Assignment

Friday, January 31, 2020 2:58 AM

Shaun Fedrick

Homework 1

Due (in class) Monday, February 17 / Print and submit your codes if any

1. *Linear algebra recap: Do not need to submit your work.*

   Read the Appendix (A Review of Linear Algebra) in the text. Google-search for the TDMA (Thomas) algorithm. Follow the derivation with your hands, and learn the technique by yourself. Students are expected to be familiar with these materials for the rest of the course.

2. Numerically evaluate the derivative of the following function

   $$f(x) = \sin 2x \cdot \cos 20x + e^{\sin(2x)}, \quad x \in [0, 2\pi], \tag{1}$$

   using

   i) first-order forward finite difference scheme,

   ii) second-order central finite difference scheme,

   iii) fourth-order central finite difference scheme, and

   iv) forth-order Padé scheme.

   Consider $N = 256$, 512, 1024 and 2048, where N is the number of intervals to discretize the x domain. Exploit the periodicity of the function $f(x)$ to deal with boundary points. Plot the numerical results and compare them to the exact derivative. Plot the L_∞ error vs the grid spacing Δ, and verify the order of accuracy of each scheme empirically (include asymptotes for 1st, 2nd, 3rd, and 4th order lines, see Figure 2.1 in the textbook).

   For the Padé4 scheme, report results obtained with two different boundary treatments: 1) third-order one-sided boundary schemes as in (2.17), and 2) periodic boundary condition where the Padé4 formula (2.16) can be applied as it is to the boundary points. Discuss the resulting matrix structures for each approach. Can the Thomas algorithm be applied to 2)?

3. Problem 2.1 from the text.

4. Problem 2.3 from the text. Plot the result along with the modified wavenumber of the second-order central difference scheme as in Figure 2.2 ($k'\Delta$ *vs.* $k\Delta$).

5. Problem 2.8 from the text.

6. Problem 2.10 from the text.

7. Problem 3.2 from the text, where $\frac{\delta}{\delta x}\Big|_i$ is the second-order central difference operator.

8. Problem 3.3 from the text.

9. Problem 3.5 from the text.

10. Problem 3.8 from the text. For part (c), do not use canned subroutines from MATLAB. Implement your own adaptive quadrature code using the Simpson's rule as the base method.

Wednesday, February 5, 2020 7:12 PM

2. Numerically evaluate the derivative of the following function

$$f(x) = \sin 2x \cdot \cos 20x + e^{\sin(2x)}, \quad x \in [0, 2\pi],$$

using

i) first-order forward finite difference scheme,

ii) second-order central finite difference scheme,

iii) fourth-order central finite difference scheme, and

iv) forth-order Padé scheme.

Consider $N = 256$, 512, 1024 and 2048, where N is the number of intervals to discretize the x domain. Exploit the periodicity of the function $f(x)$ to deal with boundary points. Plot the numerical results and compare them to the exact derivative. Plot the L_∞ error vs the grid spacing Δ, and verify the order of accuracy of each scheme empirically (include asymptotes for 1st, 2nd, 3rd, and 4th order lines, see Figure 2.1 in the textbook).

For the Padé4 scheme, report results obtained with two different boundary treatments: 1) third-order one-sided boundary schemes as in (2.17), and 2) periodic boundary condition where the Padé4 formula (2.16) can be applied as it is to the boundary points. Discuss the resulting matrix structures for each approach. Can the Thomas algorithm be applied to 2)?

1) first order

$$f'_j = \frac{f_j - f_{j-1}}{h} + O(h),$$

2) second order

$$f'_j = \frac{f_{j+1} - f_{j-1}}{2h} - \frac{h^2}{6} f'''_j + \cdots.$$

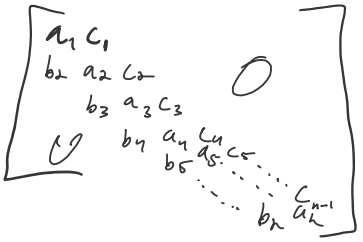
3) fourth order

$$f'_j = \frac{f_{j-2} - 8f_{j-1} + 8f_{j+1} - f_{j+2}}{12h} + O(h^4).$$

4) Padé scheme

$$f'_{j+1} + f'_{j-1} + 4f'_j = \frac{3}{h}(f_{j+1} - f_{j-1}) + \frac{h^4}{30} f^v_j,$$

$$f'_{j+1} + f'_{j-1} + 4f'_j = \frac{3}{h}(f_{j+1} - f_{j-1}) + \frac{h^4}{30} f^v_j,$$



Begin:

$$f'_1 + f'_N + 4f'_0 = \frac{3}{h}(f_1 - f_n)$$

end:

$$f'_0 + f'_{N-1} + 4f'_N = \frac{3}{h}(f_0 - f_{n-1})$$

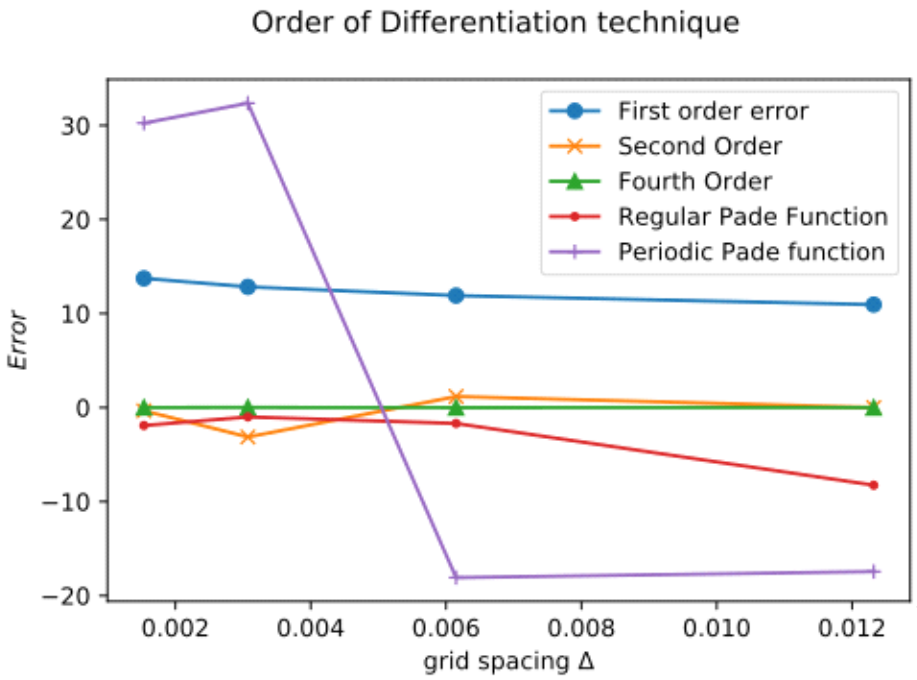
$$f'_0 + 2f'_1 = \frac{1}{h}\left(-\frac{5}{2}f_0 + 2f_1 + \frac{1}{2}f_2\right)$$

$$f'_n + 2f'_{n-1} = \frac{1}{h}\left(\frac{5}{2}f_n - 2f_{n-1} - \frac{1}{2}f_{n-2}\right).$$

you cannot use Thomas algorithm because you lose the tridiagonal structure when you exploit the periodicity of the function

Part iv

Friday, February 7, 2020 9:27 PM



This was obtained by using the code in folder 2.1 labelled plotting. For each difference scheme I made a method to calculate a bin and another method to collect all the bins. For example the second order scheme was found using:

```
def secondorder(f1,f3,h):
    answer=(f3-f1)/(2*h)
    return answer
def secondorderwhole(listx,listy,h):
    L=len(listy)
    y=[]
    x=[]

    for i in range(1,L-1,2):
        y.append(secondorder(listy[i],listy[i+1],h))
        x.append(listx[i])
    return x,y
```

I then calculated the same values in the analytic solution and took the differences at each point between the exact and approximate. I then divided these differences by the corresponding exact value at that point. I did this for each method and at each given bin size to get ten plot points.

Code

Monday, February 17, 2020 6:23 AM

a small snippet of the code used to do this can be seen here:

```
x2,y2=secondorderwhole(test[i].copy(),y[i],h[i])
```

test[] is a list of x values for the given y values

```
secondorder.append((((y2-x2)**2)**0.5)/x2)
```

```
for x in secondorder:
    x[x == np.inf] = 0
    errorsecond.append(np.mean(x))
```

Monday, February 17, 2020 10:07 AM

```
Derivative code sets up several methods for taking finite differences
"""
Created on Tue Feb 11 23:32:56 2020

@author: shaun
"""
import numpy as np
def firstorder(f1,f2,h):
   answer=(f1-f2)/h
   return answer
def firstorderwhole(listx,listy,h):
   L=len(listy)
   y=[]
   x=[]
   for i in range(0,L-1):
      y.append(firstorder(listy[i],listy[i+1],h))
      x.append(listx[i])
   return [x,y]

def secondorder(f1,f3,h):
      y.append(secondorder(listy[i],listy[i+1],h))
      x.append(listx[i])
   answer=(f3-f1)/(2*h)
   return answer
def secondorderwhole(listx,listy,h):
   L=len(listy)
   y=[]
   x=[]

   for i in range(1,L-1,2):
   return x,y
def fourthorder(f1,f2,f3,f4,h):
   f=(f1-8*f2+8*f3-f4)/12*h
   return f
def fourthorderwhole(listx,listy,h):
   y=[]
   x=[]
   L=len(listy)
   for i in range(3,L-2,3):
      y.append(fourthorder(listy[i-2],listy[i-1],listy[i+1],listy[i+2],h))
      x.append(listx[i])
   return x,y

def fourthorders(f1,f2,f3,f4,h):
   A=np.matrix([[1,1,1,1],[0,1,2,3],[0,0.5,2,9/2],[0,1/6,8/6,27/6]],float)
   B=np.matrix([[0],[-1/h],[0],[0]],float)
   A_inverse= np.linalg.inv(A)
   a=A_inverse*B
   answer=a[0]*f1+a[1]*f2+a[2]*f3+a[3]*f4
   return answer

def tridiag(a, b, c, k1=-1, k2=0, k3=1):
   return np.diag(a, k1) + np.diag(b, k2) + np.diag(c, k3)
```

