1. Order of accuracy

In [1]:
```
%matplotlib inline
%load_ext autoreload
```

Out[1]:
```
Using matplotlib backend: MacOSX
Populating the interactive namespace from numpy and matplotlib
```

Table 1

In [2]:
```
h=[1.0,0.5000000,0.2500000,0.1250000,0.0625000,0.0312500,0.0156250,0.0078125]
ef=[0.2419355,0.0967742,0.0322581,0.0107527,0.0035842,0.0011947,0.0003982,0.0001327]
ef[0]/ef[1]
ef[1]/ef[2]
ef[2]/ef[3]
ef[3]/ef[4]
ef[4]/ef[5]
ef[5]/ef[6]
ef[6]/ef[7]
ef[7]/ef[8]
print("{:8s}{:8s}{:8s}{:8s}{:8s}{:8s}{:8s}{:8s}".format("h","ef",
"ef[0]/ef[1]","ef[1]/ef[2]","ef[2]/ef[3]","ef[3]/ef[4]","ef[4]/ef[5]","ef[5]/ef[6]","ef[6]/ef[7]","ef[7]/ef[8]"))
for i in range(8):
    print("{:8.4f}".format(h[i]),
    "{:8.4f}".format(ef[i]),
    "{:8.4f}".format(ef[0]/ef[i]),
    "{:8.4f}".format(ef[i]/ef[i+1]),
    "{:8.4f}".format(ef[i+1]/ef[i+2]),
    "{:8.4f}".format(ef[i+2]/ef[i+3]),
    "{:8.4f}".format(ef[i+3]/ef[i+4]),
    "{:8.4f}".format(ef[i+4]/ef[i+5]),
    "{:8.4f}".format(ef[i+5]/ef[i+6]),
    "{:8.4f}".format(ef[i+6]/ef[i+7]))
```

Out[2]:
```
h       ef      ef[0]/ef[1]ef[1]/ef[2]ef[2]/ef[3]ef[3]/ef[4]ef[4]/ef[5]ef[5]/ef[6]ef[6]/ef[7]ef[7]/ef[8]
1.0000  0.2419  1.0000  2.5000  3.0000  3.0000  3.0000  3.0000  3.0000  3.0000
0.5000  0.0968  2.5000  3.0000  3.0000  3.0000  3.0000  3.0000  3.0000
0.2500  0.0323  3.0000  3.0000  3.0000  3.0000  3.0000  3.0000
0.1250  0.0108  3.0000  3.0000  3.0000  3.0000  3.0000
0.0625  0.0036  3.0000  3.0000  3.0000  3.0000
0.0313  0.0012  3.0000  3.0000  3.0000
0.0156  0.0004  3.0000  3.0000
0.0078  0.0001  3.0000
```

Table 2

In [3]:
```
h=[0.5000000,0.3000000,0.1500000,0.0750000,0.0375000,0.0187500,0.0093750,0.0046875]
ef=[0.0968,0.0387,0.0129,0.0043,0.0014,0.0005,0.0001,0.0000]
ef[0]/ef[1]
ef[1]/ef[2]
ef[2]/ef[3]
ef[3]/ef[4]
ef[4]/ef[5]
ef[5]/ef[6]
ef[6]/ef[7]
ef[7]/ef[8]
print("{:8s}{:8s}{:8s}{:8s}{:8s}{:8s}{:8s}{:8s}".format("h","ef",
"ef[0]/ef[1]","ef[1]/ef[2]","ef[2]/ef[3]","ef[3]/ef[4]","ef[4]/ef[5]","ef[5]/ef[6]","ef[6]/ef[7]","ef[7]/ef[8]"))
for i in range(8):
    print("{:8.4f}".format(h[i]),
    "{:8.4f}".format(ef[i]),
    "{:8.4f}".format(ef[0]/ef[i]),
    "{:8.4f}".format(ef[i]/ef[i+1]),
    "{:8.4f}".format(ef[i+1]/ef[i+2]),
    "{:8.4f}".format(ef[i+2]/ef[i+3]),
    "{:8.4f}".format(ef[i+3]/ef[i+4]),
    "{:8.4f}".format(ef[i+4]/ef[i+5]),
    "{:8.4f}".format(ef[i+5]/ef[i+6]),
    "{:8.4f}".format(ef[i+6]/ef[i+7]))
```

Out[3]:
```
h       ef      ef[0]/ef[1]ef[1]/ef[2]ef[2]/ef[3]ef[3]/ef[4]ef[4]/ef[5]ef[5]/ef[6]ef[6]/ef[7]ef[7]/ef[8]
0.5000  0.0968  1.0000  2.5000  3.0000  3.0000  3.0000  3.0000  3.0000  3.0000
0.3000  0.0387  2.5000  3.0000  3.0000  3.0000  3.0000  3.0000  3.0000
0.1500  0.0129  3.0000  3.0000  3.0000  3.0000  3.0000  3.0000
0.0750  0.0043  3.0000  3.0000  3.0000  3.0000  3.0000
0.0375  0.0014  3.0000  3.0000  3.0000  3.0000
0.0188  0.0005  3.0000  3.0000  3.0000
0.0094  0.0001  3.0000  3.0000
0.0047  0.0000  3.0000
```

