#  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**,



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

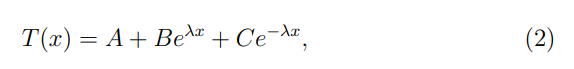
that the rod is 10 meters long (i.e. 0 ≤ x ≤ 10) and has

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+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.

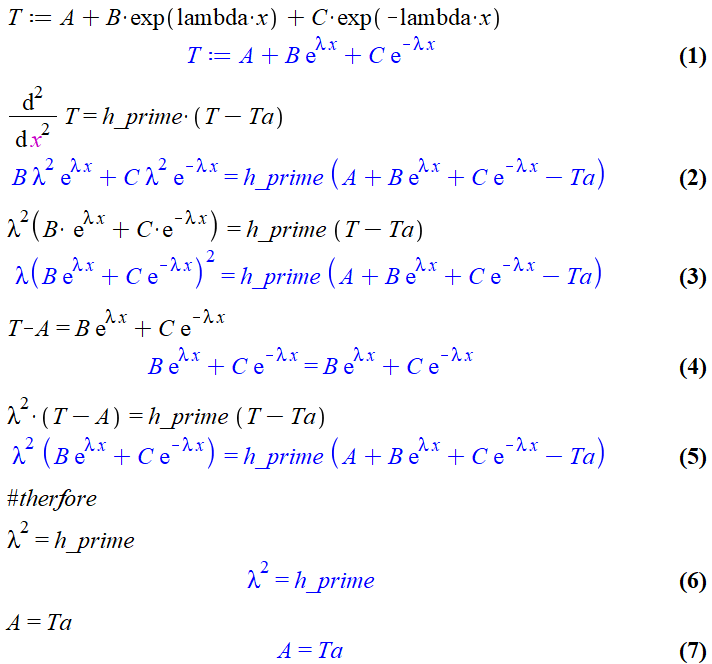
''' NO COMPUTER CODE REQUIRED

1m 5 m 10 m

| | |

^ ^

20oC 100oC



Found Lambda and A

PROBLEM 1.b

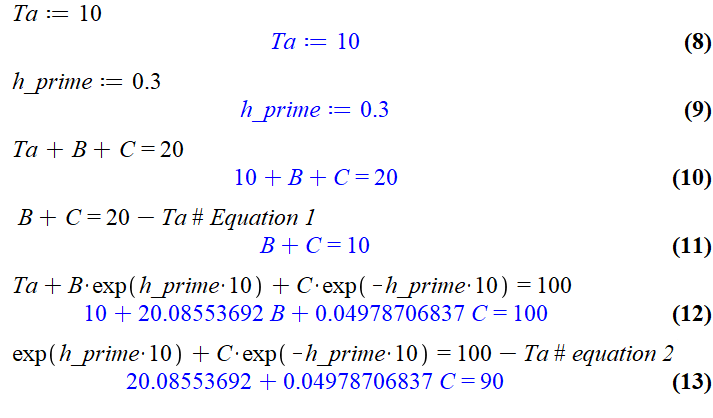
'''

(b) Next, impose the boundary conditions T (0) = 20 oC and

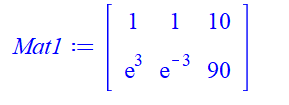
    T (10) = 100 oC to derive a system of 2 linear algebraic equations

    for B and C**. Provide the system of two equations** you have derived.

''' NO COMPUTER CODE REQUIRED



The system of two equations:



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

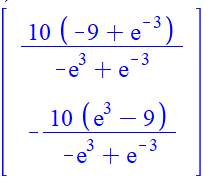
    analytic solution to (1) of the form (2). By analytic solution we mean

    an explicit solution to equation (1) which is valid for each x in

    the interval [0, 10].

'''

|  |
| --- |
| Gauss elimination |
| #  Authored by Christopher Allred  #  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)          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 |



Solution Values: [4.467121518528577, 5.532878481471423]  
A = 4.467121518528577

B = 5.532878481471423

PROBLEM 1.d

'''

(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),

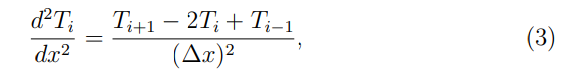
 T8 =T(8),

 T9 =T(9),

    by

    using the following **finite difference approximation**for the second

    derivative at the ith interior point,



d2Ti/ dx2 = (Ti+1 −2Ti +Ti−1)/ (∆x)\*\*2,  (3)

    where

**1≤i≤9,**

**T0 =T(0) =20oC,**

**T10=T(10)=100oC,**

    and **∆x is the equal spacing** between consecutive interior points

    (i.e. with 9 equally spaced interior points inside [0,10] it holds

    that ∆x = 1). Use (3) to rewrite (1)