Pade scheme

Monday, February 17, 2020 10:08 AM

```
#pade scheme
def pade(a,b,c,d,l,F):
    A=[]
    B=[]
    C=[]
    D=[]
    B.append(b[0])
    C.append(c[0])
    for k in range(0,l-2):
        A.append(a[1])
        B.append(b[1])
        C.append(c[1])
    B.append(b[2])
    A.append(a[2])

    D.append(d[0][0]*F[0]+d[0][1]*F[1]+d[0][2]*F[2])
    for k in range(0,len(F)-2):
        D.append(d[1][0]*(F[k+2]-F[k]))
    D.append(d[2][0]*F[-3]+d[2][1]*F[-2]+d[2][2]*F[-1])

    T = tridiag(A, B, C)
    print(T)
    alpha=[B[0]]
    beta=[]
    victor=[D[0]]
    N=len(A)
    f=[]
    for k in range(0,N):
        beta.append((A[k]/alpha[k]))
        alpha.append(B[k+1]-C[k]*beta[k])
    for k in range(0,N):
        victor.append(D[k+1]-beta[k]*victor[-1])
    f.append(victor[-1]/alpha[-1])
    for k in range(N-1,-1,-1):
      f.append((victor[k]-C[k]*f[-1])/alpha[k])
    f.reverse()
    return f

#use this if you do not have a triangular structure
def badpade(a,b,c,d,l,F):
    A=[]
    B=[]
    C=[]
    D=[]
    B.append(b[0])
    C.append(c[0])
    for k in range(0,l-2):
        A.append(a[1])
        B.append(b[1])
        C.append(c[1])
    B.append(b[2])
    A.append(a[2])

    D.append([d[1][0]*(F[1]-F[l-1])])
    for k in range(0,len(F)-2):
        D.append([d[1][0]*(F[k+2]-F[k])])
    D.append([d[1][0]*(F[0]-F[l-1])])
    T = tridiag(A, B, C)
    T[0,2]=c[2]
    T[-1,-3]=a[0]
    D=np.array(D)
    IT=np.linalg.inv(T)*(D.transpose())
    f=np.matmul(IT,D)
    return f
d=np.array([[1,2,3],[3,0,0],[7,8,9]], int)
ans=badpade([-5,1,2],[1,4,1],[2,1,82],d,5,[1,1,1,1,1])
#print(pade([0,1,2],[1,4,1],[2,1,0],d,5,[1,1,1,1,1]))
print(ans)
```

2.1

Friday, February 7, 2020 12:42 AM

1. Consider the central finite difference operator $\delta/\delta x$ defined by

$$\frac{\delta u_n}{\delta x} = \frac{u_{n+1} - u_{n-1}}{2h}.$$

(a) In calculus we have

$$\frac{duv}{dx} = u\frac{dv}{dx} + v\frac{du}{dx}.$$

Does the following analogous finite difference expression hold?

$$\frac{\delta(u_n v_n)}{\delta x} = u_n\frac{\delta v_n}{dx} + v_n\frac{\delta u_n}{\delta x}.$$

a) $\frac{\delta u_n v_n}{\delta x} = \frac{u_{n+1}v_{n+1} - u_{n-1}v_{n-1}}{2h}$

$u_n \frac{\delta v_n}{\delta x} + v_n \frac{\delta u_n}{\partial x}$ (product rule)

$\frac{u_{n+1} - v_{n-1}}{2h} u_n + \frac{v_{n+1} - u_{n-1}}{2h} v_n$

factor to get into this form and we can see that they are not equal.

(b) Show that

$$\frac{\delta(u_n v_n)}{\delta x} = \bar{u}_n\frac{\delta v_n}{\delta x} + \bar{v}_n\frac{\delta u_n}{\delta x}$$

where an overbar indicates average over the nearest neighbors,

$$\bar{u}_n = \frac{1}{2}(u_{n+1} + u_{n-1}).$$

(c) Show that

$$\phi\frac{\delta\psi}{\delta x} = \frac{\delta}{\delta x}\bar{\phi}\psi - \overline{\psi\frac{\delta\phi}{\delta x}}.$$

(d) Derive a finite difference formula for the second-derivative operator that is obtained from two applications of the first-derivative finite difference operator. Compare the leading error term of this formula and the popular second-derivative formula

$$\frac{u_{n+1} - 2u_n + u_{n-1}}{h^2}.$$

Use both schemes to calculate the second derivative of $\sin 5x$ at $x = 1.5$. Plot the absolute values of the errors as a function of h on a log–log plot similar to Figure 2.1. Use $10^{-4} \le h \le 10^{0}$. Discuss your plot.

b) $\frac{\delta(u_n v_n)}{\delta x} = \bar{u}_n \frac{\delta v_n}{\delta x} + \bar{v}_n \frac{\delta u_n}{\delta x}$

$\bar{u} = \frac{1}{2}(u_{n+1} + u_{n-1})$

$u_n \frac{\delta v_n}{\delta x} + v_n \frac{\delta u_n}{\delta x} = \frac{u_{n+1} - v_{n-1}}{2h} u_n + \frac{v_{n+1} - u_{n-1}}{2h} v_n$

from part a we can then get

$u_n \frac{\delta v_n}{\delta x} + v_n \frac{\delta u_n}{\delta x} = \frac{u_n \delta v}{\delta x} + u_n \frac{\delta u_n}{\delta x}$

$$\frac{\delta \phi_n \psi_n}{\delta x} = \frac{\phi_{n+2}\psi_{n+1} + \phi_n \psi_{n+1} - \phi_n \psi_{n-1} - \phi_{n-2}\psi_{n-1}}{4h}$$

$$\psi_n \frac{\delta \phi_n}{\delta x} = \frac{\phi_{n+2}\psi_{n-1} + \phi_n \psi_{n+1} - \phi_n \psi_{n+1} - \phi_{n-2}\psi_{n-1}}{4h}$$

$$\psi_n \frac{\delta \phi_n}{\delta x} - \frac{\delta \phi_n \psi_n}{\delta x} =$$

$$\frac{\phi_{n+2}\psi_{n+1} + \phi_n \psi_{n+1} - \phi_n \psi_{n-1} - \phi_{n-2}\psi_{n-1}}{4h} - \frac{\phi_{n+2}\psi_{n-1} + \phi_n \psi_{n+1} - \phi_n \psi_{n+1} - \phi_{n-2}\psi_{n-1}}{4h}$$

$$\frac{-\phi_n \psi_{n-1} + \phi_n \psi_{n+1}}{2h} = \phi_n \frac{\psi_{n+1} - \psi_{n-1}}{2h} = \phi_n \frac{\delta \psi_n}{\delta x}$$

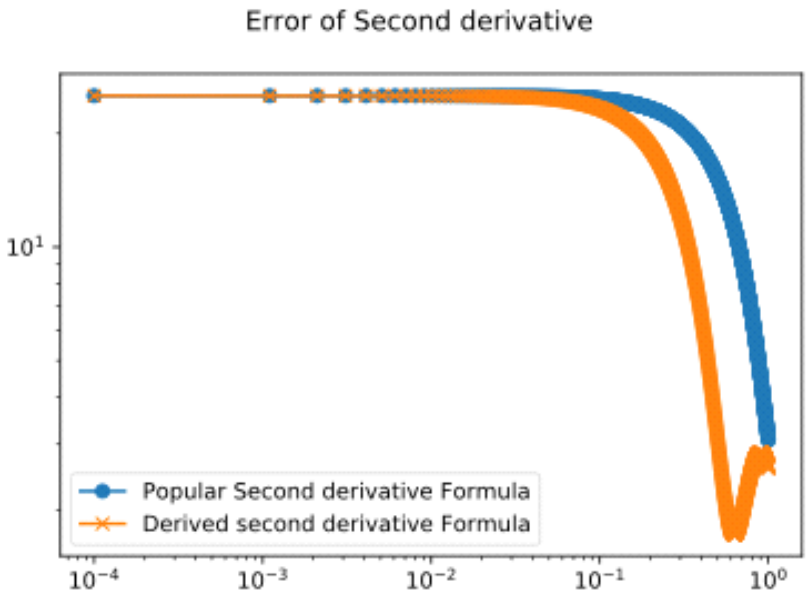
d) Taylor expand about x

$$u_{n+2} = u_n + 2h u'_n + 2h^2 u''_n$$

$$u_{n-2} = u_n - 2h u'_n + 2h^2 u''_n$$

take a linear combo to isolate u''

$$u''_n = \frac{u_{n+2} - 2u_n + u_{n-2}}{4h^2} - O(h^2)$$


Error of Second derivative
10^1
Popular Second derivative Formula
Derived second derivative Formula
10^{-4}
10^{-3}
10^{-2}
10^{-1}
10^0