2. Compact finite difference formulas

3.a)

In [4]:
```
# Error derivative
def fiveDerivative(x,h,a,b,c,d):
    '''
    Arguments
    ---------
    x: Numpy array
    f: function to be differentiated
    ab,cd: Boundary values of the function at a and b respectively
    h: Step size for the uniform grid.
    Return
    ------
    D:jacobian the array of derivatives, containing values of derivatives at the interior points
    '''
    N=len(x)-1
    D=sparse.lil_matrix((N-1,N-1))
    A=sparse.lil_matrix((N-1,N-1))
    D[0,0]=-5/2
    D[0,1]=2
    D[0,2]=1/2
    A[0,0]=1
    for i in range(1):
    D[i,i-1]=2
    D[i,i]=-5/2
    D[i,i+1]=4/3
    D[i,i+2]=-1/12
    A[i,i-1]=1/6
    A[i,i]=2/3
    A[i,i+1]=1/6
    for i in range(1,N-2):
    D[i,i-1]=4/3
    D[i,i]=-5/2
    D[i,i+1]=4/3
    D[i,i+2]=-1/12
    A[i,i-1]=1/6
    A[i,i]=2/3
    A[i,i+1]=1/6
    D=D/(h**2)
    A=A/(h**2)
    return D,A
```

In [5]:
```
# second derivative
def secondDerivative(x,h,a,b,c,d):
    '''
    Arguments
    ---------
    x: Numpy array
    f: function to be differentiated
    ab,cd: Boundary values of the function at a and b respectively
    h: Step size for the uniform grid.
    Return
    ------
    D:jacobian the array of derivatives, containing values of derivatives at the interior points
    '''
    N=len(x)-1
    D=sparse.lil_matrix((N-1,N-1))
    A=sparse.lil_matrix((N-1,N-1))
    D[0,0]=-5/2
    D[0,1]=2
    D[0,2]=1/2
    A[0,0]=1
    for i in range(1):
    D[i,i-1]=2
    D[i,i]=-5/2
    D[i,i+1]=4/3
    D[i,i+2]=-1/12
    A[i,i-1]=1/6
    A[i,i]=2/3
    A[i,i+1]=1/6
    for i in range(1,N-2):
    D[i,i-1]=4/3
    D[i,i]=-5/2
    D[i,i+1]=4/3
    D[i,i+2]=-1/12
    A[i,i-1]=1/6
    A[i,i]=2/3
    A[i,i+1]=1/6
    D=D/(h**2)
    A=A/(h**2)
    return D,A
```

3.b)

In [6]:
```
# exact derivative
def exactDerivative(x):
    '''
    Arguments: Error derivative
    ---------
    return:(x)*(x**4-x)
    '''
    return(x)*(x**4-x)
def exactSecond(x):
    '''
    Arguments: second derivative
    ---------
    return:(x)*(x**3)*(x**4-x)
    '''
    return(x)*(x**3)*(x**4-x)
def exactFive(x):
    '''
    Arguments:
    ---------
    x: Is a vector containing approximated values
    f: Contains exact values of the derivative.
    Return
    ------
    y:f at some of the error
    '''
    return(-5*x**4+4*x**3)
def exactFive(x):
    '''
    Arguments: Five derivative
    ---------
    return:(x)*(x**4-x)
    '''
    return(x)*(x**4-x)
def exactSecond(x):
    '''
    Arguments: second derivative
    ---------
    return:(x)*(x**3)*(x**4-x)
    '''
    return(x)*(x**3)*(x**4-x)
def exactFive(x):
    '''
    Arguments:
    ---------
    x: Is a vector containing approximated values
    f: Contains exact values of the derivative.
    Return
    ------
    y:f at some of the error
    '''
    return(-5*x**4+4*x**3)
```

In [7]:
```
# Initialization
x=np.linspace(0,1,N+1)[1:-1]
h=x[1]-x[0]
ab=exact(x[0],h)
cd=exact(x[-1],h)
x,A=fiveDerivative(x,h,ab,cd)
```

In [8]:
```
# Initialization
x=np.linspace(0,1,N+1)[1:-1]
h=x[1]-x[0]
ab=exactSecond(x[0],h)
cd=exactSecond(x[-1],h)
x,A=secondDerivative(x,h,ab,cd)
```

In [9]:
```
# Initialization
x=np.linspace(0,1,N+1)[1:-1]
h=x[1]-x[0]
ab=exactFive(x[0],h)
cd=exactFive(x[-1],h)
x,A=fiveDerivative(x,h,ab,cd)
```

In [10]:
```
# Solution 5-point
u=np.linspace(0,1,N+1)[1:-1]
u=exactFive(u)
u=exact(u)
u=u[:-1]
u=u[1:]
b=A*u
f=np.linspace(0,1,N+1)
f=exactFive(f)
f=f[1:-1]
print("Exact first derivative of u(x)",u,f,b)
print("Error",f-b)
print("Maximum error",np.max(np.abs(f-b)))
```