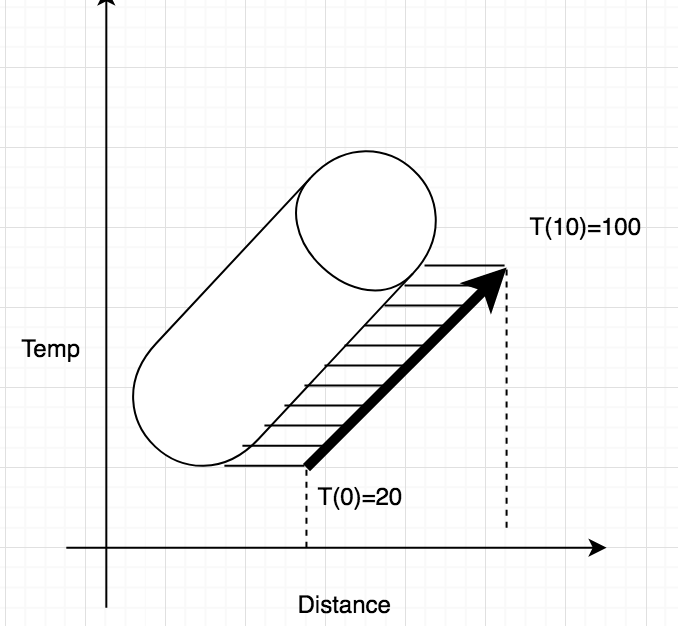


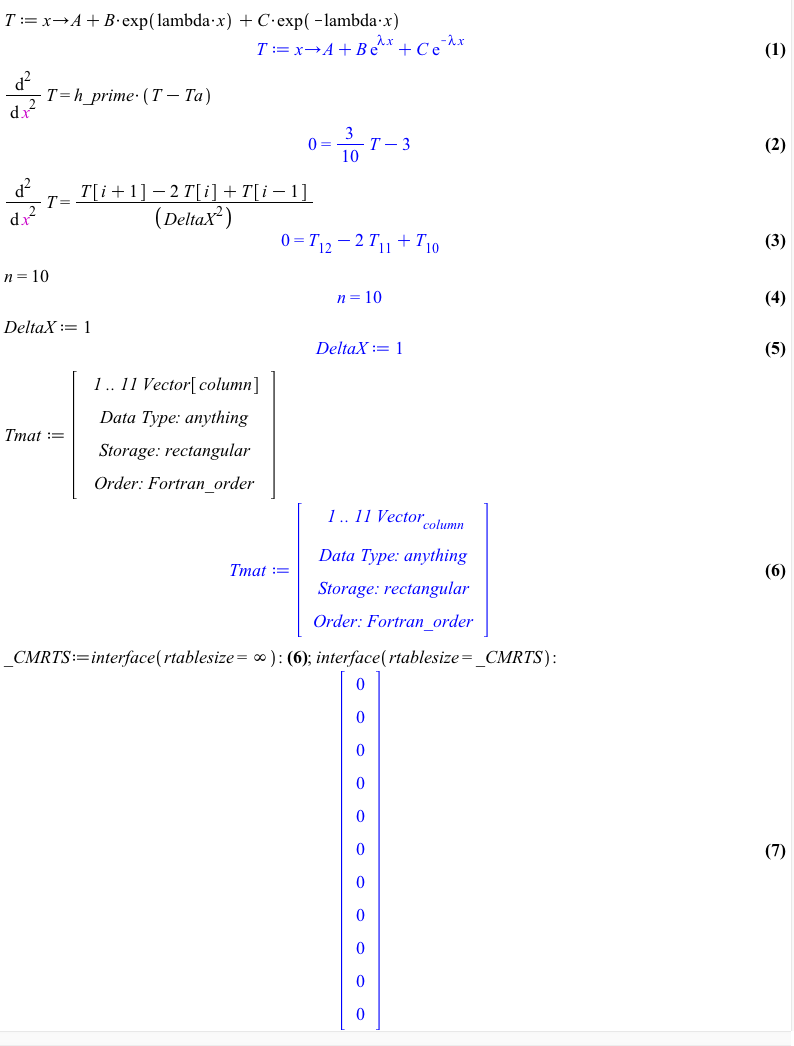
as a system of 9 linear algebraic

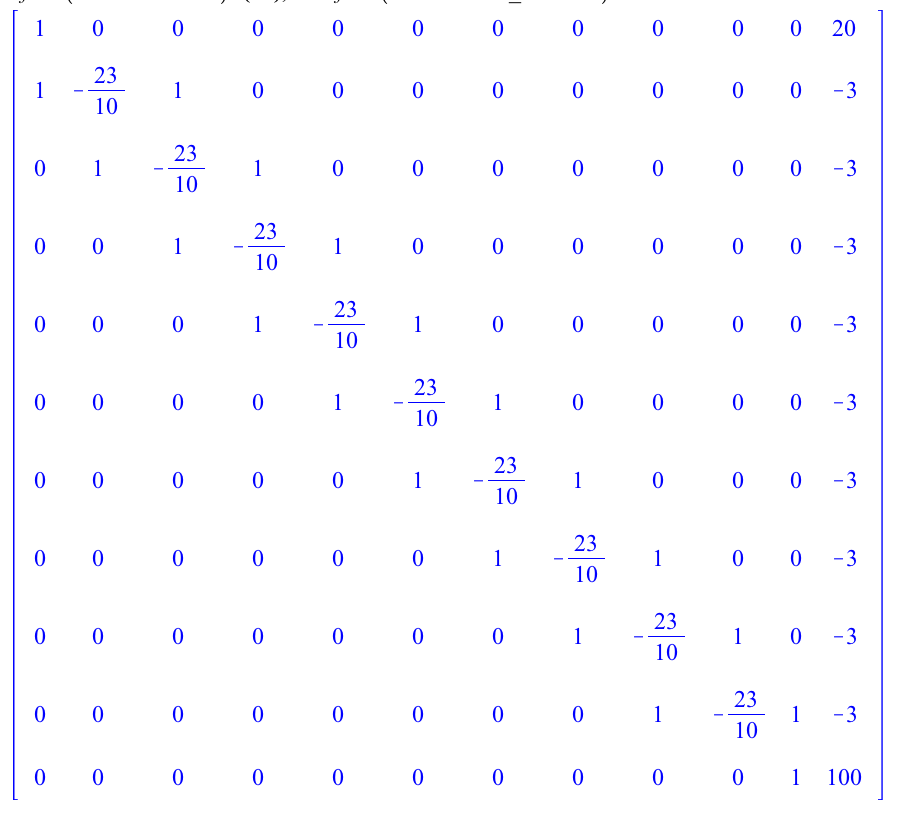
