```
#   Authored by Christopher Allred
#   A02233404
#   Numerical Methods
#   Spring 2020
```

PROBLEM 1.a

1. Linear algebraic equations can arise in the solution

of differential equations. For example, the following heat

equation describes the equilibrium temperature

T = T(x)(oC)

at a point x (in meters m) along a long thin **rod**,

$$\frac{d^2T}{dx^2} = h'(T - T_a), \tag{1}$$

$$\frac{d^2T}{dx^2} = \mathrm{h}'(\mathrm{T} - \mathrm{T_a}),$$

d2T/dx2 = h′(T − Ta), (1)

where Ta = 10oC denotes the temperature of the surrounding

air, and h′ = 0.03 (m-2) is a heat transfer coefficient. Assume

that the rod is 10 meters long (i.e. 0 ≤ x ≤ 10) and has

boundary conditions imposed at its ends given by T(0) = 20oC

and T(10) = 100oC.

a) Using standard ODE methods, which you do not need to repeat here,

the general form of an analytic solution to (1) can be derived as

$$T(x) = A + Be^{\lambda x} + Ce^{-\lambda x}, \tag{2}$$

T(x)=A+Beλx +Ce-λx, (2)

where A, B, C, and λ are constants. Plug the solution of type (2)

into both sides of equation (1). This should give you an equation

that must be satisfied for all values of x, for 0 ≤ x ≤ 10, for some

fixed constants A, B, C, and λ. Analyze this conclusion to determine what the values of **A and λ** must be.

''' NO COMPUTER CODE REQUIRED

```
1m                    5 m                   10 m
|                     |                     |
^                                           ^
20oC                                        100oC
```

$T := A + B\cdot\exp(\text{lambda}\cdot x) + C\cdot\exp(-\text{lambda}\cdot x)$

$$T := A + B\,e^{\lambda x} + C\,e^{-\lambda x} \tag{1}$$

$\frac{d^2}{dx^2}T = h\_prime\cdot(T - Ta)$

$$B\lambda^2 e^{\lambda x} + C\lambda^2 e^{-\lambda x} = h\_prime\left(A + B\,e^{\lambda x} + C\,e^{-\lambda x} - Ta\right) \tag{2}$$

$\lambda^2\left(B\cdot e^{\lambda x} + C\cdot e^{-\lambda x}\right) = h\_prime\,(T - Ta)$

$$\lambda\left(B\,e^{\lambda x} + C\,e^{-\lambda x}\right)^2 = h\_prime\left(A + B\,e^{\lambda x} + C\,e^{-\lambda x} - Ta\right) \tag{3}$$

$T - A = B\,e^{\lambda x} + C\,e^{-\lambda x}$

$$B\,e^{\lambda x} + C\,e^{-\lambda x} = B\,e^{\lambda x} + C\,e^{-\lambda x} \tag{4}$$

$\lambda^2\cdot(T - A) = h\_prime\,(T - Ta)$

$$\lambda^2\left(B\,e^{\lambda x} + C\,e^{-\lambda x}\right) = h\_prime\left(A + B\,e^{\lambda x} + C\,e^{-\lambda x} - Ta\right) \tag{5}$$

#therfore

$\lambda^2 = h\_prime$

$$\lambda^2 = h\_prime \tag{6}$$

$A = Ta$

$$A = Ta \tag{7}$$

Found Lambda and A

PROBLEM 1.b

```
'''
(b) Next, impose the boundary conditions T (0) = 20 oC and
    T (10) = 100 oC to derive a system of 2 linear algebraic equations
    for B and C. Provide the system of two equations you have derived.
''' NO COMPUTER CODE REQUIRED
```

$Ta := 10$

$$Ta := 10 \quad (8)$$

$h\_prime := 0.3$

$$h\_prime := 0.3 \quad (9)$$

$Ta + B + C = 20$

$$10 + B + C = 20 \quad (10)$$

$B + C = 20 - Ta$ # Equation 1

$$B + C = 10 \quad (11)$$

$Ta + B \cdot \exp(h\_prime \cdot 10) + C \cdot \exp(-h\_prime \cdot 10) = 100$

$$10 + 20.08553692\, B + 0.04978706837\, C = 100 \quad (12)$$

$\exp(h\_prime \cdot 10) + C \cdot \exp(-h\_prime \cdot 10) = 100 - Ta$ # equation 2

$$20.08553692 + 0.04978706837\, C = 90 \quad (13)$$

The system of two equations:

$$Mat1 := \begin{bmatrix} 1 & 1 & 10 \\ e^{3} & e^{-3} & 90 \end{bmatrix}$$

PROBLEM 1.c

```
'''
(c) Use one of the numerical algorithms you developed for
    homework 3 (Gauss elimination or LU decomposition) to solve the
    algebraic system you derived in question 2(b) above, and obtain an
    analytic solution to (1) of the form (2). By analytic solution we mean
    an explicit solution to equation (1) which is valid for each x in
    the interval [0, 10].
'''
```

```
Gauss elimination
#  Authored by Christopher Allred
```

```python
#  A02233404
#  Numerical Methods
#  Spring 2020
import numpy as np
import math

TOLERANCE = 0.0001

def substitute(a, b):
    n = len(a)
    aCopy = a[:]
    x = [None]*(n)
    # x[n] = b[n]/aCopy[n][n]
    for i in range(n-1,-1,-1):
        # sum = b[i]
        x[i] =aCopy[i][n]/aCopy[i][i]
        for j in range( i-1,-1,-1 ):
            # sum = sum + aCopy[i][j]*x[j]
            aCopy[j][n] -= aCopy[j][i] *x[i]

    return x



def guss2(a, b):
    print(np.matrix(a))
    print("Solution:...")
    n = len(a)-1
    m = len(a[1])-1
    er = 0

    # get the Max val in each row
    s = [None] * (n+1)
    for i in range(0,n+1):
        s[i] = abs(a[i][0])
        for j in range( 1,n+1):
            if abs(a[i][j]) > s[i]:
                s[i] = abs(a[i][j])



    #Elimination:
    for k in range(0,n+1):
        # print("k:" +str(k))
        #Pivot:
```

```
        p = k
        big = abs(a[k][k]/s[k])
        for ii in range(k+1,n+1):
            dummy = abs(a[ii][k]/s[ii])
            if dummy > big:
                big = dummy
                p = ii

        # Pivot
        if p != k:
            for jj in range( k,n+1):
                dummy = a[p][jj]
                a[p][jj] = a[k][jj]
                a[k][jj] = dummy

            dummy = b[p]
            b[p] = b[k]
            b[k] = dummy
            dummy = s[p]
            s[p] = s[k]
            s[k] = dummy


        if abs(a[k][k]/s[k]) < 0:
            er =- 1
            break #EXIT FOR

        for i in range(k+1,n+1):

            factor =- a[i][k]/a[k][k]
            for j in range (k,m+1):

                a[i][j] = a[i][j]+factor*a[k][j]

            b[i] = b[i]+factor*b[k]

    if abs(a[n][n]/s[n]) < 0:
        er = -1
    print(np.matrix(a))
    print("Coefficients: " + str(np.matrix(b)))
    # Elimination
    if er != -1:

        #Substitute:
        vals = substitute(a,b)
```

```python
        print("Solution Values: "+ str(vals))
        return vals



if __name__ == "__main__":

    A1 =[[1,  1, 10],
        [math.exp(3) , math.exp(-3),  90]]

    bConsts1 = [10,90]

    guss2(A1,bConsts1)
```