Out[10]:
```
Exact first derivative of u(x) [0.2083333 0.3437333 0.4218333 0.4166250 0.4166250 0.4166250 0.4166250
 0.4166250]
Error [0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000]
Maximum error 0.0000000
```

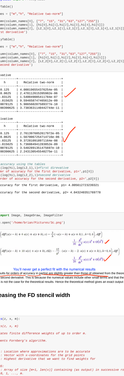
In [11]:
```
# Solution 5-point
u=np.linspace(0,1,N+1)[1:-1]
u=exactSecond(u)
u=u[:-1]
u=u[1:]
b=A*u
f=np.linspace(0,1,N+1)
f=exactSecond(f)
f=f[1:-1]
print("Exact second derivative of u(x)",f,f,f)
print("Error",f-b)
print("Maximum error",np.max(np.abs(f-b)))
```

Out[11]:
```
Exact second derivative of u(x) [0.2083333 0.3437333 0.4218333 0.4166250 0.4166250 0.4166250 0.4166250
 0.4166250]
Error [0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000]
Maximum error 0.0000000
```

In [12]:
```
# Solution 5-point
u=np.linspace(0,1,N+1)[1:-1]
u=exactFive(u)
u=u[:-1]
u=u[1:]
b=A*u
f=np.linspace(0,1,N+1)
f=exactFive(f)
f=f[1:-1]
print("Exact five derivative of u(x)",f,f,f)
print("Error",f-b)
print("Maximum error",np.max(np.abs(f-b)))
```

Out[12]:
```
Exact five derivative of u(x) [0.2083333 0.3437333 0.4218333 0.4166250 0.4166250 0.4166250 0.4166250
 0.4166250]
Error [0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000]
Maximum error 0.0000000
```

In [13]:
```
# Error plot
def plot(x,f,b):
    '''
    Arguments: Error derivative
    ---------
    x: Numpy array
    f: Exact first derivative
    b: Numpy array
    '''
    plt.loglog(x,f-b,'.')
    plt.xlabel('$x$')
    plt.ylabel('Error derivative')
    plt.title('Relative error')
    plt.legend('$f-b$')
    plt.show()
plot(x,f,b)
```



3.c)

In [14]:
```
# Time
%time ef=[exact(five,h),exact(five,h),exact(five,h)]
ef
```

Out[14]:
```
CPU times: user 2.4 µs, sys: 1.1 µs, total: 3.5 µs
Wall time: 6.2 µs
[0.0, 0.0, 0.0]
```

$$\text{CPU time}[\mu s]=\frac{(0.0+0.0+0.0)\times 10^{-6}}{3}=0.0\times 10^{-6}[s]$$

$$\text{Wall time}[\mu s]=\frac{(6.2)\times 10^{-6}}{3}=2.0\times 10^{-6}[s]$$

You'll never get a perfect 0 with the numerical results

Numerical results by using the exact derivative values, the error is exactly zero. Differentiation from the theoretical results which is pi for the first derivative and 2pi for the second derivative. Because we are using exact values of the derivative and the step size of h depends on x,h and ab, which is not the case for the theoretical results. Hence the theoretical method gives an exact output of 0.

4. Increasing the FD stencil width

(a)

In [15]:
```
def weights(x, n, m):
    '''
    weights(x, n, m)

    Calculates finite difference weights of up to order m.

    Implements Fornberg's algorithm.

    ARGS:
    - l location where approximations are to be accurate
    - n grid locations, stored in array x, for stencil
    - m highest derivative that we want to find weights for

    RETURNS:
    - c array of size (n,m), indexed (stencil location, order) in successive rows the weights for the
    derivatives of 0, 1, ..., m

    EXAMPLE:
    - to calculate the 2nd order centered FD formula for the zeroth, first and second derivatives, x
    would be following call to weights:
        - c = weights(0, x[:3], 2)[:,:3]

    ''' Translated by Hans Fehr from the original source by Fornberg
    c = np.zeros(n-1,m)
    c1 = 1
    c4 = x[0] - x[l]
    for i in range(n):
    mn = min(i,m)
    c2 = 1
    for j in range(-1):
    c3 = x[i] - x[j]
    c2 = c2 * c3
    if j == i-1:
    c5 = c4
    for k in range(mn):
    c[i,k] = (c1 * (k * c[j,k-1] - c5 * c[j,k]) / c2
    c[j,k] = (c4 * c[j,k] - k * c[j,k-1]) / c3
    c1 = c2
    return c
```

Equispaced weights

In [16]:
```
N=100
h=linspace(0,1,N)
f=exp(sin(2*h))

for i in range(N):
    c=weights(h[i],h,1)
    ef[i]=abs(f[i]-c[0,1]*f[i])
    eb[i]=abs(f[i]-c[0,1]*f[i])
    ec[i]=abs(f[i]-c[1,1]*f[i])
print("Equispaced,ef[-1]")
plot(h,ef)
print("Equispaced,eb[-1]")
plot(h,eb)
print("Equispaced,ec[-1]")
plot(h,ec)
```