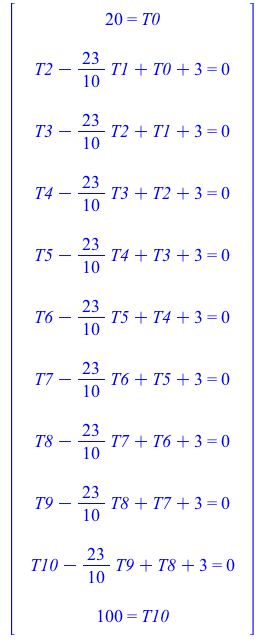
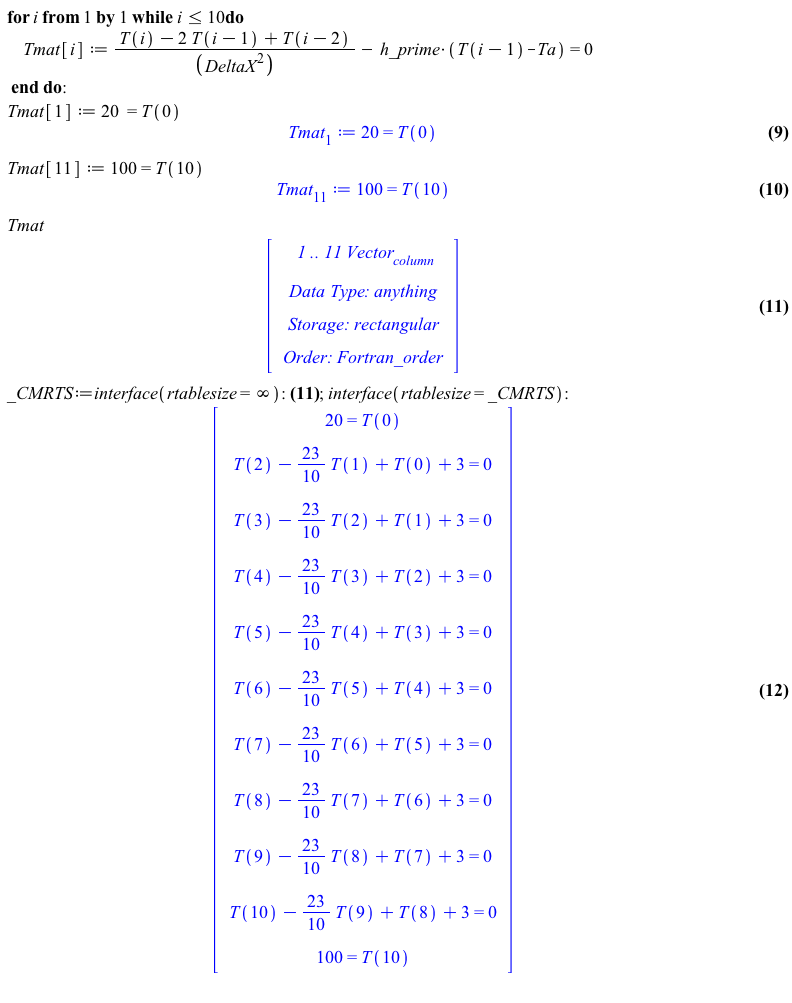
    equations for the unknowns T1, T2, T3, T4, T5, T6, T7, T8, and T9.