Terminal Output

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 2: Python Debug Consc

```
PS C:\Users\Christopher\Documents\GitHub\NumMethods\HW\project1> ${env:PTVSD_LAUNCHER_PORT}='54259'; & 'C:\Users\Christopher\AppData\Local\Programs\Python\Python37-32\python.exe' 'c:\Users\Christopher\.vscode\extensions\ms-python.python-2020.2.63990\pythonFiles\lib\python\new_ptvsd\wheels\ptvsd\launcher' 'c:\Users\Christopher\Documents\GitHub\NumMethods\HW\project1\NM_HW3.py'
[[1.00000000e+00 1.00000000e+00 1.00000000e+01]
 [2.00855369e+01 4.97870684e-02 9.00000000e+01]]
Solution:...
[[   1.            1.           10.        ]
 [   0.          -20.03574985 -110.85536923]]
Coefficients: [[  10.         -110.85536923]]
Solution Values: [4.467121518528577, 5.532878481471423]
PS C:\Users\Christopher\Documents\GitHub\NumMethods\HW\project1>
```

⊗ 0 ⚠ 0 ▷ Python: Current File (project1) Ln 160, Col 20 Spaces: 4 UTF-8 CRLF Python

$$\begin{bmatrix} \dfrac{10\left(-9+e^{-3}\right)}{-e^{3}+e^{-3}} \\[2ex] -\dfrac{10\left(e^{3}-9\right)}{-e^{3}+e^{-3}} \end{bmatrix}$$

Solution Values: [4.467121518528577, 5.532878481471423]
A = 4.467121518528577
B = 5.532878481471423

```
'''
(d) Next we will discuss how to obtain a numerical solution to (1).
    That is, we will seek to obtain an approximate solution to (1) which
    describes the value of T at 9 intermediate points inside the interval
    [0,10]. More precisely, the equation (1) can be transformed into a
    linear algebraic system for the temperature at 9 interior points
T1 = T(1),
 T2 = T(2),
 T3 = T(3),
 T4 =T(4),
 T5 =T(5),
 T6 =T(6),
 T7 =T(7),
 T8 =T(8),
 T9 =T(9),
    by
    using the following finite difference approximation for the second
    derivative at the ith interior point,
```

$$\frac{d^2T_i}{dx^2} = \frac{T_{i+1} - 2T_i + T_{i-1}}{(\Delta x)^2}, \qquad (3)$$

```
d2Ti/ dx2 = (Ti+1 -2Ti +Ti-1)/ (Δx)**2,                      (3)

    where
```

1≤i≤9,

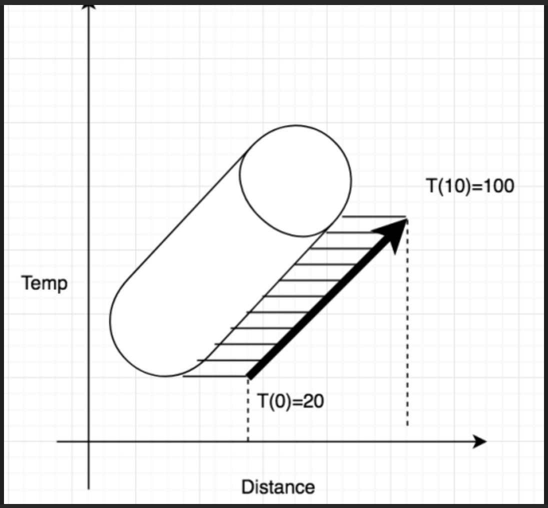
T0 =T(0) =20oC,

T10=T(10)=100oC,

```
    and Δx is the equal spacing between consecutive interior points
    (i.e. with 9 equally spaced interior points inside [0,10] it holds
    that Δx = 1). Use (3) to rewrite (1)
```

$$\frac{d^2T}{dx^2} = h'(T - T_a), \qquad (1)$$

```
      as a system of 9 linear algebraic
    equations for the unknowns T1, T2, T3, T4, T5, T6, T7, T8, and T9.
    Provide the system of 9 equations you have derived.
''' NO COMPUTER CODE REQUIRED
```


T(10)=100
Temp
T(0)=20
Distance

$T := x \rightarrow A + B \cdot \exp(\text{lambda} \cdot x) + C \cdot \exp(-\text{lambda} \cdot x)$

$$T := x \rightarrow A + B\,e^{\lambda x} + C\,e^{-\lambda x} \quad (1)$$

$\frac{d^2}{dx^2} T = h\_prime \cdot (T - Ta)$

$$0 = \frac{3}{10} T - 3 \quad (2)$$

$\frac{d^2}{dx^2} T = \frac{T[i+1] - 2\,T[i] + T[i-1]}{(DeltaX^2)}$

$$0 = T_{12} - 2\,T_{11} + T_{10} \quad (3)$$

$n = 10$

$$n = 10 \quad (4)$$

$DeltaX := 1$

$$DeltaX := 1 \quad (5)$$

$Tmat := \begin{bmatrix} \text{1 .. 11 Vector[column]} \\ \text{Data Type: anything} \\ \text{Storage: rectangular} \\ \text{Order: Fortran\_order} \end{bmatrix}$

$$Tmat := \begin{bmatrix} \text{1 .. 11 Vector}_{column} \\ \text{Data Type: anything} \\ \text{Storage: rectangular} \\ \text{Order: Fortran\_order} \end{bmatrix} \quad (6)$$

_CMRTS:=interface(rtablesize = ∞) : **(6)**; interface(rtablesize = _CMRTS) :

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad (7)$$

for i **from** 1 **by** 1 **while** $i \leq 10$**do**

$$Tmat[i] := \frac{T(i) - 2\,T(i-1) + T(i-2)}{(DeltaX^2)} - h\_prime\cdot(T(i-1) - Ta) = 0$$

end do:

$Tmat[1] := 20 = T(0)$

$$Tmat_1 := 20 = T(0) \tag{9}$$

$Tmat[11] := 100 = T(10)$

$$Tmat_{11} := 100 = T(10) \tag{10}$$

$Tmat$

$$\left[\begin{array}{c} \textit{1 .. 11 Vector}_{column} \\ \textit{Data Type: anything} \\ \textit{Storage: rectangular} \\ \textit{Order: Fortran\_order} \end{array}\right] \tag{11}$$