Out[16]:
```
Equispaced,ef[-1] 2.5127527e-04 4.7261270e-03 2.7123123e-02 1.0100001e-01
Equispaced,eb[-1] 2.5127527e-04 4.7261270e-03 2.7123123e-02 1.0100001e-01
Equispaced,ec[-1] 2.5127527e-04 4.7261270e-03 2.7123123e-02 1.0100001e-01
Equispaced,ef[-2] 2.5127527e-04 4.7261270e-03 2.7123123e-02 1.0100001e-01
Equispaced,eb[-2] 2.5127527e-04 4.7261270e-03 2.7123123e-02 1.0100001e-01
Equispaced,ec[-2] 2.5127527e-04 4.7261270e-03 2.7123123e-02 1.0100001e-01
Equispaced,ef[-3] 2.5127527e-04 4.7261270e-03 2.7123123e-02 1.0100001e-01
Equispaced,eb[-3] 2.5127527e-04 4.7261270e-03 2.7123123e-02 1.0100001e-01
Equispaced,ec[-3] 2.5127527e-04 4.7261270e-03 2.7123123e-02 1.0100001e-01
Equispaced,ef[-4] 2.5127527e-04 4.7261270e-03 2.7123123e-02 1.0100001e-01
Equispaced,eb[-4] 2.5127527e-04 4.7261270e-03 2.7123123e-02 1.0100001e-01
Equispaced,ec[-4] 2.5127527e-04 4.7261270e-03 2.7123123e-02 1.0100001e-01
```

Chebyshev weights

In [17]:
```
N=100
h=linspace(0,1,N)
f=exp(sin(2*h))

for i in range(N):
    c=weights(h[i],h,1)
    ef[i]=abs(f[i]-c[0,1]*f[i])
    eb[i]=abs(f[i]-c[0,1]*f[i])
    ec[i]=abs(f[i]-c[1,1]*f[i])
print("Chebyshev,ef[-1]")
plot(h,ef)
print("Chebyshev,eb[-1]")
plot(h,eb)
print("Chebyshev,ec[-1]")
plot(h,ec)
Chebyshev,ef[-2] 0.0000000e+00 1.6257148e-02 1.2000000e-01 1.2700000e-00
Chebyshev,eb[-2] 0.0000000e+00 1.6257148e-02 1.2000000e-01 1.2700000e-00
Chebyshev,ec[-2] 0.0000000e+00 1.6257148e-02 1.2000000e-01 1.2700000e-00
Chebyshev,ef[-3] 1.6257148e-02 1.2000000e-01 1.2700000e-00
Chebyshev,eb[-3] 1.6257148e-02 1.2000000e-01 1.2700000e-00
Chebyshev,ec[-3] 1.6257148e-02 1.2000000e-01 1.2700000e-00
Chebyshev,ef[-4] 1.2000000e-01 1.2700000e-00
Chebyshev,eb[-4] 1.2000000e-01 1.2700000e-00
Chebyshev,ec[-4] 1.2000000e-01 1.2700000e-00
```

Legendre weights

In [18]:
```
[-1.0,-0.9999999999999999,-0.9999999999999997,-0.9999999999999991,-0.9999999999999974,-0.9999999999999926,-0.9999999999999786,-0.9999999999999388,-0.9999999999998252,-0.9999999999995000,-0.9999999999985714,-0.9999999999959184,-0.9999999999883382,-0.9999999999666667,-0.9999999999047619,-0.9999999997278912,-0.9999999991666667,-0.9999999976190476,-0.9999999933333333,-0.9999999809523810,-0.9999999457142857,-0.9999999428571429,-0.9999999142857143,-0.9999999142857143,-0.9999999142857143
```

5. (Trigonometric interpolation)

9/10

$x_j = \frac{2\pi j}{N}$, $j = 0, 1, \dots, N-1$

Given

$$P_N(x) = \sum_{j=0}^{N-1} S_N(x - x_j)\, u(x_j)$$

where

$$S_N(t) = \begin{cases} \frac{1}{N} \sin\left(\frac{N}{2} t\right) \cot\left(\frac{t}{2}\right), & \text{if } N \text{ is even,} \\ \frac{1}{N} \sin\left(\frac{N}{2} t\right) \csc\left(\frac{t}{2}\right), & \text{if } N \text{ is odd.} \end{cases}$$

a) Verify that $P_N(x)$ actually interpolates the data.