    Provide the system of 9 equations you have derived.

''' NO COMPUTER CODE REQUIRED







'''

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

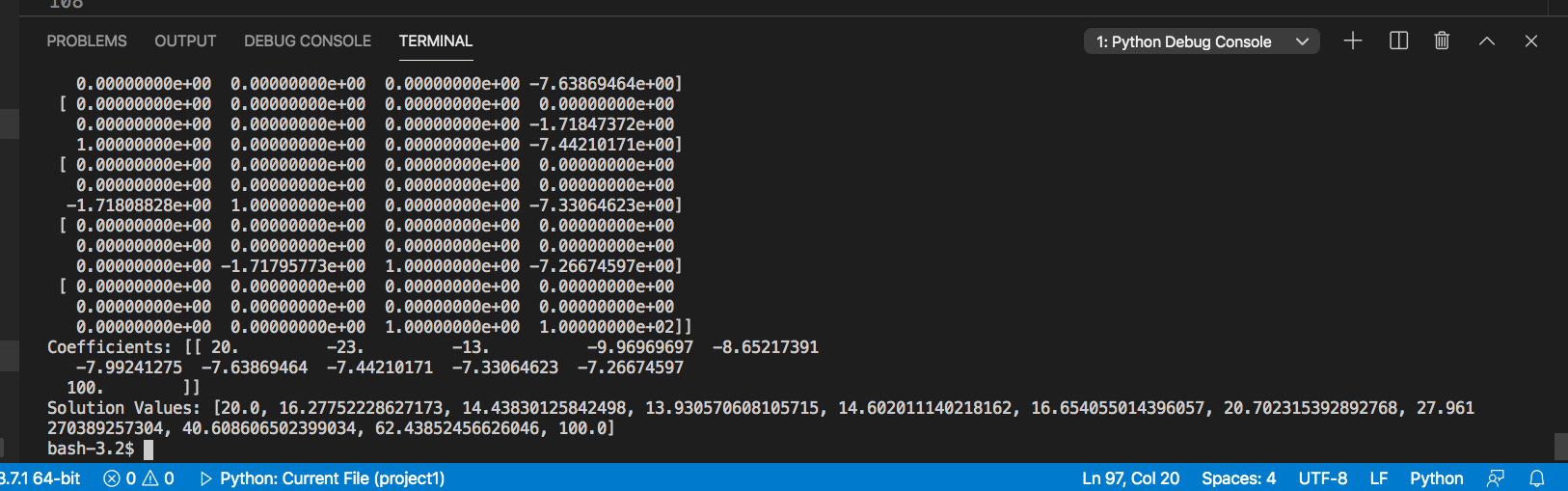
(Gauss elimination or LU decomposition) to solve the system derived

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.

'''



Solution Values:

[20.0,

16.27752228627173,

14.43830125842498,

13.930570608105715,

14.602011140218162,

16.654055014396057,

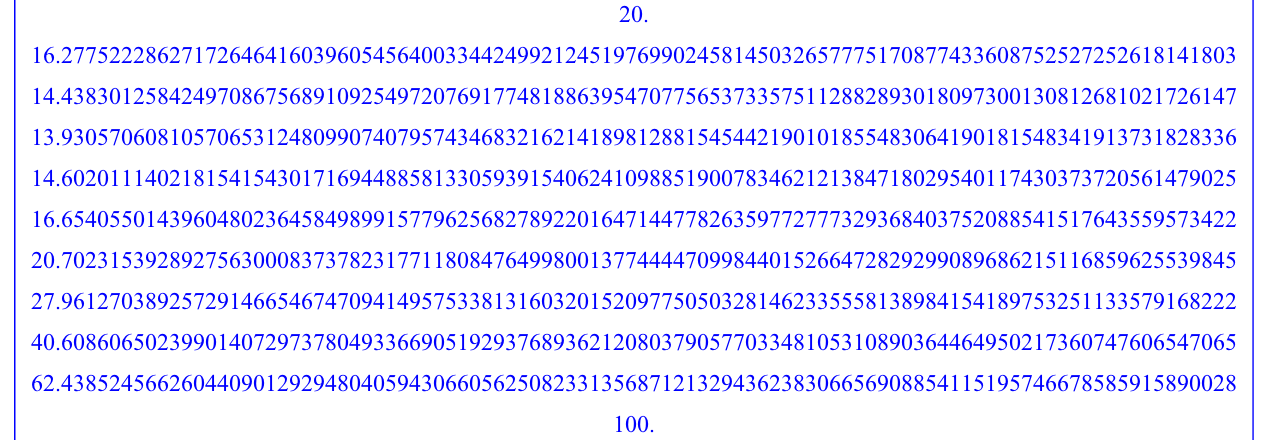
20.702315392892768,

27.961270389257304,

40.608606502399034,

62.43852456626046,

100.0]



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() |

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(∆x), T2 = T(2∆x),..., Tn = T(n∆x).