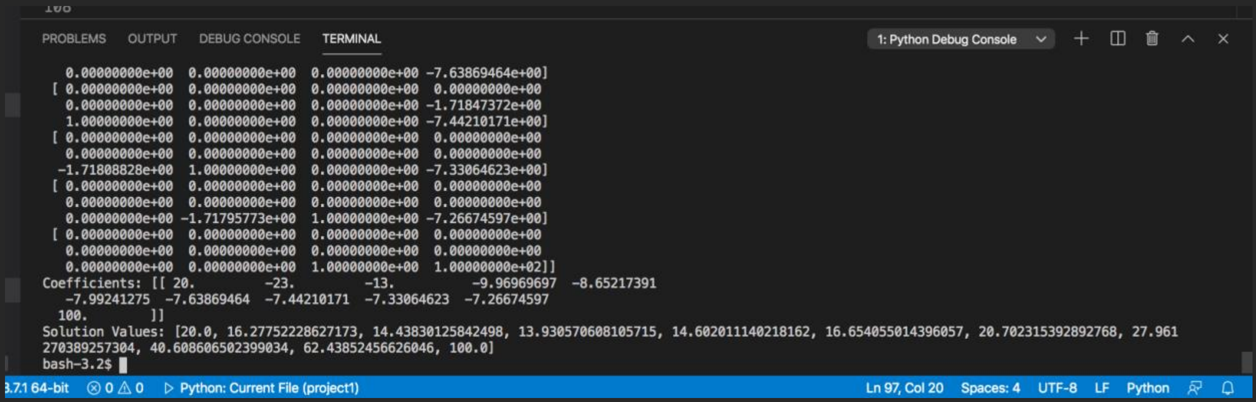
$\_CMRTS := interface(rtablesize = \infty)$: **(11)**; $interface(rtablesize = \_CMRTS)$:

$$\left[\begin{array}{c} 20 = T(0) \\ T(2) - \frac{23}{10}\,T(1) + T(0) + 3 = 0 \\ T(3) - \frac{23}{10}\,T(2) + T(1) + 3 = 0 \\ T(4) - \frac{23}{10}\,T(3) + T(2) + 3 = 0 \\ T(5) - \frac{23}{10}\,T(4) + T(3) + 3 = 0 \\ T(6) - \frac{23}{10}\,T(5) + T(4) + 3 = 0 \\ T(7) - \frac{23}{10}\,T(6) + T(5) + 3 = 0 \\ T(8) - \frac{23}{10}\,T(7) + T(6) + 3 = 0 \\ T(9) - \frac{23}{10}\,T(8) + T(7) + 3 = 0 \\ T(10) - \frac{23}{10}\,T(9) + T(8) + 3 = 0 \\ 100 = T(10) \end{array}\right] \tag{12}$$

$$\left[\begin{array}{c}
20 = T0 \\[2ex]
T2 - \frac{23}{10}\,T1 + T0 + 3 = 0 \\[2ex]
T3 - \frac{23}{10}\,T2 + T1 + 3 = 0 \\[2ex]
T4 - \frac{23}{10}\,T3 + T2 + 3 = 0 \\[2ex]
T5 - \frac{23}{10}\,T4 + T3 + 3 = 0 \\[2ex]
T6 - \frac{23}{10}\,T5 + T4 + 3 = 0 \\[2ex]
T7 - \frac{23}{10}\,T6 + T5 + 3 = 0 \\[2ex]
T8 - \frac{23}{10}\,T7 + T6 + 3 = 0 \\[2ex]
T9 - \frac{23}{10}\,T8 + T7 + 3 = 0 \\[2ex]
T10 - \frac{23}{10}\,T9 + T8 + 3 = 0 \\[2ex]
100 = T10
\end{array}\right]$$

$$
\left[\begin{array}{cccccccccccc}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 20 \\
1 & -\frac{23}{10} & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -3 \\
0 & 1 & -\frac{23}{10} & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -3 \\
0 & 0 & 1 & -\frac{23}{10} & 1 & 0 & 0 & 0 & 0 & 0 & 0 & -3 \\
0 & 0 & 0 & 1 & -\frac{23}{10} & 1 & 0 & 0 & 0 & 0 & 0 & -3 \\
0 & 0 & 0 & 0 & 1 & -\frac{23}{10} & 1 & 0 & 0 & 0 & 0 & -3 \\
0 & 0 & 0 & 0 & 0 & 1 & -\frac{23}{10} & 1 & 0 & 0 & 0 & -3 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & -\frac{23}{10} & 1 & 0 & 0 & -3 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -\frac{23}{10} & 1 & 0 & -3 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -\frac{23}{10} & 1 & -3 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 100
\end{array}\right]
$$

```
'''
(e) Use one of the numerical algorithms you developed for homework 3
    (Gauss elimination or LU decomposition) to solve the system derived
    in question 1(d) above. Validate your numerical solution by comparison
    to the analytic solution that you obtained in 1(c) through depicting
    the two solutions on plots over the interval 0 ≤ x ≤ 10.
'''
```



```
108
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL                1: Python Debug Console
   0.00000000e+00  0.00000000e+00  0.00000000e+00 -7.63869464e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00 -1.71847372e+00
   1.00000000e+00  0.00000000e+00  0.00000000e+00 -7.44210171e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  -1.71808828e+00  1.00000000e+00  0.00000000e+00 -7.33064623e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00 -1.71795773e+00  1.00000000e+00 -7.26674597e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  1.00000000e+00  1.00000000e+02]]
Coefficients: [[ 20.         -23.         -13.          -9.96969697  -8.65217391
   -7.99241275  -7.63869464  -7.44210171  -7.33064623  -7.26674597
  100.        ]]
Solution Values: [20.0, 16.27752228627173, 14.43830125842498, 13.930570608105715, 14.602011140218162, 16.654055014396057, 20.702315392892768, 27.961270389257304, 40.608606502399034, 62.43852456626046, 100.0]
bash-3.2$
3.7.1 64-bit  ⊗ 0 ⚠ 0   ▷ Python: Current File (project1)           Ln 97, Col 20   Spaces: 4   UTF-8   LF   Python
```

Solution Values:
[20.0,
16.27752228627173,
14.43830125842498,
13.930570608105715,
14.602011140218162,
16.654055014396057,
20.702315392892768,
27.961270389257304,
40.608606502399034,
62.43852456626046,
100.0]

20.
16.277522286271726464160396054564003344249921245197699024581450326577751708774336087525272526181418O3
14.438301258424970867568910925497207691774818863954707756537335751128828930180973001308126810217261 47
13.930570608105706531248099074079574346832162141898128815454421901018554830641901815483419137318283 36
14.602011140218154154301716944885813305939154062410988519007834621213847180295401174303737205614790 25
16.654055014396048023645849899157796256827892201647144778263597727773293684037520885415176435595734 22
20.702315392892756300083737823177118084764998001377444470998440152664728292990896862151168596255398 45
27.961270389257291466546747094149575338131603201520977505032814623355581389841541897532511335791682 22
40.608606502399014072973780493366905192937689362120803790577033481053108903644649502173607476065470 65
62.438524566260440901292948040594306605625082331356871213294362383066569088541151957466785859158900 28
100.

PROBLEM 1.e

'''