Show that $P_N(x_k) = u(x_k)$ for $k = 0, 1, 2, \dots, N-1$

We have to show that

$$S_N(x_j - x_k) = 0 \text{ when } k \neq j \text{ and } S_N(0) = 1$$

$$S_N(x_j - x_k) = \begin{cases} \frac{1}{N} \sin\left(\frac{N}{2}(x_j - x_k)\right) \cot\left(\frac{x_j - x_k}{2}\right), & \text{if } N \text{ is even} \\ \frac{1}{N} \sin\left(\frac{N}{2}(x_j - x_k)\right) \csc\left(\frac{x_j - x_k}{2}\right); & \text{if } N \text{ is odd.} \end{cases}$$

but $x_j = \frac{2\pi j}{N}$, $x_k = \frac{2\pi k}{N}$

$$S_N(x_j - x_k) = \begin{cases} \frac{1}{N} \sin\left(\pi (j-k)\right) \cot\left(\frac{\pi}{N}(j-k)\right), & \text{if } N \text{ is even,} \\ \frac{1}{N} \sin\left(\pi (j-k)\right) \csc\left(\frac{\pi}{N}(j-k)\right), & \text{if } N \text{ is odd.} \end{cases}$$

When $k \neq j$ and if N is even:

$$S_N(x_j - x_k) = \frac{1}{N} \sin(\pi(j-k)) \cdot \csc\left(\frac{\pi}{N}(j-k)\right)$$

and $N \neq 0$,

Since j and k are integers, then $\sin(\pi(j-k)) = 0$

hence

$$S_N(x_j - x_k) = 0$$

When $k \neq j$, if N is odd

$$S_N(x_j - x_k) = \frac{1}{N} \sin(\pi(j-k)) \csc\left(\frac{\pi}{N}(j-k)\right)$$

Since j and k are integers, then $\sin(\pi(j-k)) = 0$

hence

$$S_N(x_j - x_k) = 0$$

therefore

$$S_N(x_j - x_k) = \begin{cases} 0, & \text{if } N \text{ is even} \\ 0, & \text{if } N \text{ is odd.} \end{cases}$$

Also show that $S_N(0) = 1$

If N is even

$$S_N(t) = \frac{1}{N} \sin\left(\frac{N}{2}t\right) \csc(t/2) = \frac{1}{N} \sin\left(\frac{N}{2}t\right) \frac{\cos(t/2)}{\sin(t/2)}$$

$$\lim_{t \to 0} S_N(t) = \frac{0}{0}$$

Using Lahopital's rule.

$$\lim_{t\to 0} S_N(t) = \frac{\frac{1}{N}\left(\frac{N}{2}\cos\left(\frac{N}{2}t\right)\cos(t/2) - \frac{1}{2}\sin(t/2)\sin\left(\frac{N}{2}t\right)\right)}{\frac{1}{2}\cos(t/2)}$$

As $t \to 0$ $\sin t/2 \sin(N/2\, t) \to 0$

$$\lim_{t\to 0} S_N(t) = \frac{2}{N}\cdot\frac{N}{2}\cos\left(\frac{N}{2}(0)\right) = \underline{\underline{1}}$$

If N is odd

$$S_N(t) = \frac{1}{N}\sin\left(\frac{N}{2}t\right)\csc(t/2) = \frac{\frac{1}{N}\sin\left(\frac{N}{2}t\right)}{\sin(t/2)}$$

$$\lim_{t\to 0} S_N(t) = \frac{0}{0}$$

Using Lahopitals Rule.

$$\lim_{t\to 0} S_N(t) = \frac{\frac{1}{N}\left(\frac{N}{2}\cos\left(\frac{N}{2}t\right)\right)}{\frac{1}{2}\cos(t/2)} = \frac{2}{N}\cdot\frac{N}{2} = 1.$$

$$\lim_{t\to 0} S_N(t) = \begin{cases} 1 & ; \text{If } N \text{ is even} \\ 1 & , \text{If } N \text{ is odd.} \end{cases}$$

Hence $P_N(x)$ actually interpolates the data.

5b) For the case of N being even show that

$$S_N'(x_k) = \begin{cases} 0, & \text{if } k=0 \\ \frac{1}{2}(-1)^k \cot\left(\frac{k\pi}{N}\right), & \text{if } k \neq 0 \end{cases}$$

~~to~~ from

$$S_N(t) = \begin{cases} \frac{1}{N}\sin\left(\frac{N}{2}t\right)\cot\left(t/2\right), & \text{if } N \text{ is even,} \\ \frac{1}{N}\sin\left(\frac{N}{2}t\right)\csc\left(t/2\right), & \text{if } N \text{ is odd.} \end{cases}$$

If N is even

$$S_N(t) = \frac{1}{N}\sin\left(\frac{N}{2}t\right)\cot\left(t/2\right)$$

taking $t = x_k$

$$S_N(x_k) = \frac{1}{N}\sin\left(\frac{N}{2}x_k\right)\cot\left(x_k/2\right)$$

$$S_N'(x_k) = \frac{d}{dx_k}S_N(x_k) = \frac{1}{N}\frac{d}{dx_k}\left[\sin\left(\frac{N}{2}x_k\right)\cot\left(\frac{x_k}{2}\right)\right]$$

$$= \frac{1}{N}\left[\cot\left(x_k/2\right)\left(\frac{N}{2}\cos\left(\frac{N}{2}x_k\right)\right) + \sin\left(\frac{N}{2}x_k\right)\left(-\frac{1}{2}\csc^2\left(x_k/2\right)\right)\right]$$

but $x_k = \frac{2\pi}{N}k$

$$S_N'(x_k) = \frac{1}{N}\left[\frac{N}{2}\cot\left(\frac{\pi}{N}k\right)\cos(\pi k) - \frac{1}{2}\sin(\pi k)\csc^2\left(\frac{\pi}{N}k\right)\right]$$