2.3

Monday, February 17, 2020 7:52 AM

$$f'_{j+1} + f'_{j-1} + 4f'_j = \frac{3}{h}(f_{j+1} - f_{j-1}) + \frac{h^4}{30} f^v_j,$$

3. Verify that the modified wavenumber for the fourth-order Padé scheme for the first derivative is

$$k' = \frac{3\sin(k\Delta)}{\Delta(2 + \cos(k\Delta))}.$$

$$ik'\left[e^{ik(x_j+\Delta)} + 4e^{ikx_j} + e^{ik(x_j-\Delta)}\right] = \frac{3}{\Delta}\left[e^{ik(x_j+\Delta)} - e^{ik(x_j-\Delta)}\right]$$

$$ik' = \frac{\frac{3}{\Delta}\left[e^{ik\Delta}e^{ikx_j} - e^{-ik\Delta}e^{ikx_j}\right]}{e^{ik\Delta}e^{ikx_j} + 4e^{ikx_j} + e^{-ik\Delta}e^{ikx_j}}$$

$$ik' = \frac{\frac{3}{\Delta}\left[e^{ik\Delta}e^{ikx_j} - e^{-ik\Delta}e^{ikx_j}\right]}{e^{ik\Delta}e^{ikx_j} + 4e^{ikx_j} + e^{-ik\Delta}e^{ikx_j}}$$

$$ik' = \frac{\frac{3}{\Delta}\left[e^{ik\Delta} - e^{-ik\Delta}\right]}{e^{ik\Delta} + 4 + e^{-ik\Delta}}$$

$$e^{ik\Delta} - e^{-ik\Delta} = \cos(ik\Delta) + i\sin(ik\Delta) - \cos(ik\Delta) + i\sin(ik\Delta)$$

$$e^{ik\Delta} - e^{-ik\Delta} = 2\sin(ik\Delta)$$

$$e^{ik\Delta} + e^{-ik\Delta} = \cos(ik\Delta) + \sin(ik\Delta) + \cos(ik\Delta) - \sin(ik\Delta)$$

$$e^{ik\Delta} + e^{-ik\Delta} = 2\cos(ik\Delta) \quad \text{subbing in}$$

$$ik' = \frac{3}{\Delta}\cdot\frac{2i\sin(ik\Delta)}{4 + 2\cos(ik\Delta)} = \frac{3}{\Delta}\frac{i\sin(ik\Delta)}{2 + \cos ik\Delta}$$

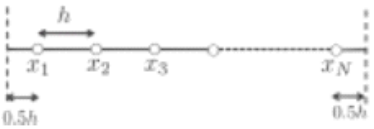
$$k' = \frac{3}{\Delta}\frac{\sin(ik\Delta)}{2 + \cos(ik\Delta)}$$

8. In numerical solution of boundary value problems in differential equations, we can sometimes use the physics of the problem not only to enforce boundary conditions but also to maintain high-order accuracy near the boundary. For example, we may know the heat flux through a surface or displacement of a beam specified at one end. We can use this information to produce better estimates of the derivatives near the boundary.

Suppose we want to numerically solve the following boundary value problem with Neumann boundary conditions:

$$\frac{d^2y}{dx^2} + y = x^3, \qquad 0 \le x \le 1$$
$$y'(0) = y'(1) = 0.$$

We discretize the domain using grid points $x_i = (i - 0.5)h, i = 1, \ldots, N$. Note that there are no grid points on the boundaries as shown in the figure below. In this problem, y_i is the numerical estimate of y at x_i. By using a finite difference scheme, we can estimate y_i'' in terms of linear combinations of y_i's and transform the ODE into a linear system of equations.



Use the fourth-order Padé formula (2.19) for the interior points.

(a) For the left boundary, derive a third-order Padé scheme to approximate y_0'' in the following form:

$$y_1'' + b_2 y_2'' = a_1 y_1 + a_2 y_2 + a_3 y_3 + a_4 y_b' + \mathcal{O}(h^3),$$

where $y_b' = y'(0)$, which is known from the boundary condition at $x = 0$.

(b) Repeat the previous step for the right boundary.

(c) Using the finite difference formulae derived above, we can write the following linear relation:

$$\mathbf{A} \begin{bmatrix} y_1'' \\ \vdots \\ \vdots \\ y_N'' \end{bmatrix} = \mathbf{B} \begin{bmatrix} y_1 \\ \vdots \\ \vdots \\ y_N \end{bmatrix}.$$

What are the elements of the matrices **A** and **B** operating on the interior and boundary nodes?

(d) Use this relationship to transform the ODE into a system with y_i's as unknowns. Use $N = 24$ and solve this system. Do you actually have to invert A? Plot the exact and numerical solutions. Discuss your result. How are the Neumann boundary conditions enforced into the discretized boundary value problem?

A

Thursday, February 13, 2020 9:06 AM

$$y_1'' + b_2 y_2'' = a_1 y_1 + a_2 y_2 + a_3 y_3 + a_4 y_b' + \mathcal{O}(h^3),$$

$$\frac{d^2y}{dx^2} + y = x^3, \qquad 0 \le x \le 1$$
$$y'(0) = y'(1) = 0.$$

$$y_1'' + b_2 y_2'' - (a_1 y_1 + a_2 y_2 + a_3 y_3 + a_4 y_b') = O(h^3)$$

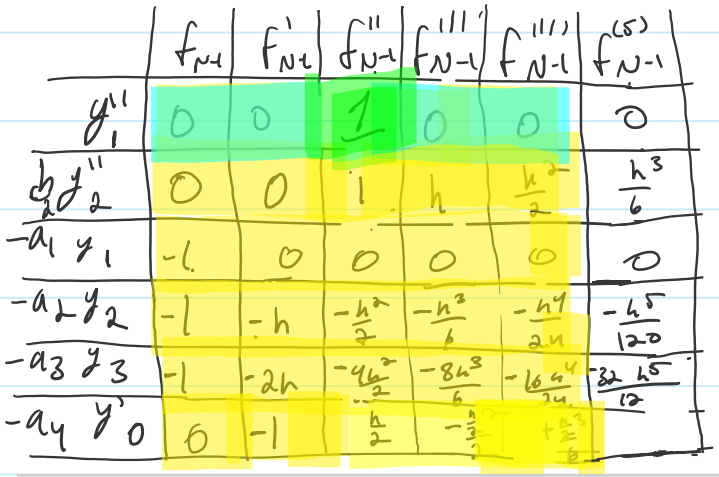
$$y_1'' + b_2 y_2'' - a_1 y_1 - a_2 y_2 - a_3 y_3 - a_4 y_b' = O(h^3)$$

	f	f'	f''	f'''	f''''	$f^{(5)}$
y_1''	0	0	1	0	0	0
$b_2 y_2''$	0	0	1	h	$\frac{h^2}{2}$	$\frac{h^3}{6}$
$-a_1 y_1$	-1	0	0	0	0	0
$-a_2 y_2$	-1	$-h$	$-\frac{h^2}{2}$	$-\frac{h^3}{6}$	$-\frac{h^4}{2\cdot 4}$	$-\frac{h^5}{120}$
$-a_3 y_3$	-1	$-2h$	$-\frac{4h^2}{2}$	$-\frac{8h^3}{6}$	$-\frac{16h^4}{24}$	$-\frac{32h^5}{120}$
$-a_4 y_b'$	0	-1	$\frac{h}{2}$	$\frac{(\frac{h}{2})^2}{2}$	$\frac{(\frac{h}{2})^3}{6}$	$\frac{(\frac{h}{2})^4}{24}$

$$\begin{bmatrix} 0 & -1 & -1 & -1 & 0 \\ 0 & 0 & -h & -2h & -1 \\ 1 & 0 & -\frac{h^2}{2} & -\frac{(2h)^2}{2} & -\frac{h^2}{2} \\ h & 0 & -\frac{h^3}{6} & -\frac{(2h)^3}{6} & -\frac{h^3}{6} \\ \frac{h^2}{2} & 0 & -\frac{h^4}{24} & -\frac{(2h)^4}{24} & -\frac{h^4}{24} \end{bmatrix} \begin{bmatrix} b_2 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -1 \\ 0 \\ 0 \end{bmatrix}$$