(e) Use one of the numerical algorithms you developed for homework 3

(Gauss elimination or LU decomposition) to solve the system derived

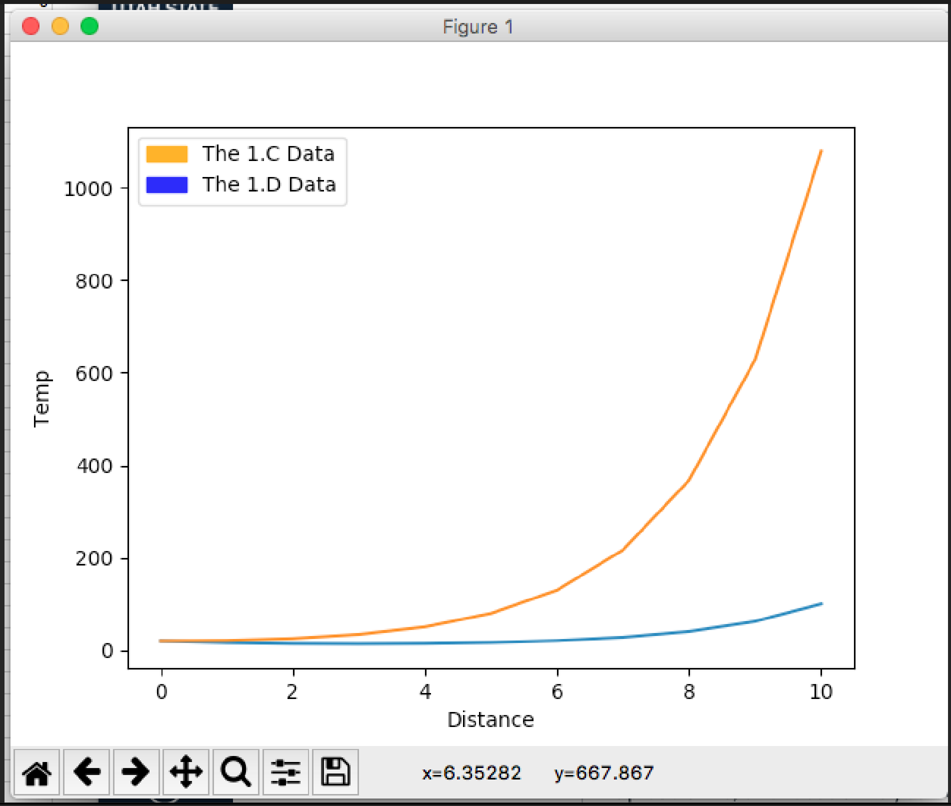
in question 1(d) above. Validate your numerical solution by comparison

to the analytic solution that you obtained in 1(c) through depicting

```
    the two solutions on plots over the interval 0 ≤ x ≤ 10.
'''
```

```
import NM_HW3 as hw
consts =[20,-3,-3,-3,-3,-3,-3,-3,-3,-3,100]
print( np.matrix(consts))
solutions = hw.guss2(bigArray,consts)
import math
def Temp(x):
    A=10
    B=4.467121520
    C=5.532878481
    λ=math.sqrt(0.3)
    return A+B*math.exp(λ*x) +C*math.exp(-λ*x)
solutions2 = [None]*11
for i in range(0,11):
    solutions2[i] = Temp(i)

import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
# xVals = range(0,11)
print('Solution1.D:',solutions)
print('Solution1.C:',solutions2)
orange_patch = mpatches.Patch(color='orange', label='The 1.C Data')
blue_patch = mpatches.Patch(color='blue', label='The 1.D Data')
plt.legend(handles=[orange_patch,blue_patch])
plt.plot(solutions)
plt.plot(solutions2)
plt.ylabel('Temp')
plt.xlabel('Distance')
```

```
plt.show()
    0.00000000e+00  0.00000000e+00  1.00000000e+00  1.00000000e+02]]
 Coefficients: [[ 20.         -23.         -13.          -9.96969697  -8.65217391
    -7.99241275  -7.63869464  -7.44210171  -7.33064623  -7.26674597
   100.        ]]
 Solution Values: [20.0, 16.27752228627173, 14.43830125842498, 13.930570608105715, 14.602011140218162, 16.654055014396057, 20.702315392892768, 27.961
 270389257304, 40.608606502399034, 62.43852456626046, 100.0]
 Solution1.D: [20.0, 16.27752228627173, 14.43830125842498, 13.930570608105715, 14.602011140218162, 16.654055014396057, 20.702315392892768, 27.9612703
 89257304, 40.608606502399034, 62.43852456626046, 100.0]
 Solution1.C: [20.000000001, 20.924509987432618, 25.209130537226223, 34.171705543234296, 50.56890262599515, 79.44409628840903, 129.67858586159338, 21
 6.7232647269521, 367.35093507664936, 627.8909633407322, 1078.479053743519]
```



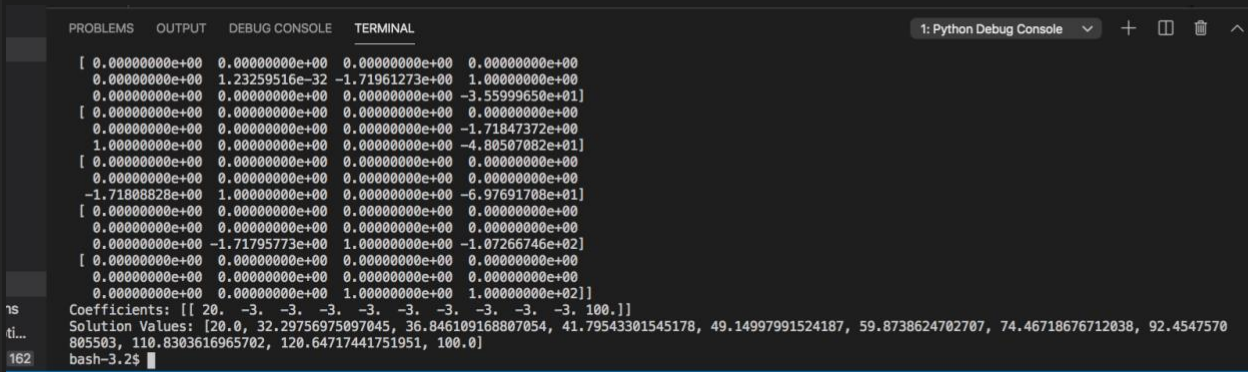
It was found that the analytical solution was more accurate in the larger temperature

PROBLEM 1.f

```
'''
(f) Write a function that takes as input the number of interior nodes
    n desired for your numerical solution (i.e. n = 9 in 1(d) above),
    and outputs the numerical solution to (1) in the form of the interior
    node values T1 = T(Δx), T2 = T(2Δx),..., Tn = T(nΔx).
'''
```

```
def n_TempSolution(n,lengthOfRod):
    # deltaX = lengthOfRod/(n+1)
    # m= n+2
    # solutionMat = [ [[0] * m for i in range(n+2)]]
    deltaX = lengthOfRod/(n+1)
    m= n+2
    solutionMat =  [[0] * m for i in range(n+2)]
    consts = [None]*(n+2)
    # Rows
    for i in range(1,n+1):
        # Colloms
        for j in range(0,m):
            if i==j:
                solutionMat[i][j]= (-2/(deltaX**2) -.3)
            elif i+1 == j or i-1==j:
                solutionMat[i][j]=1/(deltaX**2)
            else:
                solutionMat[i][j]= 0
        solutionMat[i][m-1] = (-3)
        consts[i] = (-3)



    solutionMat[0][0] = 1
    solutionMat[0][m-1] = 20
    consts[0] = 20
    solutionMat[n+1][m-2] =1
    solutionMat[n+1][m-1] = 100
    consts[n+1] = 100


    print( np.matrix(solutionMat))
```

```
    print( np.matrix(consts))
    solutions = hw.guss2(bigArray,consts)
    return solutions

lengthOfRod = 10
n_TempSolution(9,lengthOfRod)
```