'''

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))

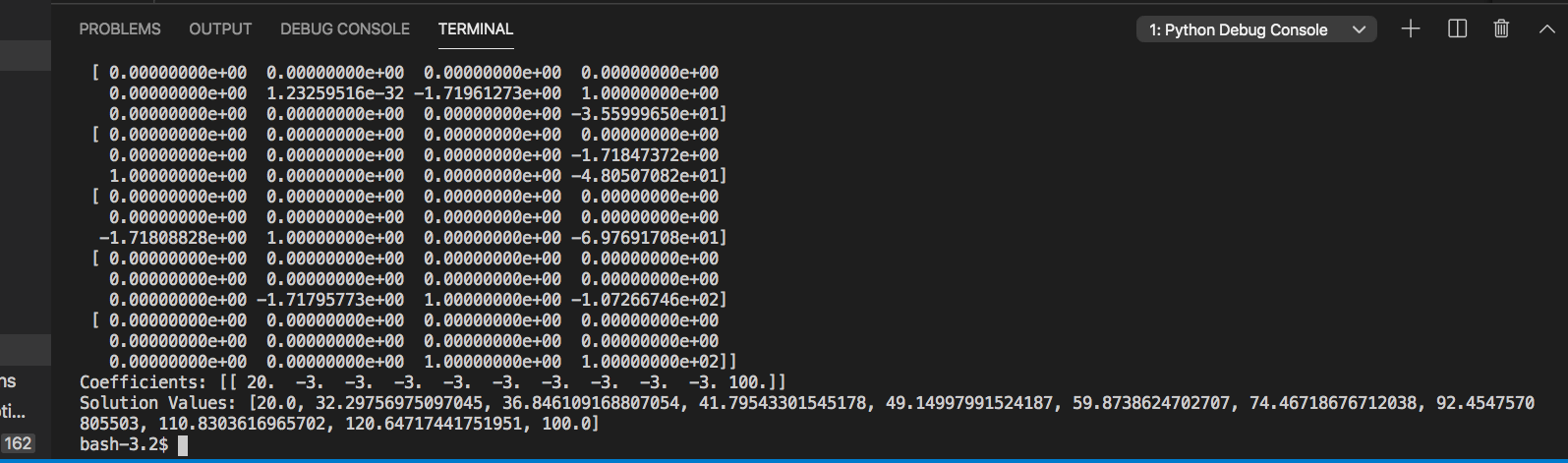
print( np.matrix(consts))

solutions = hw.guss2(bigArray,consts)

return solutions

lengthOfRod = 10

n\_TempSolution(9,lengthOfRod)



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)

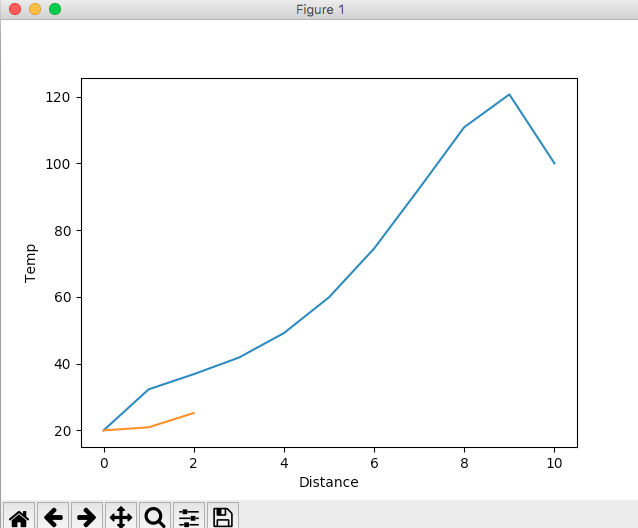
plt.plot(solutionG\_19,)

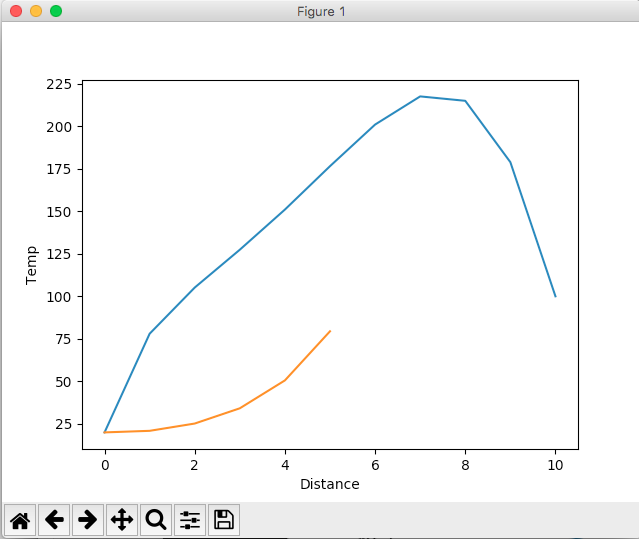
plt.plot(solutions2)