A

Thursday, February 13, 2020 7:45 PM



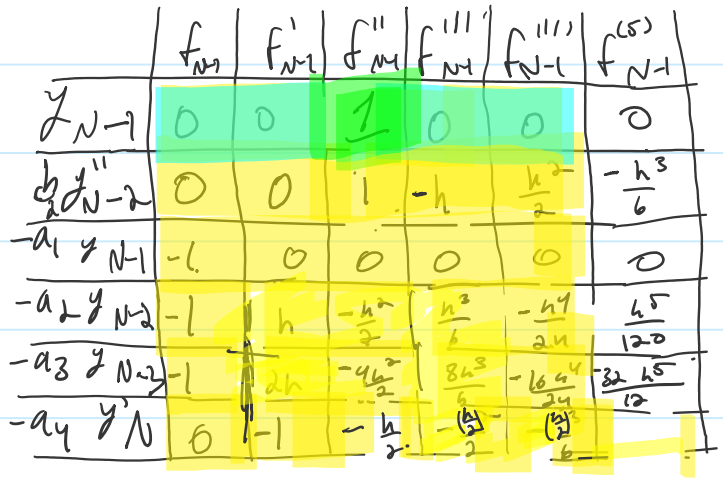
	f_{N+1}	f'_{N+1}	f''_{N-1}	f'''_{N-1}	f''''_{N-1}	$f^{(5)}_{N-1}$
y''_1	0	0	1	0	0	0
$b_2 y''_2$	0	0	1	h	$\frac{h^2}{2}$	$\frac{h^3}{6}$
$-a_1 y_1$	-1	0	0	0	0	0
$-a_2 y_2$	-1	$-h$	$-\frac{h^2}{2}$	$-\frac{h^3}{6}$	$-\frac{h^4}{24}$	$-\frac{h^5}{120}$
$-a_3 y_3$	-1	$-2h$	$-\frac{4h^2}{2}$	$-\frac{8h^3}{6}$	$-\frac{16h^4}{24}$	$-\frac{32h^5}{120}$
$-a_4 y'_0$	0	-1	$\frac{h}{2}$	$-\frac{h^2}{3}$	$+\frac{h^3}{6}$	

$$y''_1 + b_2 y''_2 = a_1 y_1 + a_2 y_2 + a_3 y_3 + a_4 y'_b + \mathcal{O}(h^3),$$

$$\left\{ a_1 : -\frac{36}{23h^2},\ a_2 : \frac{48}{23h^2},\ a_3 : -\frac{12}{23h^2},\ a_4 : \frac{24}{23h},\ b : -\frac{11}{23} \right\}$$

B

Thursday, February 13, 2020 10:00 PM



$$\left\{ a_1 : -\frac{36}{23h^2},\ a_2 : \frac{48}{23h^2},\ a_3 : -\frac{12}{23h^2},\ a_4 : \frac{24}{23h},\ b : -\frac{11}{23} \right\}$$

C

Friday, February 14, 2020 11:47 AM

$$y_1'' + b_2 y_2'' = a_1 y_1 + a_2 y_2 + a_3 y_3 + a_4 y_b' + \mathcal{O}(h^3).$$

$$b_1 y_{j-1}'' + y_j'' + b_2 y_{j+1}'' = a_1 y_{j-1} + a_2 y_j + a_3 y_{j+1} + a_4 y_{j-1}' + O(h^3)$$

taylor table

	f_j	f_j'	f_j''	f_j'''	$f_j^{(4)}$	$f_j^{(5)}$	$f^{(6)}$	$f^{(7)}$
y_j''	0	0	1	0	0	0	0	0
$b_1 y_{j-1}''$	0	0	1	$-h$	$\frac{h^2}{2}$	$\frac{-h^3}{6}$		
$b_2 y_{j+1}''$	0	0	1	h	$\frac{h^2}{2}$	$\frac{h^3}{6}$		
$-a_1 y_j$	-1	0	0	0	0	0		
$-a_2 y_{j-1}$	-1	h	$\frac{-(h)^2}{2}$	$\frac{(h)^3}{6}$	$\frac{-(h)^4}{4!}$	$\frac{(h)^5}{5!}$		
$-a_3 y_{j+1}$	-1	$-h$	$\frac{-h^2}{2}$	$\frac{-h^3}{6}$	$\frac{-h^4}{4!}$	$\frac{-h^5}{5!}$		
$-a_4 y_{j-1}'$	0	-1	h	$\frac{-(h)^2}{2}$	$\frac{(h)^3}{6}$	$\frac{(h)^4}{4!}$		

0	0	1	-1	-1	0
0	0	0	h	$-h$	-1
1	1	0	$\frac{-(h)^2}{2}$	$\frac{-h^2}{2}$	h
$-h$	h	0	$\frac{(h)^3}{6}$	$\frac{-h^3}{6}$	$\frac{-(h)^2}{2}$
$\frac{h^2}{2}$	$\frac{h}{2}$	0	$\frac{-(h)^4}{4!}$	$\frac{-h^4}{4!}$	$\frac{h^3}{6}$
$\frac{-h^3}{6}$	$\frac{h^3}{6}$	0	$\frac{-h^5}{5!}$	$\frac{-(h)^5}{5!}$	$\frac{(h)^4}{4!}$

$$\begin{bmatrix} b_1 \\ b_2 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

C

Saturday, February 15, 2020 2:07 PM

$$y_1'' + b_2 y_2'' = a_1 y_1 + a_2 y_2 + a_3 y_3 + a_4 y_b' + \mathcal{O}(h^3).$$

$$\left\{ a_1 : \frac{24}{5h^2},\ a_2 : -\frac{12}{5h^2},\ a_3 : -\frac{12}{5h^2},\ b_1 : -\frac{17}{10},\ b_2 : -\frac{17}{10} \right\}$$

$$b_1 y''_{j-1} + y''_j + b_2 y''_{j+1} = a_1 y_j + a_2 y_{j-1} + a_3 y_{j+1}$$

$$-\frac{17}{10} y_{j-1} + y''_j + \frac{-17}{10} y''_{j+1} = \frac{1}{h^2}\left(\frac{24}{5} y_j - \frac{12}{5} y_{j-1} + -\frac{12}{5} y_{j+1} \right)$$

$$-\frac{17}{10} y''_{j-1} + y_j'' + \frac{-17}{10} y''_{j+1} = \frac{-5}{12} \cdot \frac{1}{h^2}\left(y_{j-1} - 2y_j + y_{j+1} \right)$$

$$\frac{102}{25} y''_{j-1} - \frac{5}{12} y_j + \frac{102}{25} y''_{j+1} = \frac{1}{h^2}\left(y_{j-1} - 2y_j + y_{j+1} \right)$$

$A\, y'' = B\, y$ $\left\{ a_1 : -\frac{36}{23h^2},\ a_2 : \frac{48}{23h^2},\ a_3 : -\frac{12}{23h^2},\ a_4 : \frac{24}{23h},\ b : -\frac{11}{23} \right\}$ = boundaries

A

$$\begin{bmatrix} 1 & -\frac{11}{23} & 0 & \cdots & 0 \\ \frac{102}{25} & -\frac{5}{12} & \frac{102}{25} & 0 & \\ 0 & \ddots & \ddots & \ddots & 0 \\ & 0 & \frac{102}{25} & -\frac{5}{12} & \frac{102}{25} \\ 0 & \cdots & 0 & -\frac{11}{23} & 1 \end{bmatrix} \begin{bmatrix} y''_1 \\ y''_2 \\ \vdots \\ y''_{N-2} \\ y''_{N-1} \end{bmatrix}$$

$$= \frac{1}{h^2} \begin{bmatrix} -\frac{36}{23} & \frac{48}{23} & -\frac{12}{23} & \cdots & 0 \\ 1 & -2 & 1 & & \\ 0 & \ddots & \ddots & \ddots & 0 \\ & 1 & -2 & 1 & \\ 0 & \cdots & -\frac{12}{23} & \frac{48}{25} & -\frac{36}{23} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{N-1} \\ y_N \end{bmatrix}$$

B

Part D)

$A y'' = B y \qquad y'' = x^3 - y$

$y'' = A^{-1} B y \rightarrow \quad A^{-1} B y = x^3 - y \rightarrow A^{-1} B y + y = x^3$

$(A^{-1} B + I)\, y = x^3$

part D

Friday, February 14, 2020 11:50 AM

Implement

1) make tridiagnol A and B

A

1	$\frac{-11}{23}$	0	←	0	y_1''
$\frac{102}{25}$	$\frac{-5}{12}$	$\frac{102}{25}$	0	↓	y_2''
0	⋱	⋱	⋱	0	⋮
↑	0	$\frac{102}{25}$	$\frac{-5}{12}$	$\frac{102}{25}$	y_{N-2}''
0	→	0	$\frac{-11}{23}$	1	y_{N-1}''

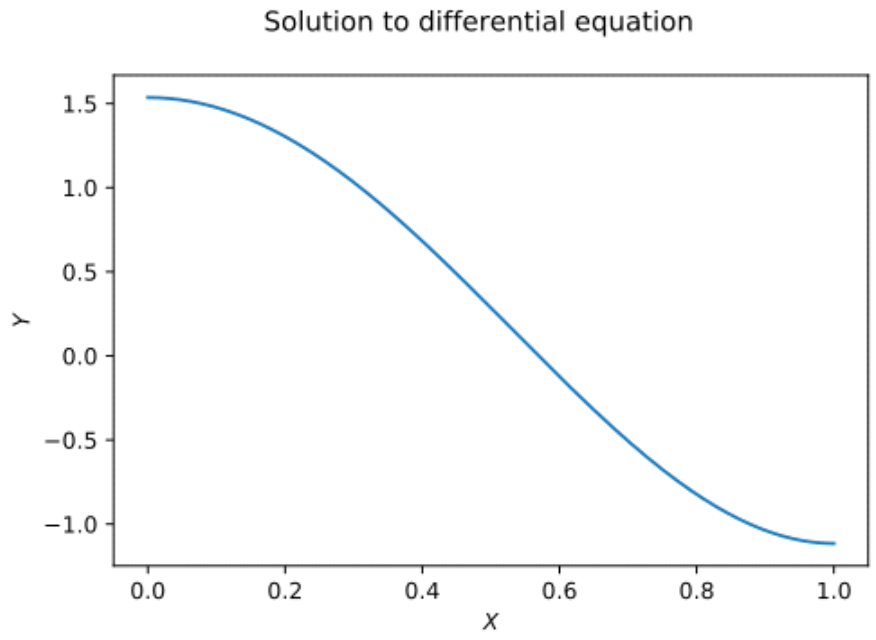
$= \frac{1}{h^2}$

B

$\frac{-26}{23}$	$\frac{48}{23}$	$\frac{-12}{23}$	←	0	y_1
1	-2	1	↙	↓	y_2
0	⋱	⋱	⋱	0	⋮
↑	→	1	-2	1	y_{N-1}
0	→	$\frac{-12}{23}$	$\frac{48}{25}$	$\frac{-26}{23}$	y_{N-2}