Since k is an integer $\sin(\pi k) = 0$

$$S_N'(x_k) = \frac{1}{N}\left[\frac{N}{2}\cot\left(\frac{\pi}{N}k\right)\cos(\pi k)\right]$$

$$S_N'(x_k) = \frac{1}{2}\cot\left(\frac{\pi}{N}k\right)\cos(\pi k)$$

If $k=0$

$$S_N'(x_k) = \frac{1}{2}\cos(\pi k)\,\frac{\cos\left(\frac{\pi}{N}k\right)}{\sin\left(\frac{\pi}{N}k\right)} = \frac{1}{0}$$ which is not good

Using Lahopital's Rule

You can only use L'hopital's rule when you have 0/0. 1/0 is undefined.

$$S_N'(x_k) = \frac{\frac{1}{2}\left[-\frac{\pi}{N}\sin\left(\frac{\pi}{N}k\right)\cos(\pi k) + -\pi\sin(\pi k)\cos\left(\frac{\pi}{N}k\right)\right]}{\frac{\pi}{N}\cos\left(\frac{\pi}{N}k\right)}$$

$$\lim_{k\to 0} S_N'(x_k) = \frac{0}{1} = \underline{\underline{0}}$$

To take the limit properly, you have to start with the S_N'(t) and take t->0.

If $k \neq 0$

$$S_N'(x_k) = \frac{1}{2}\cos(\pi k)\cot\left(\frac{\pi}{N}k\right).$$

$$\cos(\pi k) \simeq (-1)^k \quad \text{for } k \text{ integer.}$$

$$S_N'(x_k) = \frac{1}{2}(-1)^k \cot\left(\frac{\pi}{N}k\right)$$

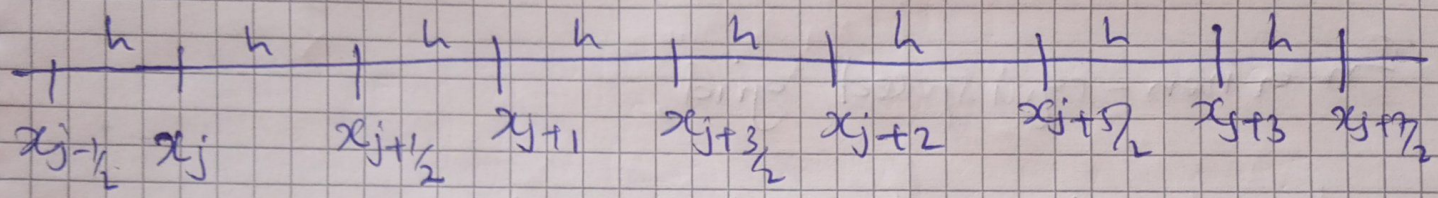
Therefore

$$S_N'(x_k) = \begin{cases} 0, & \text{if } k=0 \\ \frac{1}{2}(-1)^k \cot\left(\frac{\pi}{N}k\right), & \text{if } k\neq 0 \end{cases}$$

No 2 (Finite - volume scheme)

a) For a uniform grid, from equation (1) in the ~~x~~ question.

$$x_{j+1} - x_j = x_{j+3/2} - x_{j+1/2} = x_{j+1/2} - x_{j-1/2} = h$$

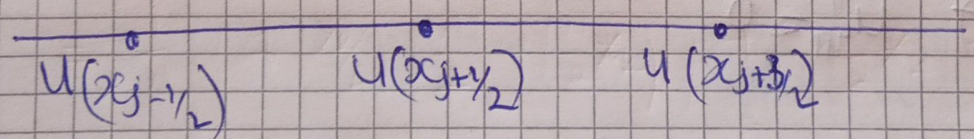


therefore

$$u''(x_{j+1/2}) \approx \frac{1}{h^2}\left[u(x_{j+3/2}) - u(x_{j+1/2}) - u(x_{j+1/2}) + u(x_{j-1/2})\right]$$

$$u''(x_{j+1/2}) \approx \frac{1}{h^2}\left[u(x_{j+3/2}) - 2u(x_{j+1/2}) + u(x_{j-1/2})\right]$$

$$h^2 u''(x_{j+1/2}) \approx u(x_{j+3/2}) - 2u(x_{j+1/2}) + u(x_{j-1/2}) \quad \text{--- (1)}$$



Hence a 3 point scheme.

In close, ~~we~~ for $u''(x_j)$, we obtained

$$h^2 u''(x_j) \approx u(\bar{x}-h) - 2u(\bar{x}) + u(\bar{x}+h) \quad \text{--- (2)}$$

which is also a 3 point scheme.

