Numerical Approximation of a Derivative Using the Method of Central Difference

November 16, 2018

Author:
Jonathan Duarte

Abstract

The purpose of this **FORTRAN** program was to perform a numerical approximation of a derivative using the central difference method. A smooth function $y = \sin(10x)$, where $x \in [-1, 1]$, was used to obtain the end result of the estimate as well as its numerical stability. In order to obtain the central difference of a smooth function, a Taylor Series expansion was taken around some neighborhood of a point x. After obtaining a finite approximation, it was possible to estimate the derivative at $x = x_i$ by using values from the left side of y as well as to the right. Since the end points are not valid around the neighborhood of x, a special one-sided formulæwas constructed. It can be shown that at $N = 6$, the special one-sided formulæcan be written in matrix form, namely **D**. With **y** being the independent variable of the function and **D** being a matrix, it was possible to estimate the derivatives y' by a matrix-vector multiplication. The end result illustrates that by taking the second derivative of y, the function's discrete points collapsed more accurately than the first derivative. An error analysis was then conducted for $y' = 10\cos(10x)$ at $y'(0) = 10$ using a log-log plot which resulted in the decrease in truncation error as the magnitude increased.

Numerical Analysis

As described in the **Abstract**, a smooth function was used for the method of central difference, namely

$$y = \sin(10x), \quad x \in [-1, 1] \tag{1}$$

In order to arrive at that conclusion it is necessary to derive the central difference method. In a general smooth function, the central difference method is first obtained by a Taylor Series expansion taken around some neighborhood of a point x. The neighboring x values that surround the point are:

$$..., x - 3h, x - 2h, x - h, x, x + h, x + 2h, x + 3h, ...$$

Using Taylor's theorem, we have:

$$y(x + h) = y(x) + y'(x)h + \frac{1}{2}y''(\lambda_2)h^2$$

By canceling out the higher order terms and reworking the equation, we have:

$$y'(x) \approx \frac{y(x + h) - y(x)}{h}$$

which is the forward difference approximation. If x is the central point of the approximation, it can be shown that:

$$y'(x) \approx \frac{y(x + h) - y(x - h)}{2h}$$

is the central difference formula approximation. To simplify the previous approximation, let's suppose we want a precise estimate such that $x + h$ is the iteration of y_{i+1}, $x - h$ is the iteration

of y_{i-1} and $y'(x) = y'_i$. Lets also suppose that, $h = \Delta x$ where $\Delta x = \frac{b-a}{N}$, $x \in [a, b]$, and N is the number of spaces uniformly distributed around point $x = x_i$. Our new approximation becomes:

$$y'_i \approx \frac{y_{i+1} - yi - 1}{2\Delta x}, \quad i = 2, ..., N$$

The estimate y'_i does not become valid when the end points, y'_i and y'_{N+1}, because there are no values for y_0 or y_{N+2}. Thus a special one-sided formulæwas constructed for the end points, namely:

$$y'_1 = \frac{-3y_1 + 4y_2 - y_3}{2\Delta x}, \tag{2}$$

$$y'_{N+1} = \frac{y_{N-1} - 4y_N + 3y_{N+1}}{2\Delta x} \tag{3}$$

All of the y'_i can be written in matrix form with $N = 6$. To illustrate we now have:

$$\underbrace{\begin{bmatrix} y'_1 \\ y'_2 \\ y'_3 \\ y'_4 \\ y'_5 \\ y'_6 \\ y'_7 \end{bmatrix}}_{\mathbf{y'}} = \underbrace{\begin{bmatrix} -\frac{3}{2\Delta x} & \frac{4}{2\Delta x} & -\frac{1}{2\Delta x} & 0 & 0 & 0 & 0 \\ -\frac{1}{2\Delta x} & 0 & \frac{1}{2\Delta x} & 0 & 0 & 0 & 0 \\ 0 & -\frac{1}{2\Delta x} & 0 & \frac{1}{2\Delta x} & 0 & 0 & 0 \\ 0 & 0 & -\frac{1}{2\Delta x} & 0 & \frac{1}{2\Delta x} & 0 & 0 \\ 0 & 0 & 0 & -\frac{1}{2\Delta x} & 0 & \frac{1}{2\Delta x} & 0 \\ 0 & 0 & 0 & 0 & -\frac{1}{2\Delta x} & 0 & \frac{1}{2\Delta x} \\ 0 & 0 & 0 & 0 & \frac{1}{2\Delta x} & -\frac{4}{2\Delta x} & \frac{3}{2\Delta x} \end{bmatrix}}_{\mathbf{D}} \underbrace{\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix}}_{\mathbf{y}}$$