2) do $(A^{-1}B + I)y = x^3$

3) Plot x vs y



Part D

Monday, February 17, 2020 9:19 AM

The neuman boundary conditions are enforced using the end corrections by requiring that $f'(0)=0$ we force the necessary boundaries. The result produced a cubic like function.

code

Monday, February 17, 2020 10:28 AM

In [1]:
```
import numpy as np
from sympy import *
from IPython.display import display
init_printing(use_latex='mathjax')
```

In [2]:
```
h = symbols('h',Real=True)
#A=Matrix([[0,1,1,1,0],[0,0,h,2*h,1],[1/h,0,(h)/2,-h/2,-1],[0,0,(h**3)/6,8*(h**3)/6,((-h/2)**2)/2],[0,0,(h**4)/24,16*(h**4)/24,((-h/2)**3)/6]])


row1=[0,-1,-1,-1,0,0]
row2=[0,0,-(h),(-2*h),-1,0]
row3=[1,0,-((h)**2)/2,-((2*h)**2)/2,h/2,-1]
row4=[h,0,-((h)**3)/6,-((2*h)**3)/6,-((-h/2)**2)/2,0]
row5=[(h**2)/2,0,-((h)**4)/24,-((2*h)**4)/24,-(((-h/2))**3)/6,0]
A=Matrix([row1,row2,row3,row4,row5])
```

In [3]:
```
display(A)
```

$$\begin{bmatrix} 0 & -1 & -1 & -1 & 0 & 0 \\ 0 & 0 & -h & -2h & -1 & 0 \\ 1 & 0 & -\frac{h^2}{2} & -2h^2 & \frac{h}{2} & -1 \\ h & 0 & -\frac{h^3}{6} & -\frac{4h^3}{3} & -\frac{h^2}{8} & 0 \\ \frac{h^2}{2} & 0 & -\frac{h^4}{24} & -\frac{2h^4}{3} & \frac{h^3}{48} & 0 \end{bmatrix}$$

In [4]:
```
system = A
b, a1, a2, a3, a4 = symbols('b a1 a2 a3 a4')
solve_linear_system(system, b,a1,a2,a3,a4)
```

Out[4]:
$$\left\{a_1 : -\frac{36}{23h^2},\ a_2 : \frac{48}{23h^2},\ a_3 : -\frac{12}{23h^2},\ a_4 : -\frac{24}{23h},\ b : -\frac{11}{23}\right\}$$

In [5]:
```
#This goes into the python script I made
X=solve_linear_system(system, b,a1,a2,a3,a4)
print(X)
```

```
{b: -11/23, a1: -36/(23*h**2), a2: 48/(23*h**2), a3: -12/(23*h**2), a4: -24/(23*h)}
```

This is for part B

In [6]:
```
h=-h
row1=[0,-1,-1,-1,0,0]
```

In [6]:
```
h=-h
row1=[0,-1,-1,-1,0,0]
row2=[0,0,-(h),(-2*h),-1,0]
row3=[1,0,-((h)**2)/2,-((2*h)**2)/2,h/2,-1]
row4=[h,0,-((h)**3)/6,-((2*h)**3)/6,-((-h/2)**2)/2,0]
row5=[(h**2)/2,0,-((h)**4)/24,-((2*h)**4)/24,-(((-h/2))**3)/6,0]
B=Matrix([row1,row2,row3,row4,row5])
```

In [7]:
```
display(B)
```

$$\begin{bmatrix} 0 & -1 & -1 & -1 & 0 & 0 \\ 0 & 0 & h & 2h & -1 & 0 \\ 1 & 0 & -\frac{h^2}{2} & -2h^2 & -\frac{h}{2} & -1 \\ -h & 0 & \frac{h^3}{6} & \frac{4h^3}{3} & -\frac{h^2}{8} & 0 \\ \frac{h^2}{2} & 0 & -\frac{h^4}{24} & -\frac{2h^4}{3} & -\frac{h^3}{48} & 0 \end{bmatrix}$$

In [8]:
```
system = B
b, a1, a2, a3, a4 = symbols('b a1 a2 a3 a4')
solve_linear_system(system, b,a1,a2,a3,a4)
```

Out[8]:
$$\left\{a_1 : -\frac{36}{23h^2},\ a_2 : \frac{48}{23h^2},\ a_3 : -\frac{12}{23h^2},\ a_4 : \frac{24}{23h},\ b : -\frac{11}{23}\right\}$$

In [9]:
```
h=-h
```

This is for part C

In [11]:
```
row1=[0,0,-1,-1,-1,0]
row2=[0,0,0,(h),-(h),0]
row3=[1,1,0,-((h)**2)/2,-((h)**2)/2,-1]
row4=[-h,h,0,((h)**3)/6,-((h)**3)/6,0]
row5=[(h**2)/2,(h**2)/2,0,-((2*h)**4)/24,-((h)**4)/24,0]

C=Matrix([row1,row2,row3,row4,row5])
display(C)
```

$$\begin{bmatrix} 0 & 0 & -1 & -1 & -1 & 0 \\ 0 & 0 & 0 & h & -h & 0 \\ 1 & 1 & 0 & -\frac{h^2}{2} & -\frac{h^2}{2} & -1 \\ -h & h & 0 & \frac{h^3}{6} & -\frac{h^3}{6} & 0 \\ \frac{h^2}{2} & \frac{h^2}{2} & 0 & -\frac{2h^4}{3} & -\frac{h^4}{24} & 0 \end{bmatrix}$$

In [12]:
```
system = C
b1,b2, a1, a2, a3= symbols('b1 b2 a1 a2 a3 ')
```

In [12]:
```
system = C
b1,b2, a1, a2, a3= symbols('b1 b2 a1 a2 a3 ')
solve_linear_system(system, b1,b2,a1,a2,a3)
```

Out[12]:
$$\left\{a_1 : \frac{24}{5h^2},\ a_2 : -\frac{12}{5h^2},\ a_3 : -\frac{12}{5h^2},\ b_1 : -\frac{17}{10},\ b_2 : -\frac{17}{10}\right\}$$

In [ ]:
```

```

Solution Code

Monday, February 17, 2020 10:28 AM

```
from derivatives import *
import matplotlib.pyplot as plt
from sympy import *
def construct(a,b,c,l):
    A=[]
    B=[]
    C=[]
    D=[]
    B.append(b[0])
    C.append(c[0])
    for k in range(0,l-2):
        A.append(a[1])
        B.append(b[1])
        C.append(c[1])
    B.append(b[2])
    A.append(a[2])
    T = tridiag(A, B, C)
    T[0,2]=c[2]
    T[-1,-3]=a[0]
    return T
N=1000
x=np.linspace(0,1,N)
h=((x[1]-x[0])**2.0)**0.5
A=construct([0,102.0/25,-11/23],[1,-5/12.0,1],[-11/23,102.0/25,0],N)
B=(1/(h**2))*construct([-12/23.0,1,48/23.0],[-36.0/23,-2,-36.0/23],[48.0/23,1,-12/23],N)

IA=np.linalg.inv(A)
C=np.matmul(IA,B)
C=C+np.identity(N)
y=np.matmul(np.linalg.inv(C),x**3)

figure2=plt.figure()
figure2.suptitle("Solution to differential equation")
ax=figure2.add_subplot(1,1,1)
ax.plot(x,y)
ax.set_xlabel(r"$X$")
ax.set_ylabel(r"$Y$")
```

2.10

Friday, February 7, 2020 12:43 AM

10. *Nonuniform mesh.*
Consider the function $f(x) = 1 - x^8$ and a grid defined as follows:

$$\begin{cases} j = 0, 1, 2, \ldots, N \\ \xi_j = -1 + \frac{2j}{N} \\ x_j = \frac{1}{a}\tanh(\xi_j \tanh^{-1}[a]) \quad 0 < a < 1. \end{cases}$$

The parameter a can be used to adjust the spacing of the grid points, with large a placing more points near the boundaries. For this problem take $a = 0.98$ and $N = 32$.

(a) Compute and plot the derivative of f with the central difference formula (2.20) and the coordinate transformation method described in Section 2.5 and compare with the exact derivative in $-1 \le x < 1$. How would the results change with $a = 0.9$?

(b) Repeat part (a) with the transformation:

$$\begin{cases} j = 0, 1, 2, \ldots, N \\ \xi_j = \frac{\pi_j}{N} \\ x_j = \cos(\xi_j). \end{cases}$$

Which transformation would you choose, the one in part (a) or this one?

(c) How many uniformly spaced grid points would be required to achieve the same accuracy as the transformation method in (a)? The maximum error in the derivative over the domain for the uniform case should be less than or equal to the maximum error over the domain for the transformed case.