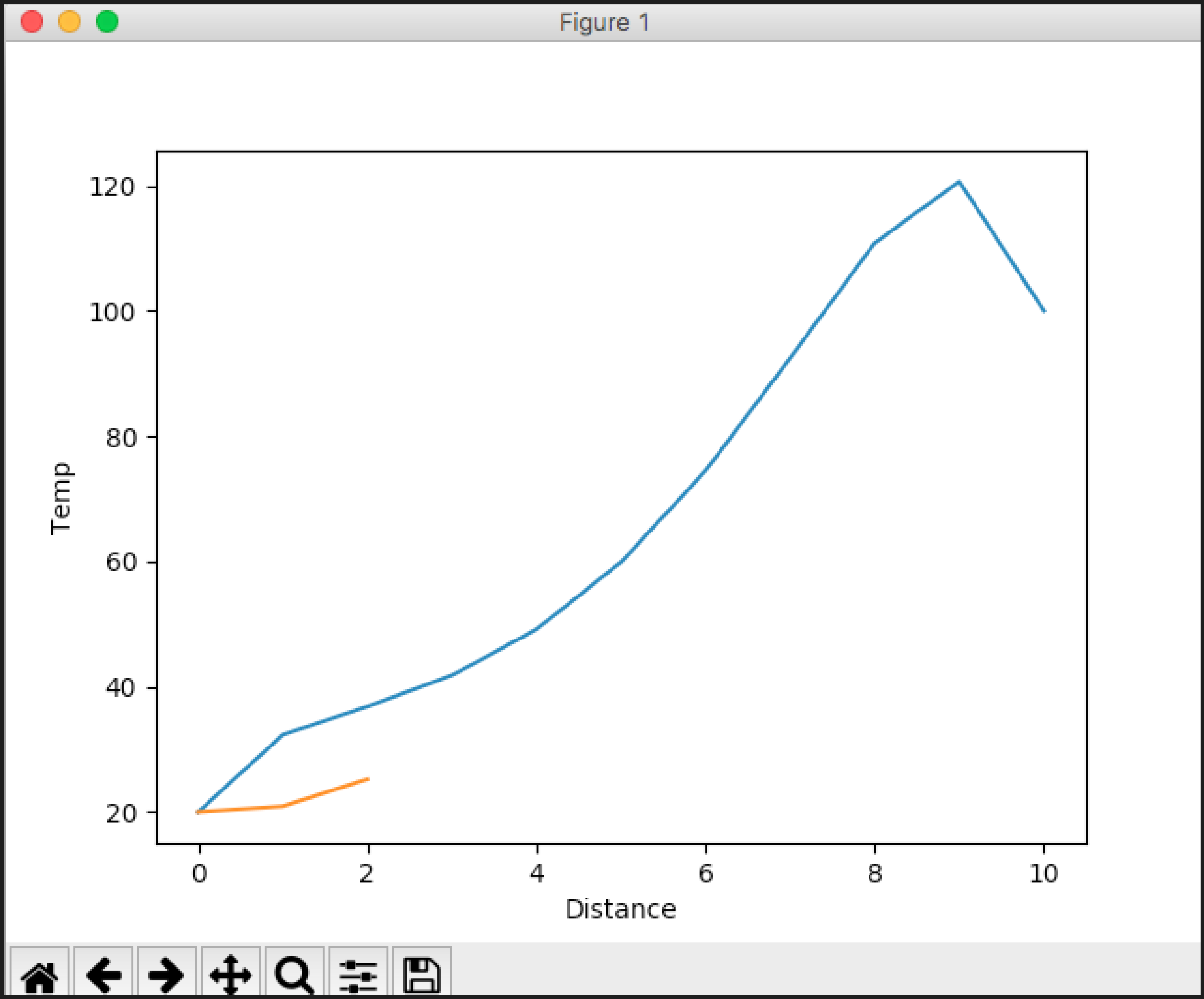

```
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  1.23259516e-32 -1.71961273e+00  1.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00 -3.55999650e+01]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00 -1.71847372e+00
   1.00000000e+00  0.00000000e+00  0.00000000e+00 -4.80507082e+01]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  -1.71808828e+00  1.00000000e+00  0.00000000e+00 -6.97691708e+01]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00 -1.71795773e+00  1.00000000e+00 -1.07266746e+02]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  1.00000000e+00  1.00000000e+02]]
Coefficients: [[ 20.  -3.  -3.  -3.  -3.  -3.  -3.  -3.  -3.  -3. 100.]]
Solution Values: [20.0, 32.29756975097045, 36.846109168807054, 41.79543301545178, 49.14997991524187, 59.8738624702707, 74.46718676712038, 92.4547570
805503, 110.8303616965702, 120.64717441751951, 100.0]
bash-3.2$
```

PROBLEM 1.g

```
'''
(g) Produce and submit 4 plots that compare your analytic solution
    to (1) derived in question 2(b) to the numerical solution generated
    in question 2(f) for n = 1, n = 4, n = 9, and n = 19, respectively.
'''
lengthOfRod = 10
val = 1
solutions2 = [None]*(val+2)
for i in range(0,val+2):
    solutions2[i] = Temp(i)
```

```
solutionG_1=n_TempSolution(1,lengthOfRod)
plt.plot(solutionG_1,)
plt.plot(solutions2)
plt.ylabel('Temp')
plt.xlabel('Distance')
plt.show()

val = 4
solutions2 = [None]*(val+2)
for i in range(0,val+2):
    solutions2[i] = Temp(i)
solutionG_4=n_TempSolution(4,lengthOfRod)
plt.plot(solutionG_4,)
plt.plot(solutions2)
plt.ylabel('Temp')
plt.xlabel('Distance')
plt.show()

val = 9
solutions2 = [None]*(val+2)
for i in range(0,val+2):
    solutions2[i] = Temp(i)
solutionG_9=n_TempSolution(9,lengthOfRod)
plt.plot(solutionG_9,)
plt.plot(solutions2)
plt.ylabel('Temp')
plt.xlabel('Distance')
plt.show()

val = 19
```

```
solutions2 = [None]*(val+2)
for i in range(0,val+2):
    solutions2[i] = Temp(i)
solutionG_19=n_TempSolution(19,lengthOfRod)
plt.plot(solutionG_19,)
plt.plot(solutions2)
plt.ylabel('Temp')
plt.xlabel('Distance')
plt.show()
```