This is of the form:

$$\mathbf{y'} = \mathbf{D} \cdot \mathbf{y} \tag{4}$$

D can be reduced into:

$$\mathbf{D} = \frac{1}{2\Delta x}\begin{bmatrix} -3 & 4 & -1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & -4 & 3 \end{bmatrix}, \quad (N = 6)$$

In index notation, equation (4) can be written as:

$$y_i' = \sum_{j=1}^{N+1} D_{ij} y_j \tag{5}$$

Higher order derivatives can also be estimated by a simple repeated application of derivative matrix **D**, which leads equation (4) to become:

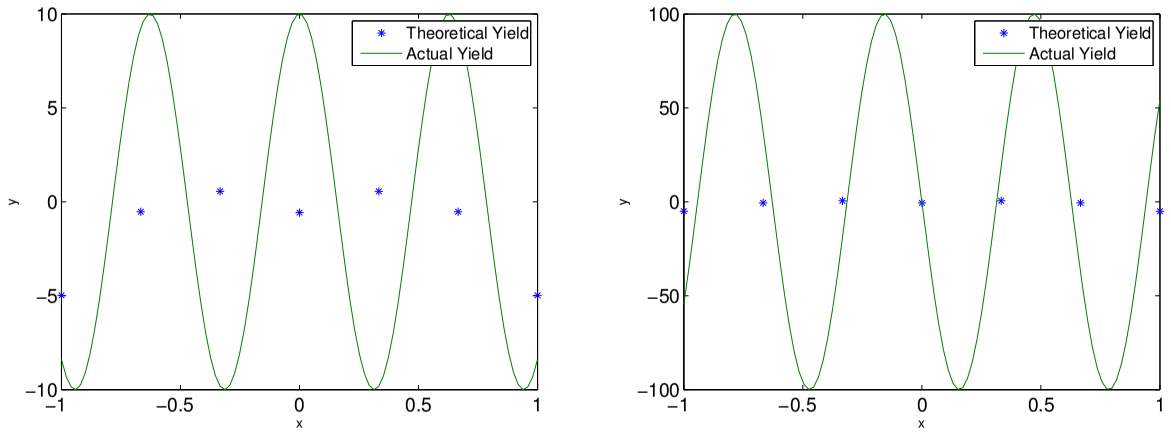
$$\mathbf{y}'' = \mathbf{D} \cdot \mathbf{D} \cdot \mathbf{y} \tag{6}$$

Similarly to before, the index notation of the second derivative in equation (6) is as follows:

$$y_i'' = \sum_{j=1}^{N+1} D_{ij} \sum_{k=1}^{N+1} D_{jk} y_k = \sum_{j=1}^{N+1} \sum_{k=1}^{N+1} D_{ij} D_{jk} y_k \tag{7}$$

In order to develop a complete solution for the central difference program in **Appendix A**, it is imperative to allocate any sized dimensional array when doing large iterative calculations. This is done so that the values of the element of a dimensional array can have a shape to store the solution and iterate them in a manner that is specified by the user. Afterwards, integers and counters are initialized to help iterate the loop. In this program, we are working with a square matrix and a single vector. Therefore, initializing the dimensions for matrix-vector multiplication will insure that the proper element of a dimensional array are used and stored iteratively. In a nested do loop the matrix-vector multiplication technique was first applied to estimate the first and second derivatives of equation 1 at each x_i. Then the **FORTRAN** intrinsic **matmul** feature