$f' = g'(x) \cdot \frac{df}{d\xi_j}$

$\{ = g(x)$ $\quad f'_j = \frac{f_{j+1} - f_{j-1}}{x_{j+1} - x_{j-1}}$

$x_j = \frac{1}{a}\tanh(\xi_j \tan^{-1}[a])$

$a x_j = \tanh(\xi_j \tan^{-1}[a])$

$\frac{\tanh^{-1}(a x_j)}{\tanh^{-1}[a]} = \xi_j \rightarrow$ ~~$\tanh^{-1}(x_j) = \xi_j$~~

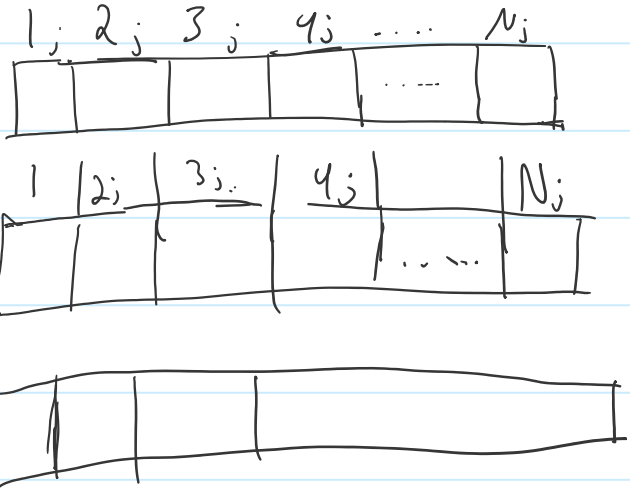
$\frac{d}{dx}\operatorname{artanh} x = \frac{1}{1 - x^2} \rightarrow \frac{a}{1 - x_j^2} \cdot \frac{1}{\tanh^{-1}[a]} = \frac{d\xi_j}{dx}$

$\frac{d\,g(x)}{dx} = \frac{a}{1 - x_j^2} \cdot (\tanh(a)) \cdot \frac{f(x_{i+1}) - f(x_{j-1})}{2 \cdot \Delta\xi}$

$\xi_j = -1 + \frac{2j}{N}$

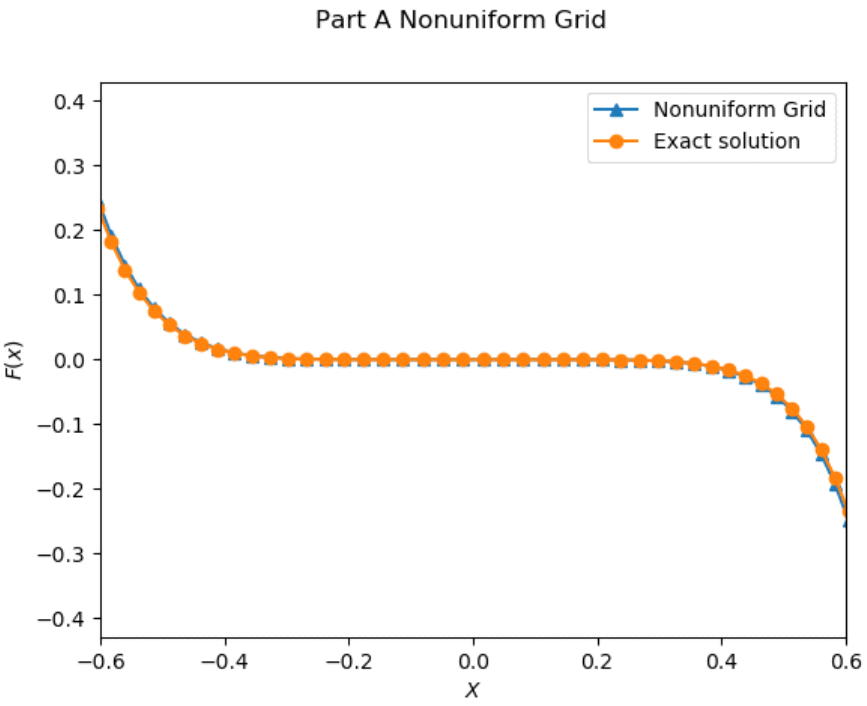
you get x_j values by passing them through ξ_j transform

$X(N) = \xi_j(N)$

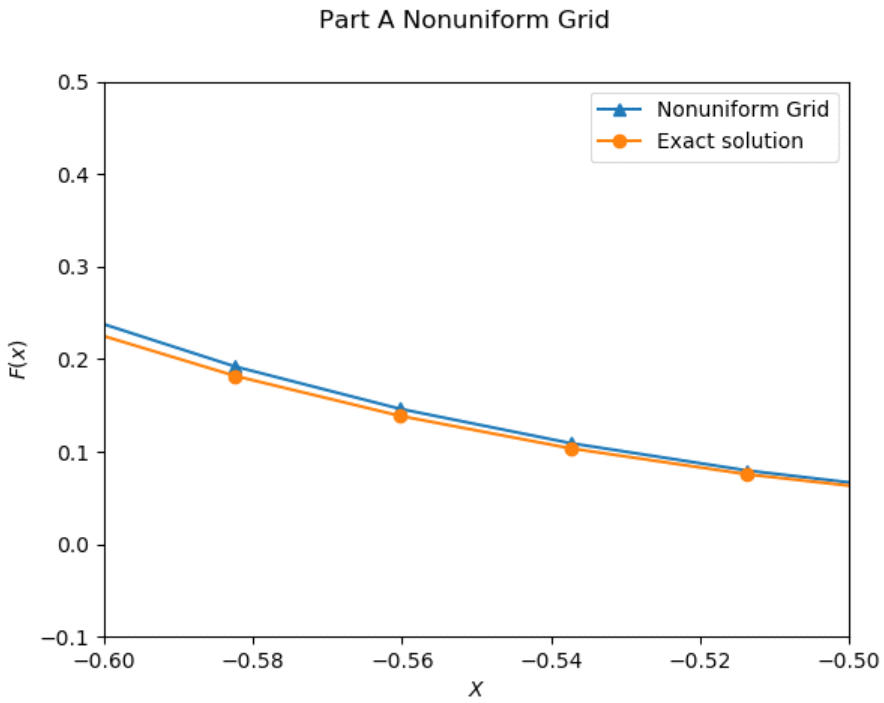


Part A

Monday, February 17, 2020 9:06 AM



I found that changing a spread out the points further from the x=0 center.



Part b

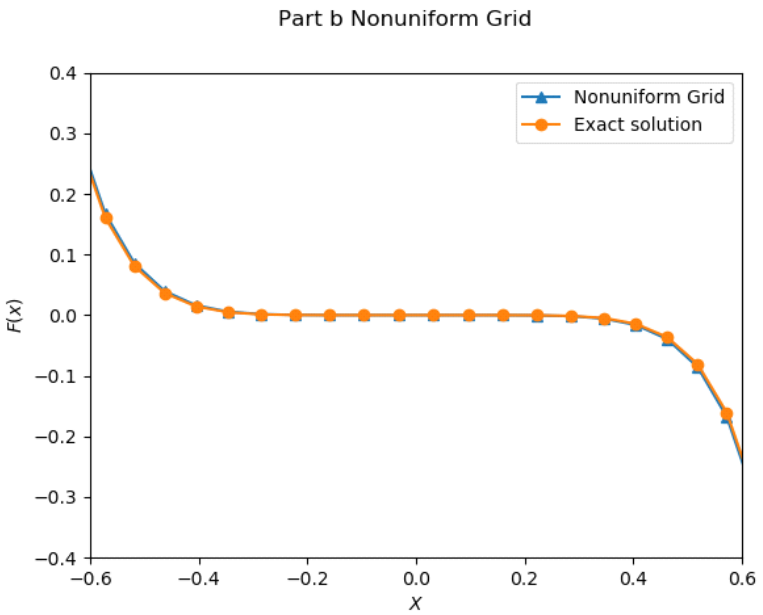
Monday, February 17, 2020 12:25 AM

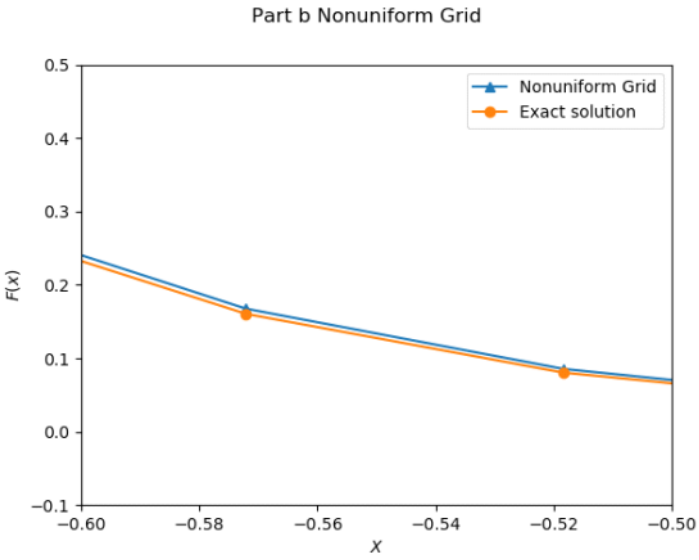
$$\begin{cases} j = 0, 1, 2, \ldots, N \\ \xi_j = \frac{\pi_j}{N} \\ x_j = \cos(\xi_j). \end{cases}$$

$$\cos^{-1}(x_j) = \{_j$$

$$\frac{d}{dx}\cos^{-1}(x_j) = \frac{-1}{\sqrt{1-x^2}}$$

$$\frac{df}{dx} = \frac{-1}{\sqrt{1-x^2}} \cdot \frac{(x_{j+1}) - (x_{j-1})}{2 \cdot \Delta \{}$$





after comparing the two I would have decided I would go more use of a nonuniform grid that allows me to customize where my mesh refines.