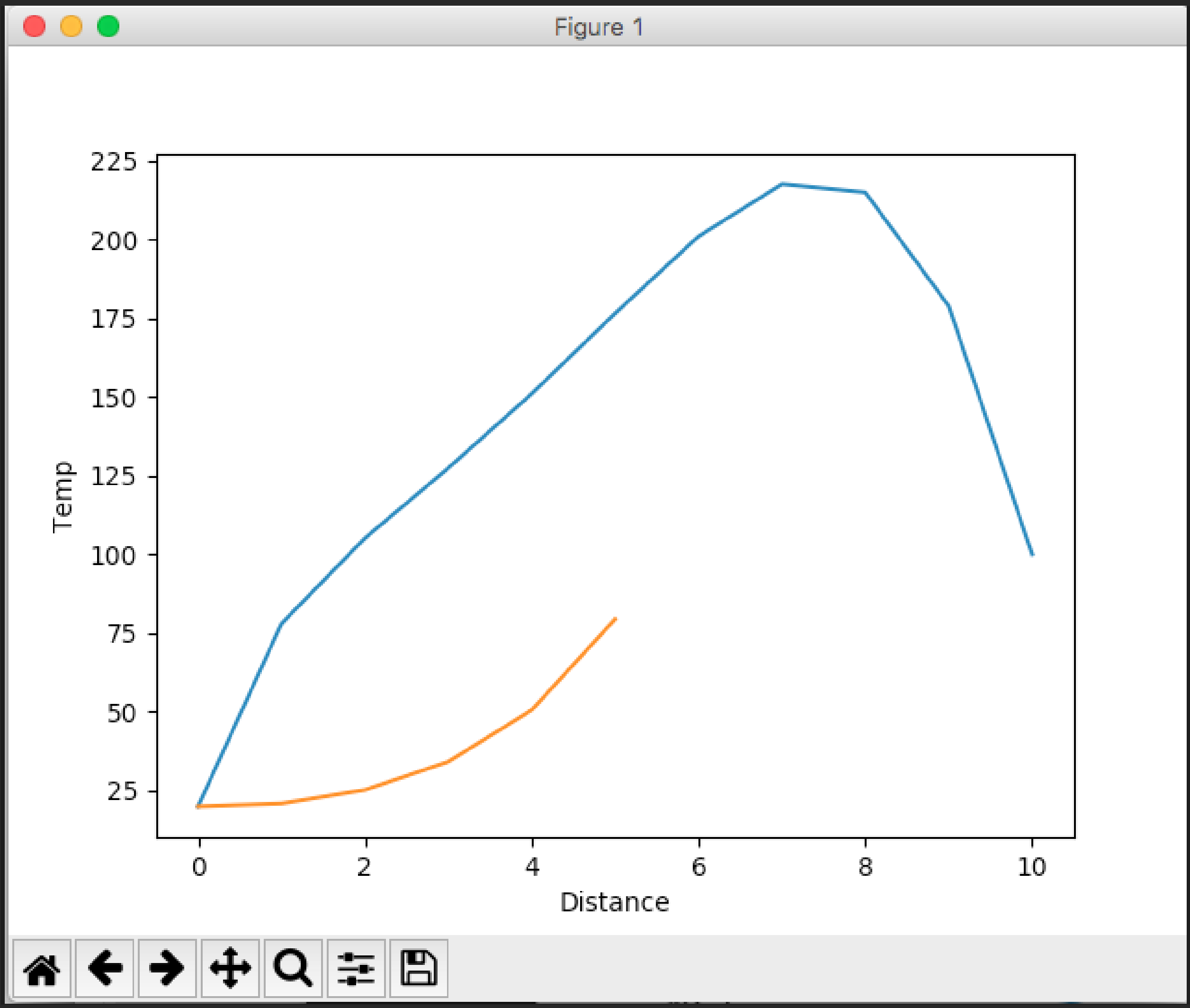


Figure 1
Temp
225
200
175
150
125
100
75
50
25
0
2
4
6
8
10
Distance

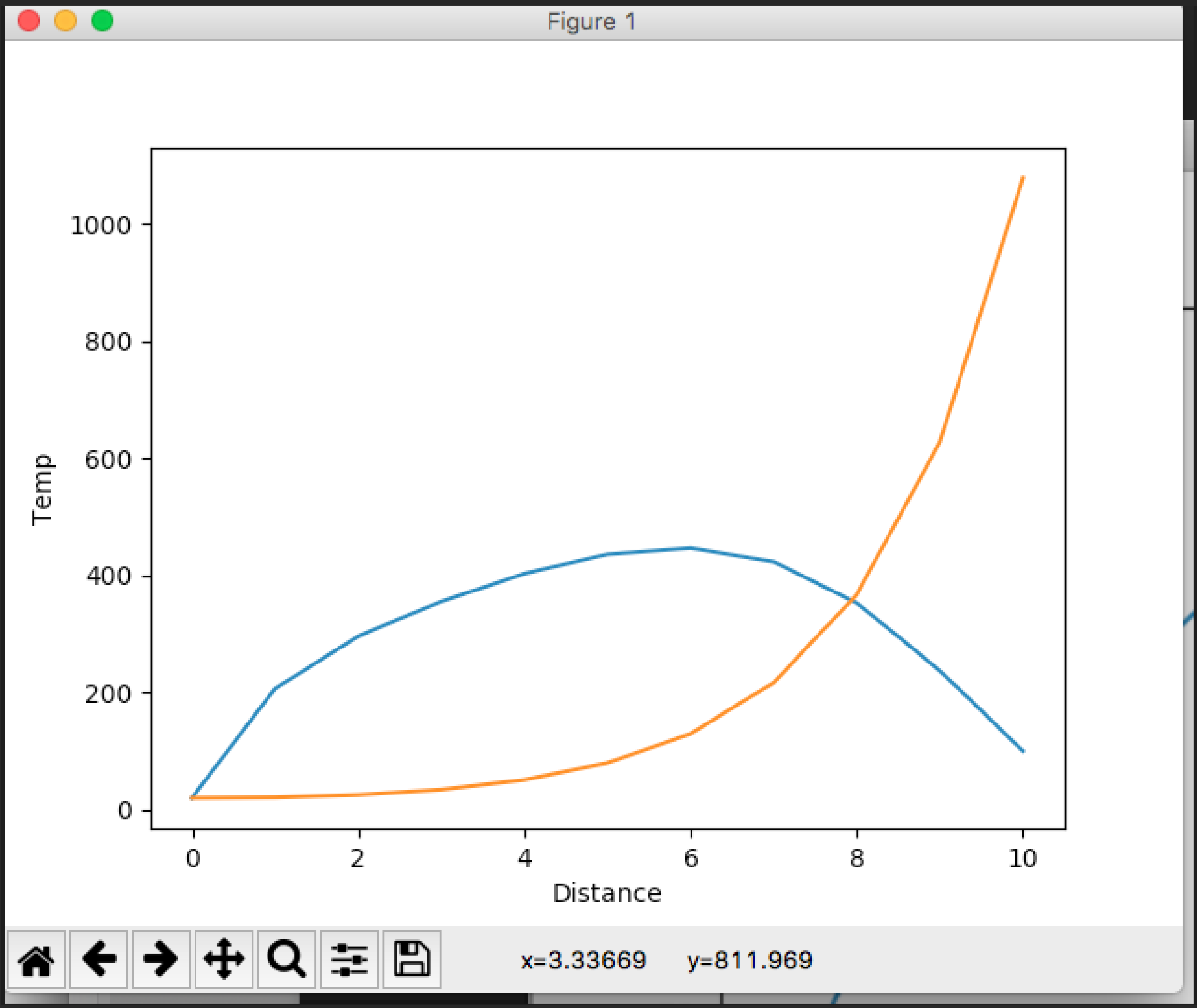


Figure 1
1000
800
600
400
200
0
Temp
0
2
4
6
8
10
Distance
x=3.33669 y=811.969

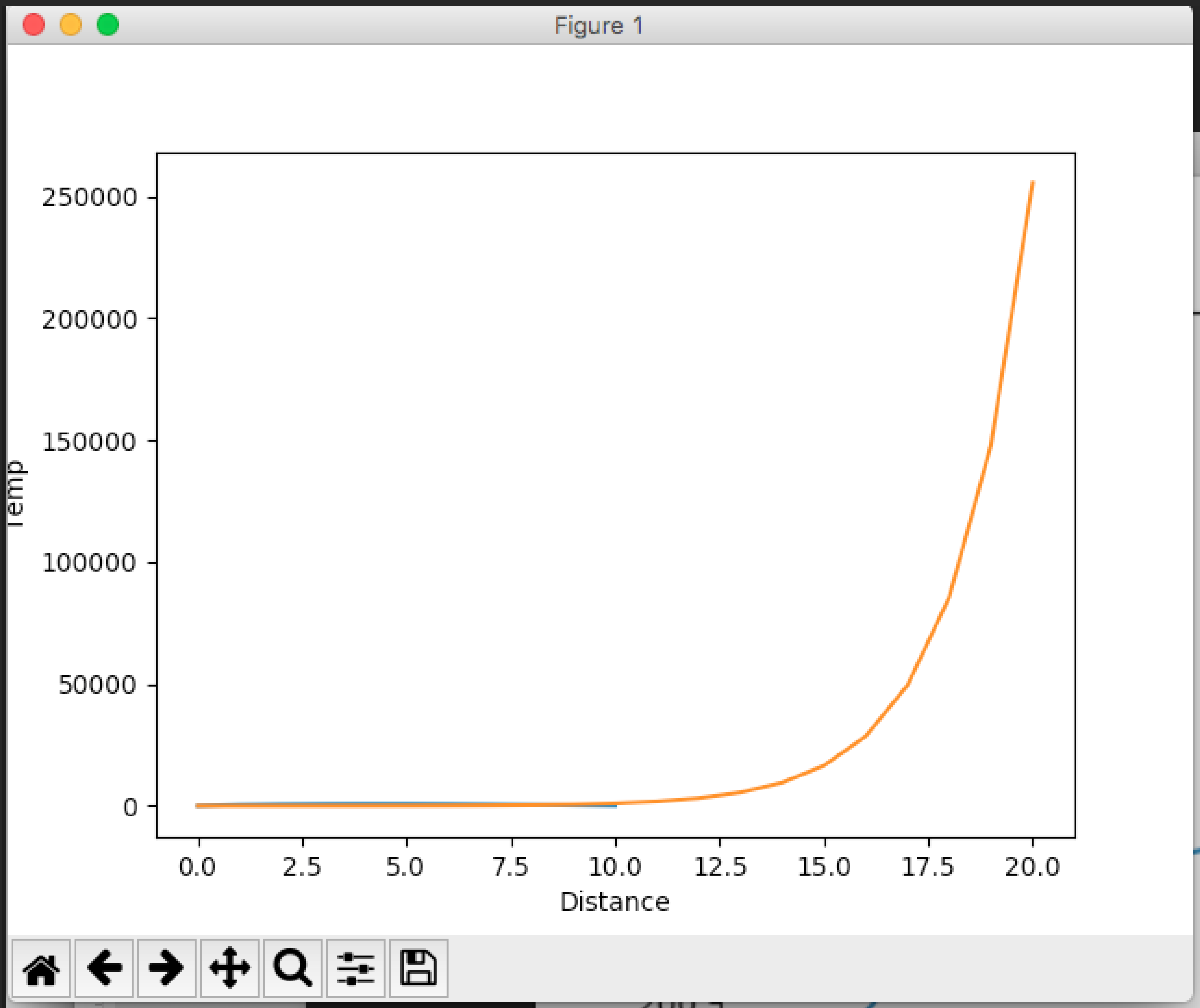


Figure 1
250000
200000
150000
100000
50000
0
Temp
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Distance

```
'''
2. Develop an algorithm that uses the golden section search to locate
   the minimum of a given function. Rather than using the iterative
   stopping criteria we have previously implemented, design the
   algorithm to begin by determining the number of iterations n required
   to achieve a desired absolute error |Ea| (not a percentage), where
   the value for |Ea| is input by the user. You may gain insight by
   comparing this approach to a discussion regarding the bisection
   method on page 132 of the textbook. Test your algorithm by applying
   it to find the minimum of f(x) = 2x+ (6/x) with initial guesses xl = 1
   and xu = 5 and desired absolute error |Ea| = 0.00001.
'''
```

$$n = \frac{\log(\Delta x^0 / E_{a,d})}{\log 2} = \log_2\left(\frac{\Delta x^0}{E_{a,d}}\right) \qquad (5.5)$$

PROBLEMS TERMINAL ··· 1: Python Debug Console

```
New Lower Test Point =  1.7315417613997808
f2 < f1
New Upper Bound = 1.732661895728071
New Lower Bound = 1.7315417613997808
New Lower Test Point =  1.731969614641222
New Upper Test Point =  1.7322340424866296
New Upper Bound = 1.7322340424866296
New Lower Bound = 1.7315417613997808
New Upper Test Point =  1.731969614641222
New Lower Test Point =  1.7318061892451884
f2 < f1
New Upper Bound = 1.7322340424866296
New Lower Bound = 1.7318061892451884
New Lower Test Point =  1.731969614641222
New Upper Test Point =  1.732070617090596