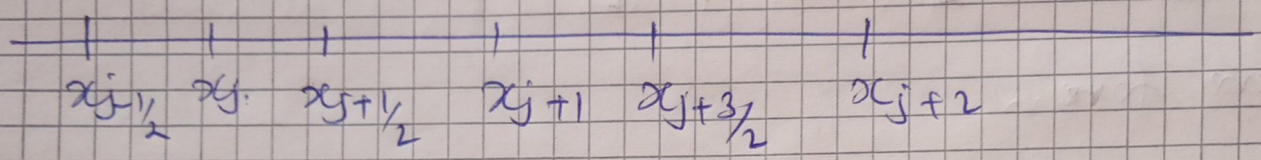
in equation (1), taking $\bar{x} = x_{j+1/2}$, we obtain

$h^2 u''(\bar{x}) \approx u(\bar{x}-h) - 2u(\bar{x}) + u(\bar{x}+h)$, which is

Similar to equation ②, hence the same stencil.

b) Show that on a non-equispaced grid, ~~Show~~ (1) is exact for $u=1$ and $u=x$, but not $u=x^2$

For a non-equispaced grid,



$$x_{j+1/2} - x_{j-1/2} \neq x_{j+3/2} - x_{j+1/2} \neq x_j - x_{j+1} \neq h.$$

For $u=1$

$$\frac{d}{dx} u'_{exact} = 0 \quad , \quad u''_{exact} = 0$$

into (1), if $u=1$ then $u(x_{j+3/2}) - u(x_{j+1/2}) = 0$ and also $u(x_{j+1/2}) - u(x_{j-1/2}) = 0$, ~~therefore~~

~~$u''(x_{j+1/2})$~~ $u''_{app} = 0$

Since

$u''_{exact} = u''_{approx}$, hence ① is exact for $u=1$

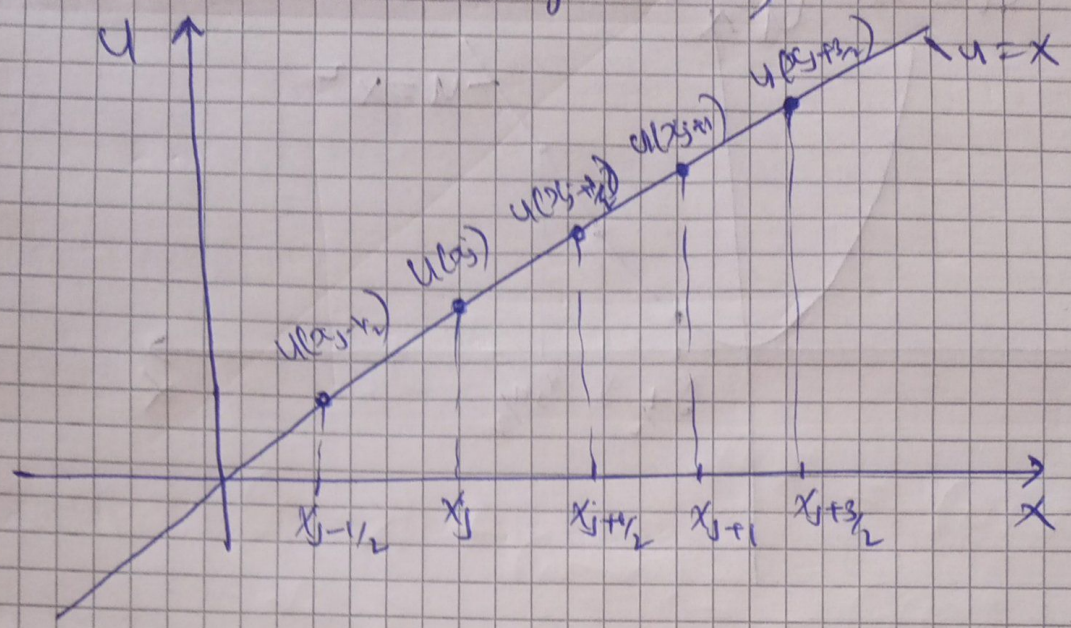
For $u = x$

<u>Exact</u>

$u'_{exact} = 1$, $u''_{exact} = 0$

<u>Numerical</u>

Since $u = x$, is a straight line,



then

$$\frac{u(x_{j+3/2}) - u(x_{j+1/2})}{x_{j+3/2} - x_{j+1/2}} = \frac{u(x_{j+1/2}) - u(x_{j-1/2})}{x_{j+1/2} - x_{j-1/2}}$$