code part a

Monday, February 17, 2020 10:32 AM

```
lt.ion()
def function(x):
    f=1-(x**8)
    return f
def exact(x):
    f=-8*(x**7)
    return(f)
N=32
j=np.linspace(0,N)
y=-1+ (2/N)*j
delta=y[1]-y[0]
a=0.9
X=(1/a)*np.tanh(y*np.arctan(a))

central=[]
x=[]
for i in range(1,len(X)-1):
    cd=(function(X[i+1])-function(X[i-1]))/(2.0*delta)
    central.append(cd)
    x.append(X[i])
central= np.asarray(central)
x=np.asarray(x)
df=(a/(1-(x**2)))*(1/np.tanh(a))*central

dfexact=[]
for i in x:
    dfexact.append(exact(i))

figure=plt.figure()
ax=figure.add_subplot(111)
figure.suptitle("Part A Nonuniform Grid")
ax.plot(x,df,marker='^',label="Nonuniform Grid")
ax.plot(x,dfexact,marker='o',label="Exact solution")
ax.legend()
ax.set_xlabel(r"$X$")
ax.set_ylabel(r"$F(x)$")
ax.set_xlim(-0.6,-0.5)
ax.set_ylim(-0.1,0.5)
```

code part b

Monday, February 17, 2020 10:33 AM

```
import numpy as np
import matplotlib.pyplot as plt
from derivatives import *
plt.ion()
def function(x):
   f=1-(x**8)
   return f
def exact(x):
   f=-8*(x**7)
   return(f)
N=32
j=np.linspace(0,N)
y=(np.pi/N)*j
delta=y[1]-y[0]
a=0.98
X=np.cos(y)

central=[]
x=[]
for i in range(1,len(X)-1):
   cd=(function(X[i+1])-function(X[i-1]))/(2.0*delta)
   central.append(cd)
   x.append(X[i])
central= np.asarray(central)
x=np.asarray(x)
df=(-1/((1-(x**2))**(0.5)))*central
dfexact=[]
for i in x:
   dfexact.append(exact(i))

figure=plt.figure()
ax=figure.add_subplot(111)
figure.suptitle("Part b Nonuniform Grid")
ax.plot(x,df,marker='^',label="Nonuniform Grid")
ax.plot(x,dfexact,marker='o',label="Exact solution")
ax.legend(loc='upperleft')
ax.set_xlabel(r"$X$")
ax.set_ylabel(r"$F(x)$")
ax.set_xlim(-0.6,-0.5)
ax.set_ylim(-0.1,0.5)
```

3.2

Friday, February 7, 2020 12:43 AM

2. Show that

$$\sum_{i=1}^{N-1} u_i \left.\frac{\delta v}{\delta x}\right|_i = -\sum_{i=1}^{N-1} v_i \left.\frac{\delta u}{\delta x}\right|_i + \text{boundary terms.}$$

What are the boundary terms? Compare this discrete expression to the rule of integration by parts.

$$= \sum_{i=1}^{N-1} \left(u_i \left(\frac{v_{i+1} - v_{i-1}}{2h} \right) + v_i \left(\frac{u_{i+1} - u_{i-1}}{2h} \right) \right)$$

$$= \frac{1}{2h} \left(\sum_{i=1}^{N-1} u_i v_{i+1} - \sum_{i=0}^{N-2} u_i v_{i+1} + \sum_{i=1}^{N-1} u_{i+1} v_i - \sum_{i=0}^{N-2} u_{i+1} v_i \right.$$

$$= \frac{1}{2h} \left(u_{N-1} u_N - u_0 v_1 + u_N v_{N-1} - u_1 v_0 \right)$$

$$= \frac{1}{h} \left(\frac{u_{N-1} v_N - u_0 v_1}{2} + \frac{u_N v_{n-1} - u_1 v_0}{2} \right)$$

This is very similar to integration by parts because it requires doing the chain rule in reverse.

3.3

Friday, February 7, 2020 12:44 AM

3. Using the error analysis for the trapezoidal and rectangle rules, show that Simpson's rule for integration over the entire interval is fourth-order accurate.

We know that

$$\int_{x_i}^{x_{i+2}} f\,dx = 2h_i f_{i+1} + \frac{(2h_i)^3}{24} f''_{i+1} + \frac{(2h_i)^5}{1920} f^{(iv)}_{i+1} + \dots$$

$$\int_{x_i}^{x_{i+2}} f\,dx = h(f_i + f_{i+2}) - \frac{(2h_i)^3}{12} f''_{i+1} - \frac{(2h_i)^5}{480}$$

By plugging in 2h into 3.7 and 3.8

We know simpson can be formed from a linear combination of ten 2

$$\int_{x_i}^{x_{i+2}} f\,dx = \frac{h_1}{3}\left(f_i + 4f_{i+1} + f_{i+2}\right) - \frac{h_i^5}{90} f^{(iv)}_{i+1}$$

Summing over all and introducing the mean value theorem

$$I = \frac{\Delta}{3}\left(f_0 + f_n + 4\sum_{j=\text{odd}}^{n-1} f_j + 2\sum_{j=\text{Even}}^{n-2} f_j\right) - \boxed{\frac{\Delta^4}{180}}$$

4th order accurate

3.5

Friday, February 7, 2020 12:44 AM

5. Explain why the rectangle and trapezoidal rules can integrate a straight line exactly and the Simpson's rule can integrate a cubic exactly.

The trapezoidal Rule can do this because its error is proportional to f'', and because all lines have 0 as their second derivative the trapezoidal error falls out. In the same way Simpson's rule has leading error proportional to $f^{(iv)}$ and cubic functions have a 4th derivative of zero.

3.10

Friday, February 7, 2020 12:44 AM

8. Consider the integral

$$\int_0^1 \left[\frac{100}{\sqrt{x+.01}} + \frac{1}{(x-0.3)^2+.001} - \pi\right] dx.$$

(a) Numerically evaluate this integral using the trapezoidal rule with n panels of uniform length h. Make a log–log plot of the error (%) vs. n and discuss the accuracy of the method. Take $n = 8, 16, 32, \ldots$.
(b) Repeat part (a) using the Simpson's rule and the trapezoidal rule with end-correction.
(c) Evaluate the integral using an adaptive method with various error tolerances (you may want to use the *Numerical Recipes* subroutine `odeint` or MATLAB's function `quad8`). How are the x points for function evaluations distributed? Plot the integrand showing the positions of its evaluations on the x axis.

A) Integral:

$$10^{\frac{3}{2}}\left(\arctan\left(\sqrt{10}\,(10x-3)\right)+2\sqrt{10x+1}\right)-\pi x+C$$

derivative:

$$-\frac{50}{\left(x+\frac{1}{100}\right)^{\frac{3}{2}}}-\frac{2\left(x-\frac{3}{10}\right)}{\left(\left(x-\frac{3}{10}\right)^2+\frac{1}{1000}\right)^2}$$

B)

$$I = \frac{h}{2}\sum_{i=0}^{n-1}(f_i + f_{i+1}) - \frac{h^2}{12}\sum_{i=0}^{n-1}(f'_{i+1} - f'_i) + O(h^4).$$

Cancellations in the second summation on the right-hand side lead to

$$I = \frac{h}{2}\sum_{i=0}^{n-1}(f_i + f_{i+1}) - \frac{h^2}{12}(f'(b) - f'(a)) + O(h^4). \qquad (3.11)$$

Part C

Saturday, February 15, 2020 6:52 PM

$A = T(h)$

Rhomberg equations to solve 2^n for each increase

$$I_1 = \bar{I}_h = \boxed{T(h) \mid C_1 h^2 \mid C_2 h^4 \mid C_3 h^6 \mid C_4 h^8}$$

$$I_2 = \bar{I}_{(\frac{h}{2})} = \boxed{T(\tfrac{h}{2}) \mid C_1 \tfrac{h^2}{4} \mid C_2 h^4 \mid C_3 h^6 \mid C_4 h^8}$$

$$I = A + C_1 h^2$$

$$I = B + \frac{C_1}{4} h^2$$

$$I_1 - A = C_1 h^2$$

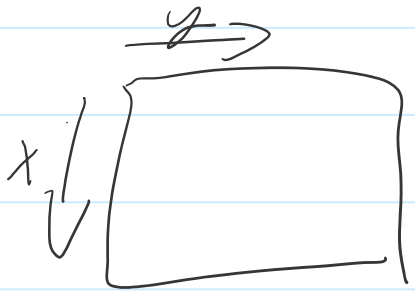
$$4(I_2 - B) = C_1 h^2$$

$$\begin{bmatrix} 1 & -h^2 \\ 1 & -\frac{1}{4}h^2 \end{bmatrix} \begin{bmatrix} I \\ C_2 \end{bmatrix} = \begin{bmatrix} A \\ B \end{bmatrix}$$

$$\begin{bmatrix} 1 & -h^2 \\ 0 & \frac{3}{4}h^2 \end{bmatrix} = \begin{bmatrix} -\frac{1}{3} & \frac{4}{3} \\ -1 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & -h^2 & -h^4 & -h^6 \\ 1 & -\frac{1}{4}h^2 & \frac{h^4}{2^4} & -\frac{h^6}{2^6} \\ 1 & -\frac{h^2}{4^2} & -\frac{h^4}{4^4} & -\frac{h^6}{4^6} \\ 1 & \frac{h^2}{4^2} & \frac{h^4}{8^4} & \frac{h^6}{8^6} \end{bmatrix} \begin{bmatrix} I \\ C_2 \\ C_3 \\ C_4 \end{bmatrix} = \begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix}$$