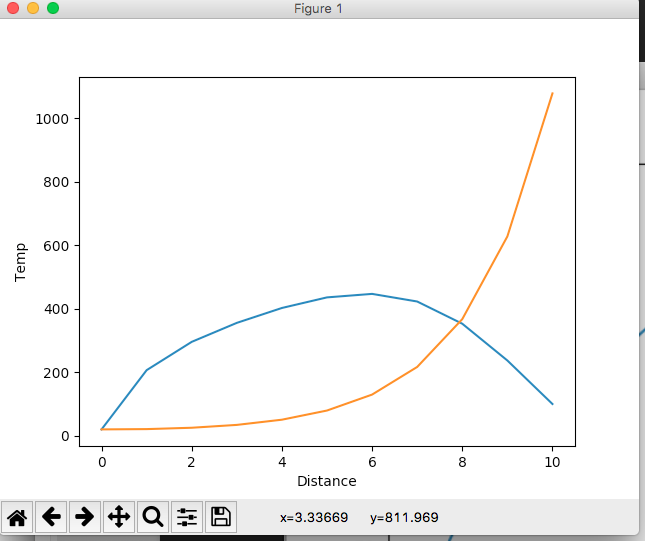
plt.ylabel('Temp')

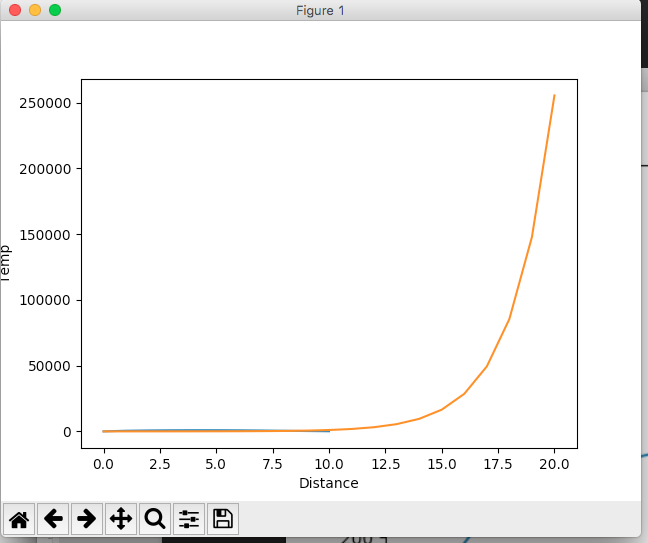
plt.xlabel('Distance')

plt.show()









PROBLEM 2

'''

**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

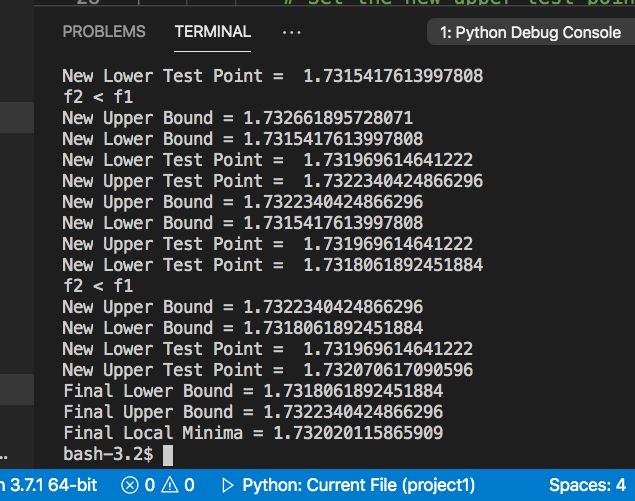
method on page 132 of the textbook. Test your algorithm by applying

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.

'''





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)  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

'''

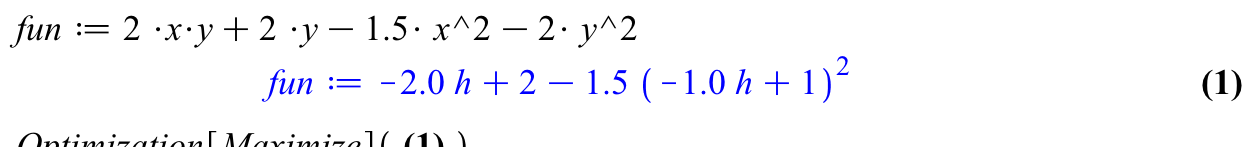
**3. Given f(x,y)=2xy+2y−1.5x^2−2y^2,**

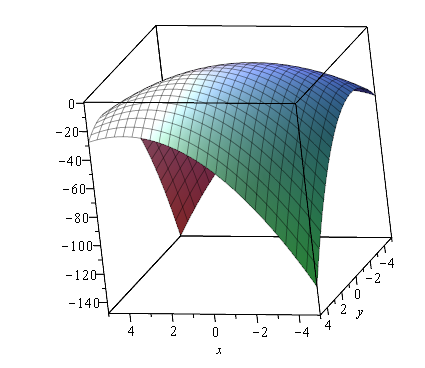
(a) Start with an initial guess of (x0,y0) = (1,1) and determine

