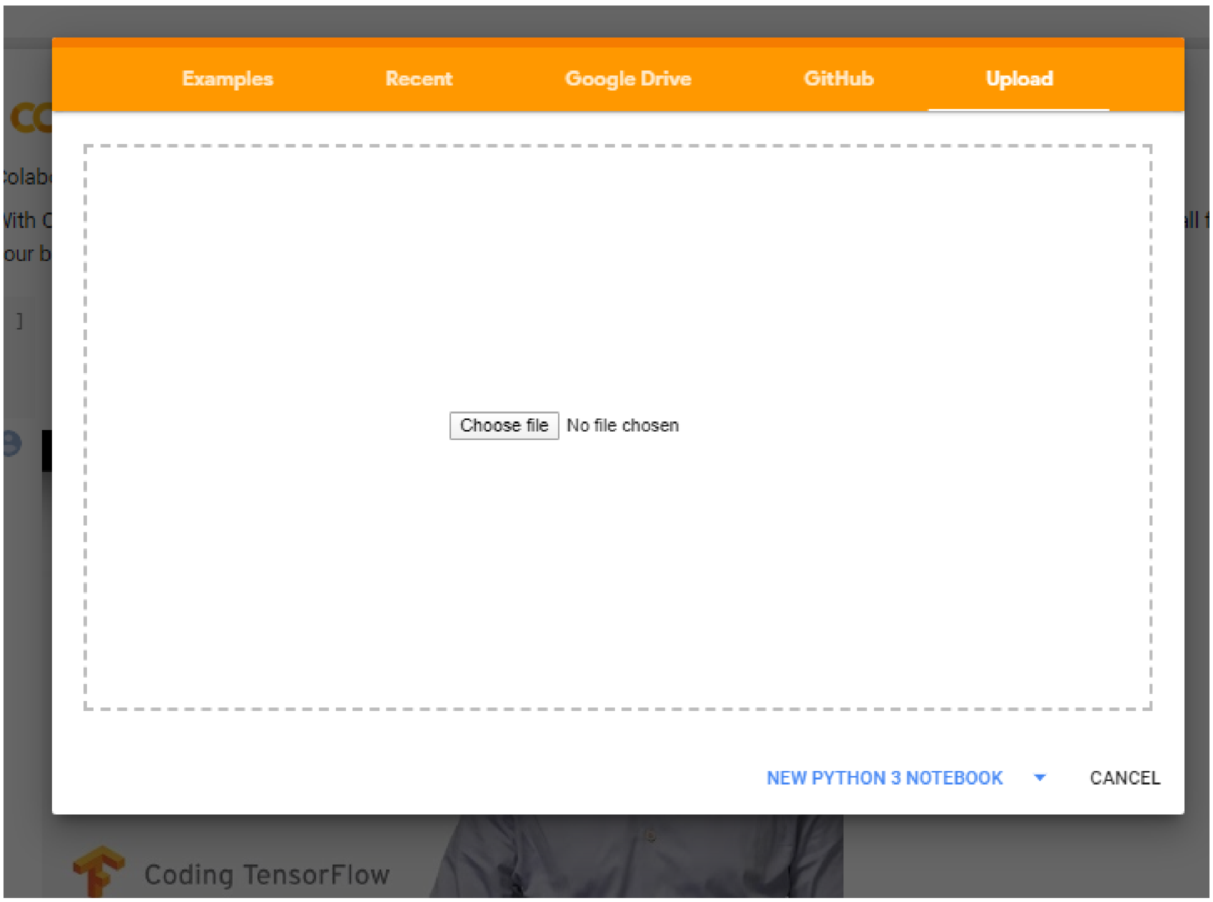
By:

Alaa Hesham 201500638

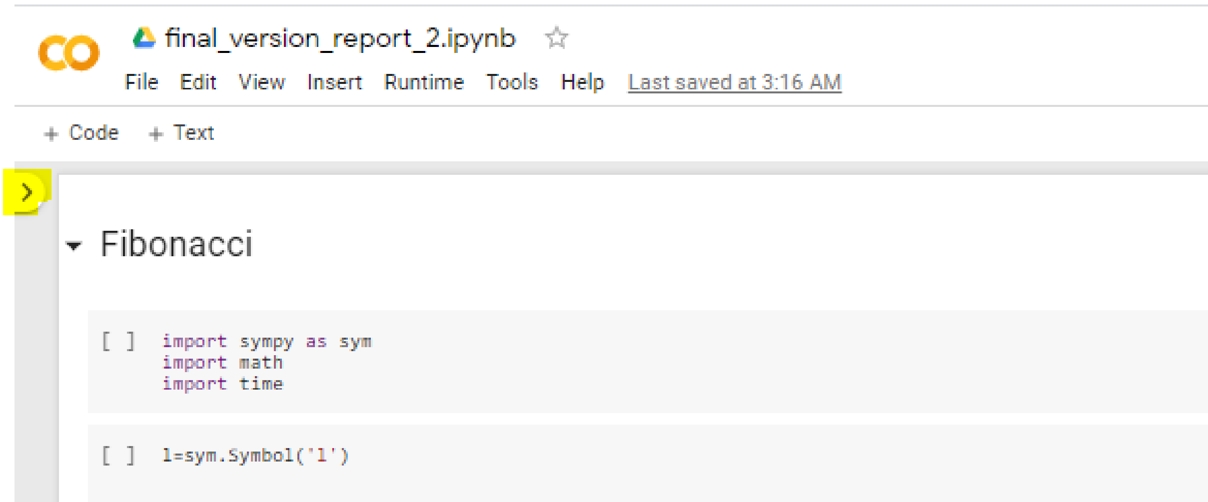
1| How to run code

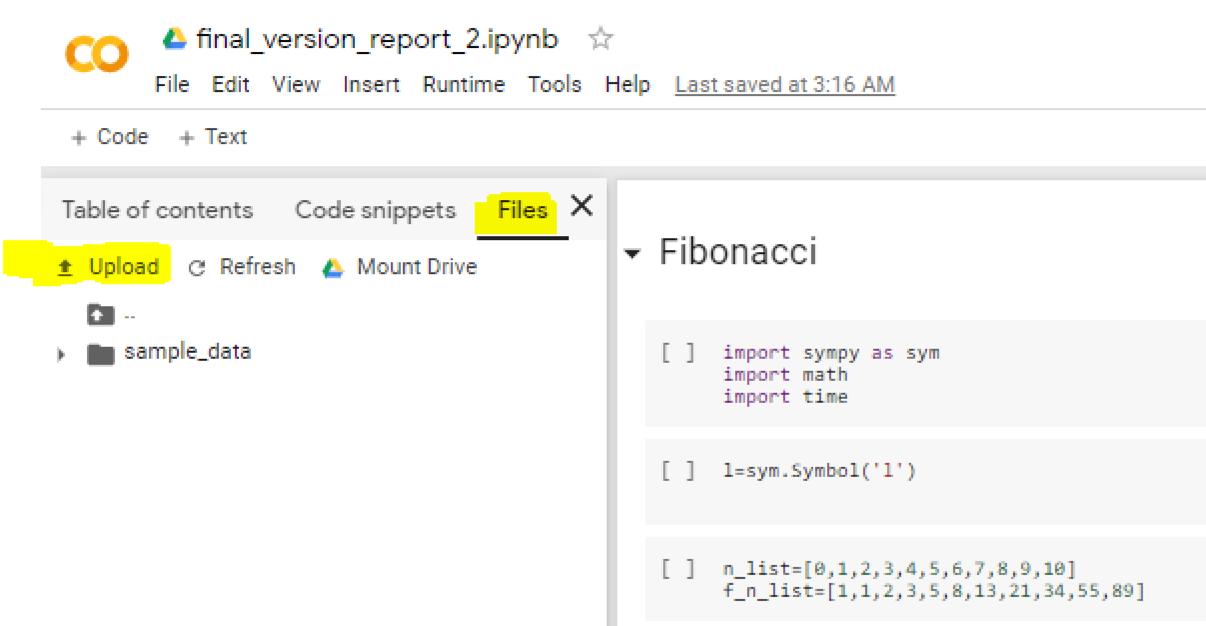
It is a python notebook (.ipynb).