therefore

$$u''_{approx} = \frac{1}{x_{j+1} - x_j}(0) = 0$$

Since $u''_{exact} = u''_{approx} = 0$ Hence, (1) is exact for $u = x$

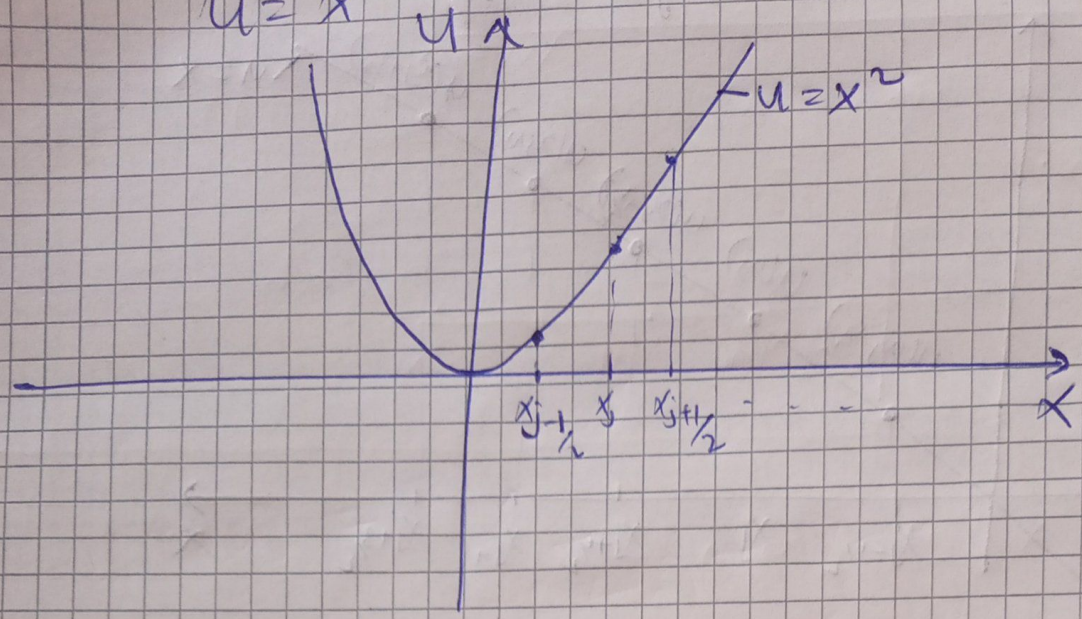
For $u = x^2$

$u'_{exact} = 2x$, $u''_{exact} = 2$

<u>Numerically</u>

Since $u = x^2$ is a curve, and that grid space points are not equal spaced,

$u = x^2$



Then,

$$\frac{u(x_{j+3/2}) - u(x_{j+1/2})}{x_{j+3/2} - x_{j+1/2}} \neq \frac{u(x_{j+1/2}) - u(x_{j-1/2})}{x_{j+1/2} - x_{j-1/2}}$$

therefore, $u''(x_{j+1/2}) \approx \frac{1}{x_{j+1} - x_j}\left[\frac{(x_{j+3/2})^2 - (x_{j+1/2})^2}{x_{j+3/2} - x_{j+1/2}} - \frac{(x_{j+1/2})^2 - (x_{j-1/2})^2}{x_{j+1/2} - x_{j-1/2}}\right]$

$u''(x_{j+1/2}) \neq 2$, Cannot equal to 2,

$u''_{approx} \neq u''_{exact}$, Therefore the technique is not Ex...

You have to use non-equally spaced points here to and generate the weights using the Lagrange interpolation technique from class. Or you can just use the weights function. Alternatively, you can argue why that method has to be exact for 1,x,x^2 regardless of how the points are distributed (as long as they are distinct).

For the Technique we discussed in Class, the grid points are equally spaced, which means that

$x_{j+1} - x_j = x_{j+3/2} - x_{j+1/2} = x_{j+1/2} - x_{j-1/2} = h$

$$u''(x_{j+1/2}) \approx \frac{1}{h^2}\left[u(x_{j+3/2}) - 2u(x_{j+1/2}) + u(x_{j-1/2})\right]$$

therefore $u = 1$, $\bar{x} = x_{j+1/2}$

$u(x_{j+3/2}) = u(x_{j+1/2}) = u(x_{j-1/2}) = 1$, since its a straight line

$u(\bar{x}+h) = u(\bar{x}) = u(\bar{x}-h) = 1$

$$u''(\bar{x}) \approx \frac{1}{h^2}\left[u(\bar{x}+h) - 2u(\bar{x}) + u(\bar{x}-h)\right] = \frac{1}{h^2}(2-2)$$

$$u''(\bar{x}) \approx 0$$

which is an <u>exact value</u>.

For $u = x$

$u(\bar{x}+h) = \bar{x}+h$, $u(\bar{x}) = \bar{x}$, $u(\bar{x}-h) = \bar{x}-h$.

$$u''(\bar{x}) \approx \frac{1}{h^2}\left((\bar{x}+h) - 2\bar{x} + (\bar{x}-h)\right) = \frac{1}{h^2}(0) = 0$$

$u''(\bar{x}) = 0$, which is an <u>exact value</u>

For $u = x^2$

$u(\bar{x}+h) = (\bar{x}+h)^2 = \bar{x}^2 + h^2 + 2\bar{x}h$

$u(\bar{x}) = \bar{x}^2$

$u(\bar{x}-h) = (\bar{x}-h)^2 = \bar{x}^2 + h^2 - 2\bar{x}h$

$$u''(\bar{x}) \approx \frac{1}{h^2}\left(\cancel{\bar{x}^2} + \cancel{h^2} + \cancel{2\bar{x}h} - 2\bar{x}^2 + \cancel{\bar{x}^2} + \cancel{h^2} - \cancel{2\bar{x}h}\right) = \frac{1}{h^2}(2h^2)$$

$$u''(\bar{x}) \approx 2$$

which is an exact value.