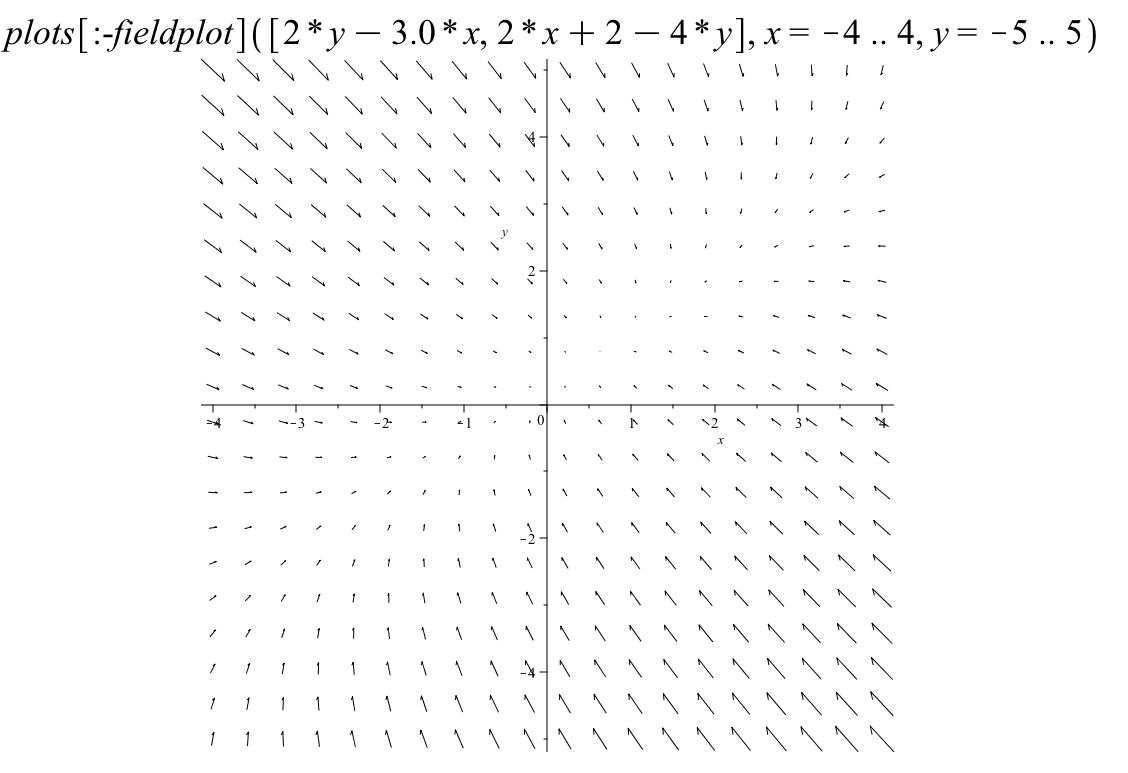
(by hand is fine) two iterations of the **steepest** ascent method to

maximize f(x,y).

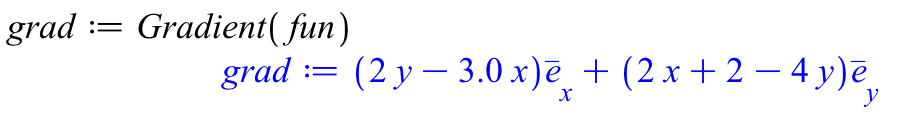
'''