$h = 0$



While (O(h) > error)

Array = [[2^n], [2^n]]

ArrayB = [2^n, 1]

for x in (0, 2^n)

for y in (0, 2^n)

if y == 0

I = trap(Data, L, Step = $\frac{1}{2^x} \cdot h$)

Array[x, y] = 1

ArrayB[x, 1] = I

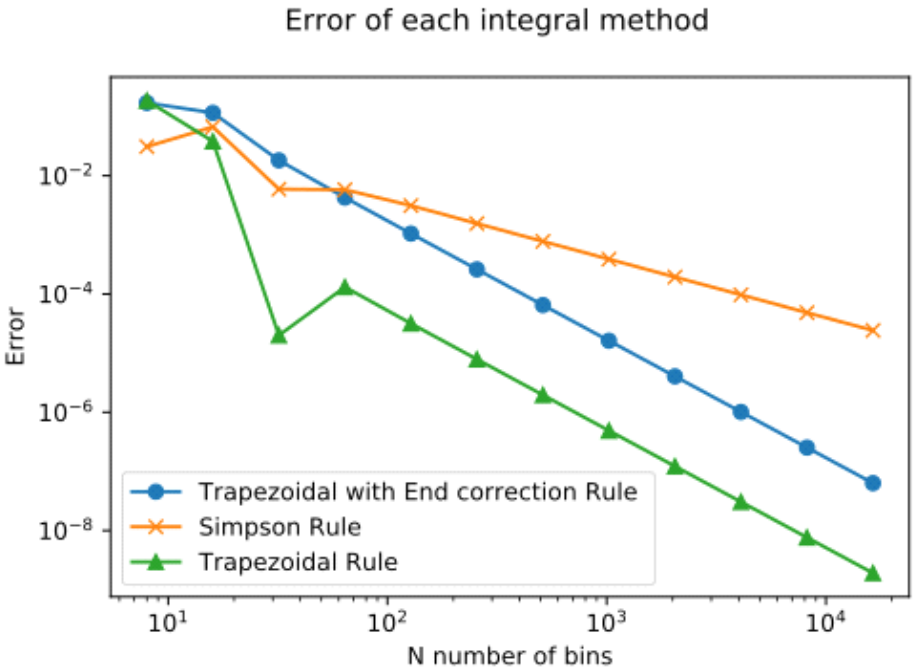
else

Array[x, y] = $-\left(\frac{h}{2^x}\right)^{2 \cdot y}$

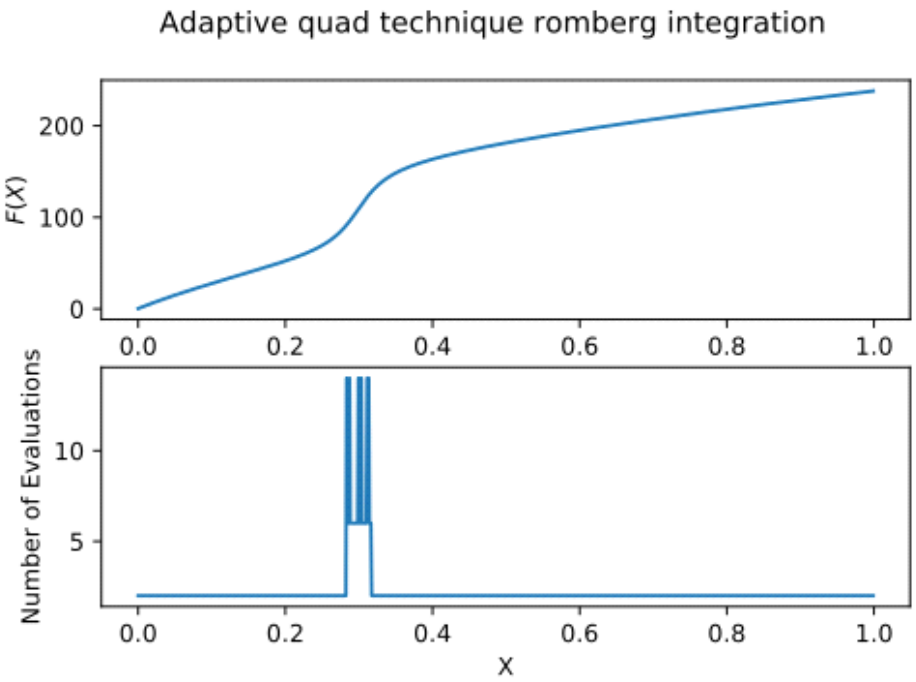
System = (A.copy()).col_insert[-1, ArrayB.copy()]

Plots

Monday, February 17, 2020 10:36 AM



I have no idea why the trapezoidal rule has such a high order. I spent several hours and went to several colleagues about the issue but I couldn't figure out the issue. My code is attached for review.



This made a great deal of sense that more evaluations were needed in places of sharper increase.

code Romberg integration and adaptive quadrature

Monday, February 17, 2020 10:37 AM

```
import numpy as np
import matplotlib.pyplot as plt
from sympy import *

def function(x):
   function=(100/((x+0.1)**(0.5)))+(1.0/(((x-0.3)**2)+0.001))-np.pi
   return function
def fderiv(x):
   a=50/((x+1/100.0)**(3.0/2))
   b=2*(x-0.3)
   c=((x-0.3)**2)+(1/1000)
   c=c**2
   f=-a-(b/c)
   return f


def rpart(y,delta):
   sumx=y*delta
   return sumx

def tpart(y1,y2,delta):
   sumx=((y1+y2)/2.0)*delta
   return sumx

def spart(y1,y2,y3,delta):
   sumx=(float(delta)/3.0)*(y1+(4.0*y2)+y3)
   return sumx

def rwhole(N,a,b,left):
   listx=np.linspace(a,b,N)
   listy=[]
   sumx=0
   if(left==True):
      for x in range(0,N):
```

```
        if(x<N-1):
            y1=function(listx[x])
            delta=listx[x]-listx[x+1]
            delta=(delta**2)**(0.5)
            sumx+= rpart(y1,delta)
            listy.append(sumx)
        else :
            return sumx
    else:
        for x in range(0,N):
        if(x<N-2):
            y2=function(listx[x+1])
            delta=listx[x]-listx[x+1]
            delta=(delta**2)**(0.5)
            sumx+= rpart(y2,delta)
            listy.append(sumx)
        else :
            return listx,listy,sumx
def twhole(N,a,b):
    listx=np.linspace(a,b,N)
    listy=[]
    sumx=0
    for x in range(0,N-1):
        y1=function(listx[x])
        y2=function(listx[x+1])
        delta=listx[x]-listx[x+1]
        delta=(delta**2)**(0.5)
        sumx+= tpart(y1,y2,delta)
        listy.append(sumx)
    return listx,listy,sumx
#this evaluates the trapezoidal rule with end corrections
def twhole2(N,a,b):
    listx=np.linspace(a,b,N)
    delta=listx[0]-listx[1]
    listy=[]
    sumx=0
```

```
    A=fderiv(a)
    B=fderiv(b)
    sumx-=((delta**2)/12.0)*(B-A)
    for x in range(0,N):
        if(x<N-1):
            y1=function(listx[x])
            y2=function(listx[x+1])
            delta=(delta**2)**(0.5)
            sumx+= tpart(y1,y2,delta)
            listy.append(sumx)
        else :
            return listx,listy,sumx


def swhole(N,a,b):
    listx=np.linspace(a,b,N)
    delta=listx[0]-listx[1]
    listy=[]
    sumx=0
    for x in range(1,int(N/2)):
        y1=function(listx[2*x-2])
        y2=function(listx[2*x-1])
        y3=function(listx[2*x])
        delta=(delta**2.0)**(0.5)
        sumx+= spart(y1,y2,y3,delta)
        listy.append(sumx)
    return listx,listy,sumx

def rhomberg(N,a,b,steps):
    L=2**N
    h=(float(b)-(a))/N
    A=zeros(2**N)
    B=zeros(2**N,1)
    print("We are currently at bin= "+ str(b))
    for x in range(0,L):
        for y in range(0,L):
```

```
            if y==0:
                A[x,y]=1.0
                n=(2**x)*steps
                print("row= "+str(x) +" has this many steps "+ str(n))
                B[x,0]=swhole(int(n),a,b)[2]

            else:
                A[x,y]=-(h/(2**x))**(2*y)
        System=(A.copy()).col_insert((L),B)
        x=symbols('a0:%d'%(L),Real=True)
        b=solve_linear_system(System,*x)

        evals=0
        i=N
        while i>0:
            evals= 2**(i)+evals
            i=i-1
        ans=[b[x[0]],b[x[L-1]],evals]

        return ans


def rhombergwhole(a,b,step,error):
    bins=np.linspace(a,b,step)
    x=[]
    y=[]
    evals=[]
    sumy=0;
    for i in range(0,len(bins)-1):
        exitwhile=False
        locala=bins[i]
        localb=bins[i+1]
        N=1
        while( not exitwhile):
            ans=rhomberg(N,locala,localb,step)
            #print("this is the N value "+str(N))
```

```
        #print("this is the error " + str(ans[1]))
        if(ans[1]<error):
            exitwhile=True
        if(N>2):
            exitwhile=True
        N=N+1
    evals.append(ans[2])
    sumy=sumy+float(ans[0])
    x.append(bins[i])
    y.append(sumy)
return x,y,evals
```