- The easiest way to open it is through [colab]
- Click upload button to upload it



- I have inserted images in the notebook, to view everything in notebook. You are going to need to upload images folder, the way to do that is to click on side arrow marked by yellow.

- 
final_version_report_2.ipynb
File Edit View Insert Runtime Tools Help Last saved at 3:16 AM
+ Code + Text
Fibonacci
import sympy as sym
import math
import time
l=sym.Symbol('l')


final_version_report_2.ipynb
File Edit View Insert Runtime Tools Help Last saved at 3:16 AM
+ Code + Text
Table of contents Code snippets Files
Upload Refresh Mount Drive
sample_data
Fibonacci
import sympy as sym
import math
import time
l=sym.Symbol('l')
n_list=[0,1,2,3,4,5,6,7,8,9,10]
f_n_list=[1,1,2,3,5,8,13,21,34,55,89]

2 | Benchmark Problem

I have chosen to start with initial point as assignment X_0=[-1 1] INSTEAD OF X_0=[-1.2,1] in order to be able to verify my solution with assignment solution

1. Consider the **Rosenbrock's parabolic valley** function:

$$f(x) = 100\left(x_2 - x_1^2\right)^2 + \left(1 - x_1\right)^2$$

Using $\boldsymbol{x_0} = \begin{bmatrix}-1 & 1\end{bmatrix}^T$ as a starting point, find the minimum of $f(x)$ along the direction $\boldsymbol{s_1} = \begin{bmatrix}4 & 0\end{bmatrix}^T$ using:

a) The Fibonacci method with $L_0 = (0, 0.1)$ up to 2 decimal places.
b) The golden section method with $L_0 = (0, 0.1)$ up to 2 decimal places.
c) The quadratic interpolation method. Use a maximum of two refits.
d) The cubic interpolation method. Use a maximum of two refits.

3| Fibonacci

How to test Fibonacci code?

There is a title in notebook to guide you where to put your input as in the following image .You should enter

- Your function in terms of lambda. Note that in python ** represents power. For example $3X^2$ will be written as 3*X**2 in python.
- Interval ending b named as org_b , Interval beginning a named as org_a
- Epsilon
- Mini is a flag to determine whether the problem is minimization or maximization. If it is 1, it will be minimization.

How to test Fibonacci code ?

```
In [79]: sfTime=timeit.default_timer()

         f_l=25600*l**4-25600*l**3+6416*l**2-16*l+4 # f(lambda)

         org_b=0.1                                  # end of region b , org stands for original i.e. b value at the end of region
         org_a=0                                    # start of region a
         epsilon=0.01

         mini=1                                     # mini stands for minimization if problem of type minimization it will equal 1
                                                    # if not it will equal to zero
         estimated_lamda,reduction_ratio,n=fiboancci_main(org_b,org_a,epsilon,length_fibonacci_list,mini)

         fTime=timeit.default_timer() - sfTime
```

Figure 1 Fibonacci testing

```
In [81]: reduction_ratio

Out[81]: 7.69

In [82]: optimal_value=f_l.evalf(subs={l:estimated_lamda})

In [83]: print ("Fibonacci : Number of iterations",n)
         print ("Fibonacci : Optimal solution /Lambda star",estimated_lamda)
         print("Fibonacci :  Optimal value",optimal_value)
         print ("Fibonacci : CPU Time",fTime)

         Fibonacci : Number of iterations 6
         Fibonacci : Optimal solution /Lambda star 0.00385
         Fibonacci :  Optimal value 4.03204587888656
         Fibonacci : CPU Time 0.02207130199894891
```

Figure 2 Fibonacci results

4 | Golden Section

It will be tested as Fibonacci .