was used to obtain the same result as in the nested loop. If a user inputs N = 6 as the number of N+1 uniformly distributed points, then the program estimates the first and second derivatives at x_i, super-posed with each respective solution. Evidently, the discrete points in Figure (1a) demonstrate discrepancies along the curve of the first derivative. However, in Figure (1b), the plot seem to collapsed accurately into the curve, confirming the method of central difference. An error analysis also was conducted for the first derivative of $y' = 10\cos(10x)$ at $y'(0)$ = 10 as seen in Figure (2). At x_i=0 of this log-log plot, the magnitude of the error varied as a function of Δx. It is also evident that the error of Δx decreases as the order of magnitude N increases. However, only four plots demonstrate this result. One hypothesis is the use single precision. If double precision numbers were used, an 10^4 may appear on the graph.



(a) (b)

Figure 1: Representation of a sinusoidal matrix-vector multiplication on the first derivative (a) and second derivative (b) based on equation 1 while $x \in$ [-1,1]

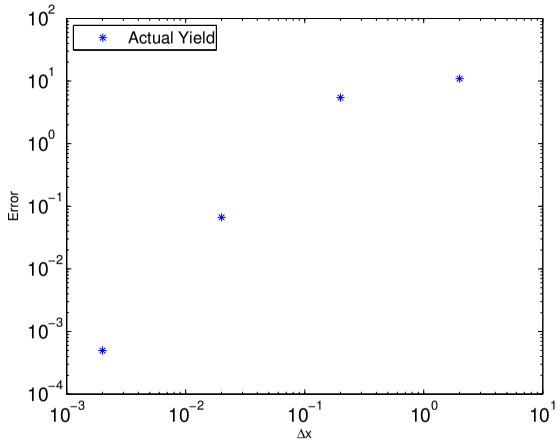


Figure 2: A log-log demonstration displaying the proximity of equation $y' = 10\cos(10x)$ as the value of N spans several orders in magnitude

Appendix A: FORTRAN Code

```
program matrix  ! program to approximate y(x); Read from LEFT TO RIGHT
implicit none   ! [!!!] indicates matmul [!!!!!] indicates do loops
integer :: n, istop, q
integer (kind=4) :: i,j,f
real (kind=4), dimension(:,:), allocatable :: D
real (kind=4), dimension(:), allocatable :: gp, gdp, g, gpb, gpc
real (kind=4):: x, dx, y, k, error
open(unit=20,file='matrix.out')
print*,' Input ONE desired integer for N amount of spaces'
read*,n
dx = 2./n
k=1./(2*dx)
print*, 'k= ',k
allocate(gpc(1:n+1))
allocate(gpb(1:n+1))
allocate(gp(1:n+1))
allocate(gdp(1:n+1))
allocate(g(1:n+1))
allocate(D(1:n+1,1:n+1))
D(1,1) = -3*k
D(1,2) = 4*k
D(1,3) = -1*k
D(n+1,n-1) = 1*k
D(n+1,n) = -4*k
D(n+1,n+1) = 3*k
do i=2,n !!!!!
D(i,i-1) = -1*k
D(i,i+1) = 1*k
enddo
print*,'Discrete [Matrix]'
print*, D
istop = n+1
i=1
q=1
do while(i¡=istop) !!!!!
x=-1+(2./n)*(i-1)
y=sin(10.*x)
G(q) = y
i=i+1
q=q+1
enddo
print*, 'F of (x) [Vector]'
print*, G
do i=1, istop [!!!!!]
   do q=1, istop
   gpb(i)=gpb(i)+D(i,q)*G(q)
   enddo
enddo
do i=1, istop [!!!!!]
   do q=1, istop
   gpc(i)=gpc(i)+D(i,q)*gpb(q)
   enddo
enddo
gp=matmul(D,G) [!!!]
gdp=matmul(D,gp) [!!!]
print*, 'First D Nested LOOP [!!!!!]'
print*,gpb
print*, 'First D MATMUL [!!!]'
print*,gp
print*, 'Second D Nested [!!!!!]'
print*,gpc
print*, 'Second D MATMUL [!!!]'
```

```
print*,gdp
print*,'error',10.-gpb((n/2.)+1.)
end program matrix
```