Final Lower Bound = 1.7318061892451884
Final Upper Bound = 1.7322340424866296
Final Local Minima = 1.732020115865909
bash-3.2$
```

n 3.7.1 64-bit ⓧ 0 ⚠ 0 ▷ Python: Current File (project1) Spaces: 4

The Value for the local minima:

1.732020115865909

Golden Search

```
import math
#  goldenSearch
def goldenSearch(function,xl, xu,tol):


    goldenRatio = 2/(math.sqrt(5) + 1)


    # Use the goldon ratio to set initial points
    x1 = xu - goldenRatio * (xu - xl)
    x2 = xl + goldenRatio * (xu - xl)


    f1 = function(x1)
```

```
f2 = function(x2)

iter = 0
deltaX = abs(xu - xl)
Ead = tol
numIteration = int(math.log2(deltaX/Ead))

# while (abs(xu - xl) > tol):
while( iter <= numIteration):
    iter = iter + 1

    if (f2 > f1):

        xu = x2
        print('New Upper Bound =', xu)
        print('New Lower Bound =', xl)
        # Set the new upper test point
        # Use result of the goldon ratio
        x2 = x1
        print('New Upper Test Point = ', x2)
        f2 = f1

        # Set the new lower test point
        x1 = xu - goldenRatio*(xu - xl)
        print('New Lower Test Point = ', x1)
        f1 = function(x1)
    else :
        print('f2 < f1')

        xl = x1
        print('New Upper Bound =', xu)
```

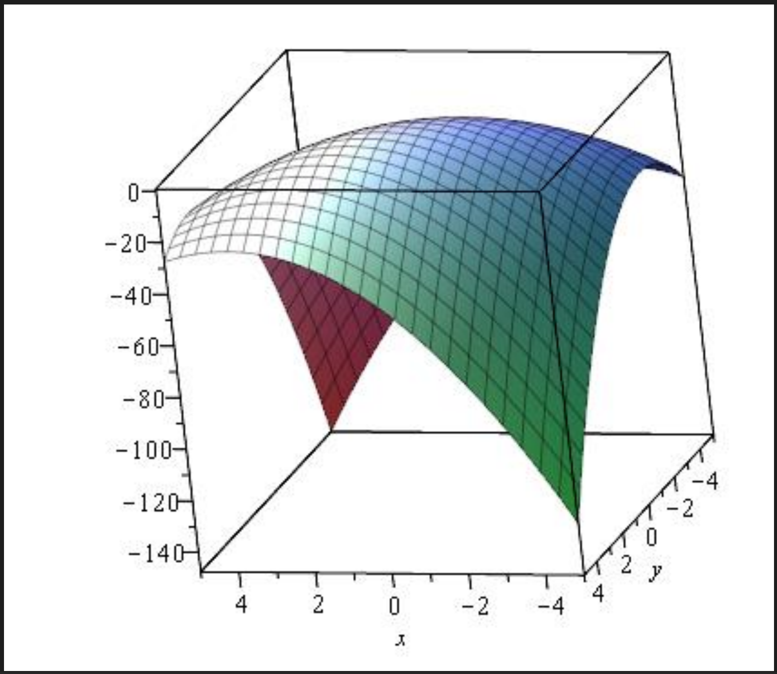
```python
            print('New Lower Bound =', xl)

            # Set the new lower test point
            x1 = x2
            print('New Lower Test Point = ', x1)

            f1 = f2

            # Set the new upper test point
            x2 = xl + goldenRatio*(xu - xl)
            print('New Upper Test Point = ', x2)
            f2 = function(x2)

    print('Final Lower Bound =', xl)
    print('Final Upper Bound =', xu)
    finalPoint = (xl + xu)/2
    print('Final Local Minima =', finalPoint )
    return finalPoint
```