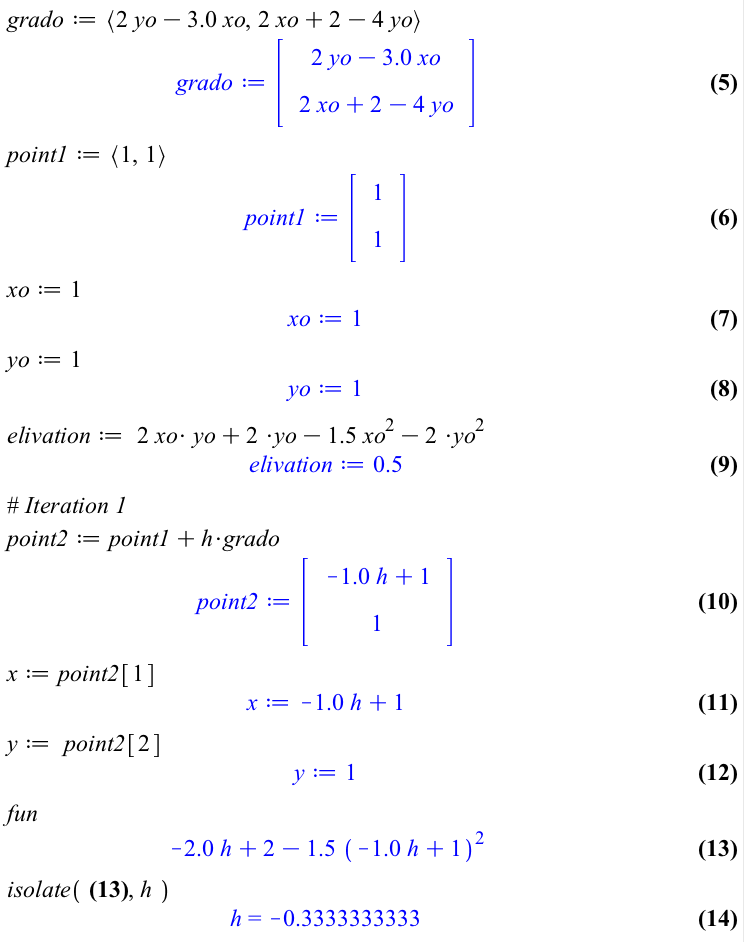


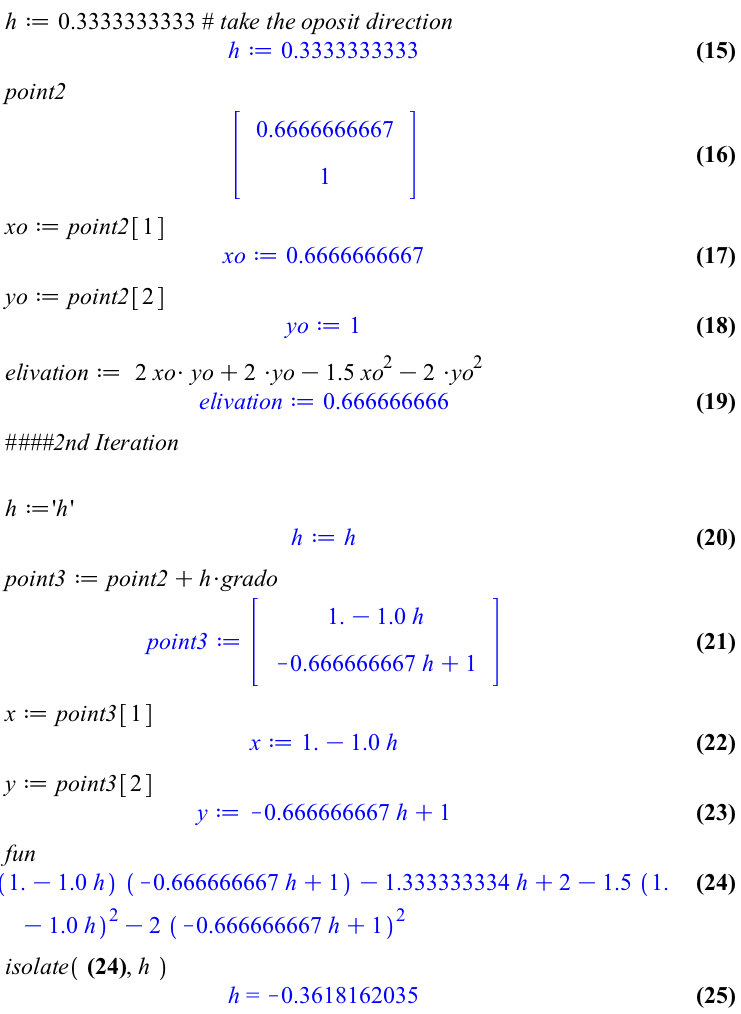
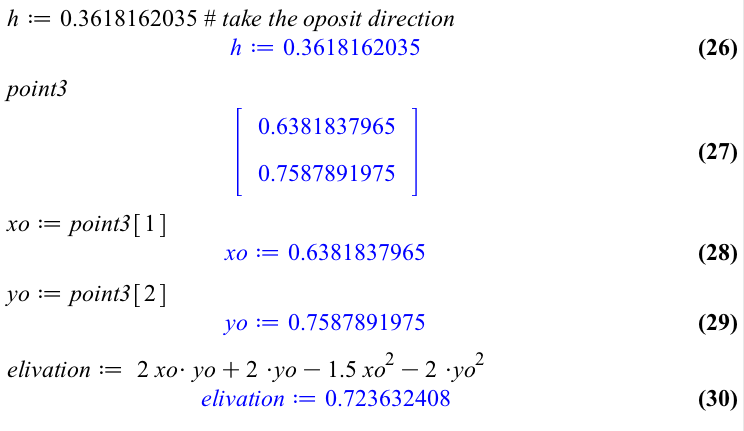




This is the Gradient Graph (woo!)





'''

(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.