How to test Golden section code ?

```
In [21]:
f_l=25600*l**4-25600*l**3+6416*l**2-16*l+4
org_b=0.1
org_a=0
epsilon=0.01
mini=1
estimated_lamda,reduction_ratio=golden_main(org_b,org_a,epsilon,mini)
```

Figure 3 golden section testing

Calculations

```
In [90]: optimal_value=f_l.evalf(subs={l:estimated_lamda})
```

```
In [91]: print ("Golden Number of iterations",n)
         print ("Golden Optimal solution /Lambda star",estimated_lamda)
         print("Golden Optimal value",optimal_value)
         print ("Golden CPU Time",golden_Time)

Golden Number of iterations 6
Golden Optimal solution /Lambda star 0.00451
Golden Optimal value 4.05600428623790
Golden CPU Time 0.2974034020007821
```

Figure 4 golden section results

5 | Quadratic interpolation

How to test quadratic interpolation?

Type f_l (f (lambda) as in the previous methods, and determine t_o

How to test quadratic interpolation

```
In [99]: q_I_startTime=timeit.default_timer()

         f_l=25600*l**4-25600*l**3+6416*l**2-16*l+4
         t_o = 0.001
         quadratic_interpolation_main(f_l,t_o,1)


         q_I_time =timeit.default_timer()- q_I_startTime
```

```
]: print ("Quadratic interpolation : Number of iterations",counter)
   print ("Quadratic interpolation : Optimal solution /Lambda star",s_lambdaa)
   print("Quadratic interpolation :  Optimal value",fs_lambdaa)
   print ("Quadratic interpolation : CPU Time",q_I_time)

   Quadratic interpolation : Number of iterations 1
   Quadratic interpolation : Optimal solution /Lambda star 0.00125792708535247
   Quadratic interpolation :  Optimal value 3.98997482706145
   Quadratic interpolation : CPU Time 0.0046636689985462
```