PROBLEM 3

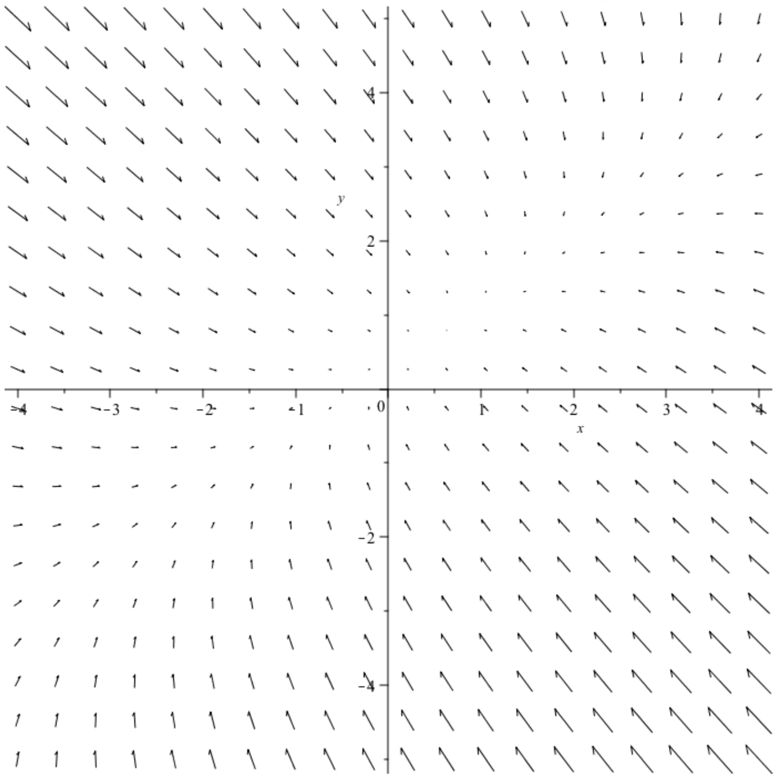
```python
'''
3. Given f(x,y)=2xy+2y−1.5x^2−2y^2,
(a) Start with an initial guess of (x0,y0) = (1,1) and determine
    (by hand is fine) two iterations of the steepest ascent method to
    maximize f(x,y).
'''
```

$$fun := 2 \cdot x \cdot y + 2 \cdot y - 1.5 \cdot x\verb|^|2 - 2 \cdot y\verb|^|2$$

$$fun := -2.0\,h + 2 - 1.5\,(-1.0\,h + 1)^2 \qquad \textbf{(1)}$$



$plots[:-fieldplot]([2*y-3.0*x, 2*x+2-4*y], x=-4..4, y=-5..5)$



This is the Gradient Graph (woo!)

$grad := Gradient(fun)$

$$grad := (2y - 3.0x)\bar{e}_x + (2x + 2 - 4y)\bar{e}_y$$

$grado := \langle 2\,yo - 3.0\,xo,\, 2\,xo + 2 - 4\,yo \rangle$

$$grado := \begin{bmatrix} 2\,yo - 3.0\,xo \\ 2\,xo + 2 - 4\,yo \end{bmatrix} \qquad \textbf{(5)}$$

$point1 := \langle 1, 1 \rangle$

$$point1 := \begin{bmatrix} 1 \\ 1 \end{bmatrix} \qquad \textbf{(6)}$$

$xo := 1$

$$xo := 1 \qquad \textbf{(7)}$$

$yo := 1$

$$yo := 1 \qquad \textbf{(8)}$$

$elivation := 2\,xo \cdot yo + 2 \cdot yo - 1.5\,xo^2 - 2 \cdot yo^2$

$$elivation := 0.5 \qquad \textbf{(9)}$$

 # Iteration 1

$point2 := point1 + h \cdot grado$

$$point2 := \begin{bmatrix} -1.0\,h + 1 \\ 1 \end{bmatrix} \qquad \textbf{(10)}$$

$x := point2[1]$

$$x := -1.0\,h + 1 \qquad \textbf{(11)}$$

$y := point2[2]$

$$y := 1 \qquad \textbf{(12)}$$

fun

$$-2.0\,h + 2 - 1.5\,(-1.0\,h + 1)^2 \qquad \textbf{(13)}$$

$isolate($ **(13)**, $h)$

$$h = -0.3333333333 \qquad \textbf{(14)}$$

$h := 0.3333333333$ # *take the oposit direction*

$$h := 0.3333333333 \tag{15}$$

point2

$$\begin{bmatrix} 0.6666666667 \\ 1 \end{bmatrix} \tag{16}$$

$xo := point2[1]$

$$xo := 0.6666666667 \tag{17}$$

$yo := point2[2]$

$$yo := 1 \tag{18}$$

$elivation := 2\, xo \cdot yo + 2 \cdot yo - 1.5\, xo^2 - 2 \cdot yo^2$

$$elivation := 0.666666666 \tag{19}$$

####2nd Iteration

$h :='h'$

$$h := h \tag{20}$$

$point3 := point2 + h \cdot grado$

$$point3 := \begin{bmatrix} 1. - 1.0\, h \\ -0.666666667\, h + 1 \end{bmatrix} \tag{21}$$

$x := point3[1]$

$$x := 1. - 1.0\, h \tag{22}$$

$y := point3[2]$

$$y := -0.666666667\, h + 1 \tag{23}$$

fun

$$(1. - 1.0\, h)\, (-0.666666667\, h + 1) - 1.333333334\, h + 2 - 1.5\, (1. - 1.0\, h)^2 - 2\, (-0.666666667\, h + 1)^2 \tag{24}$$

$isolate($ **(24)**, $h\ )$

$$h = -0.3618162035 \tag{25}$$

$h := 0.3618162035$ # *take the oposit direction*

$$h := 0.3618162035 \tag{26}$$

point3

$$\begin{bmatrix} 0.6381837965 \\ 0.7587891975 \end{bmatrix} \tag{27}$$

$xo := point3[1]$

$$xo := 0.6381837965 \tag{28}$$

$yo := point3[2]$

$$yo := 0.7587891975 \tag{29}$$

$elivation := 2\, xo \cdot yo + 2 \cdot yo - 1.5\, xo^2 - 2 \cdot yo^2$

$$elivation := 0.723632408 \tag{30}$$

'''

(b) What point is the steepest ascent method converging

towards? Justify your answer without computing any more

iterations.

''' NO COMPUTER CODE REQUIRED

It looks like the function is converging towards the point (0.5, 0.75) which has an elevation of z=0.75, this is clear in the **small increase in the calculated h**. Also, the function has a **smaller difference in the elevation** thus pointing to the converging elevation of 0.75. If you need to more evidence the **gradient graph** also helps visualize the local maxima.