Figure 5 Quadratic interpolation

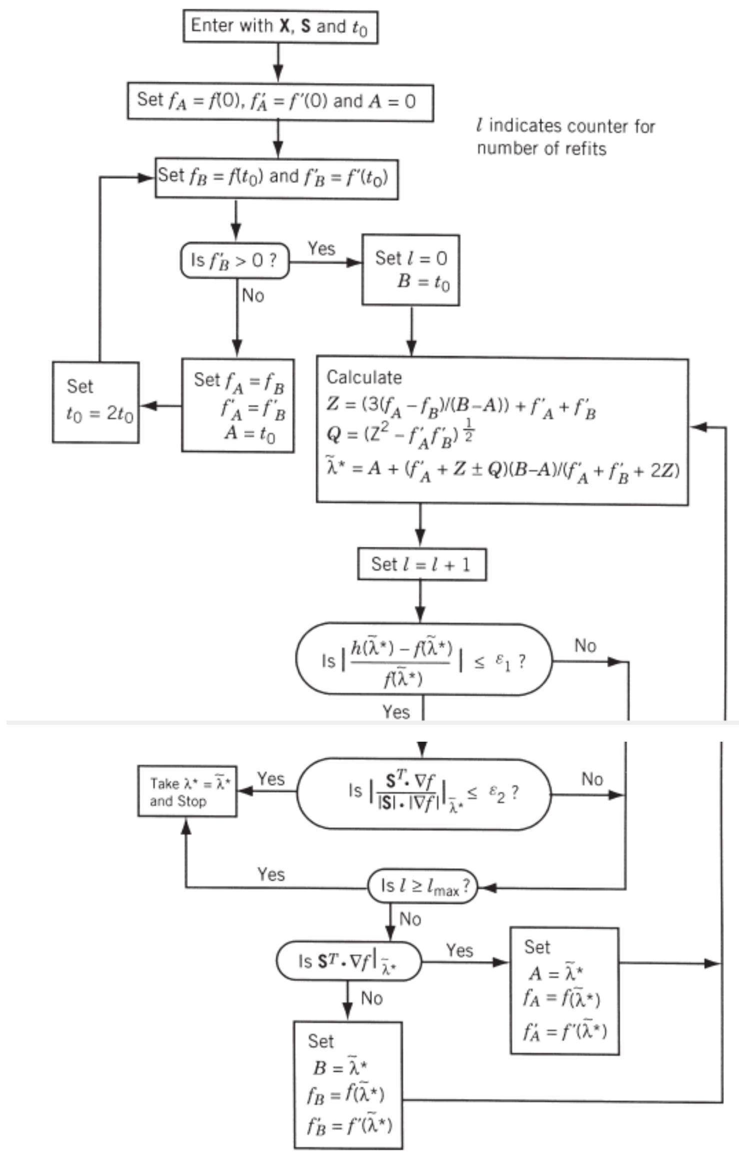
6| Comparison

I can see that quadratic interpolation is better than fibonacci and golden section in terms of number of iterations , optimal value ,and CPU time .

Since I did not manage to implement following algorithms due to messy circumstances, I have inserted their algorithm/pesudocode.

7 | Cubic interpolation

Pseudo code



8 |Fletcher Reeves

Algorithm

1. **Start with an arbitrary initial point x_1.**
2. **Set the first search direction $s_1 = -\nabla f_1$.**
3. **Find the point x_2 as $x_2 = x_1 + \lambda_1^* s_1$**
   where λ_1^* is the optimal step length in the direction s_1. Set $i = 2$.
4. **Find $\nabla f_i = \nabla f(x_i)$, and set**
$$s_i = -\nabla f_i + \frac{\|\nabla f_i\|_2^2}{\|\nabla f_{i-1}\|_2^2} s_{i-1}$$
5. **Compute the optimum step length $\lambda_i^* = \frac{\|\nabla f_i\|_2^2}{s_i^T A s_i}$ in the direction s_i, and find the new point $x_{i+1} = x_i + \lambda_i^* s_i$.**
6. **Test the optimality of the point x_{i+1} . If it is optimum, stop the process. Otherwise, set $i = i + 1$ and go to step 4.**

9 | Marquardt Method

Given: an initial point $\boldsymbol{x_1}$ and constants $\boldsymbol{\alpha_1}$ (on the order of 10^4), c_1 ($0 < c_1 < 1$), c_2 ($c_2 > 1$), and ε (on the order of 10^{-2}).

- **Step 1: $\boldsymbol{x = x_1, i = 1}$**
- **Step 2:** compute $\boldsymbol{\nabla f_i = \nabla f(x_i)}$.
- **Step 3:** check for optimality: if $\|\boldsymbol{\nabla f_i}\| < \boldsymbol{\varepsilon}$, then stop.
- **Step 4:** $\boldsymbol{x_{i+1} = x_i + \lambda_i^* s_i = x_i - \lambda_i^* \left[\nabla^2 f_i + \alpha_i I\right]^{-1} \nabla f_i}$
- **Step 5:** if $\boldsymbol{f_{i+1} < f_i}$, go to Step 6, else go to Step 7.
- **Step 6:** $\boldsymbol{\alpha_{i+1} = c_1 \alpha_i}$,$\boldsymbol{i = i + 1}$, go to Step 2.
- **Step 7:** $\boldsymbol{\alpha_{i+1} = c_2 \alpha_i}$, go to Step 4.

9 |Quasi Newton BFGS

Algorithm

Given starting point x_0, convergence tolerance $\epsilon > 0$,
inverse Hessian approximation H_0;
$k \leftarrow 0$;
while $\|\nabla f_k\| > \epsilon$;
Compute search direction

$$p_k = -H_k \nabla f_k;$$

Set $x_{k+1} = x_k + \alpha_k p_k$ where α_k is computed from a line search procedure to satisfy the Wolfe conditions (3.6);
Define $s_k = x_{k+1} - x_k$ and $y_k = \nabla f_{k+1} - \nabla f_k$;
Compute H_{k+1} by means of (6.17);
$k \leftarrow k + 1$;
end (while)