

Lecture 13: Implementation of Differentiation and Integration

Some **second order difference formulas** for computing the first derivative of a function $y(x)$ we will use in our code:

$$\begin{aligned} \text{centered } O(h^2) &: \frac{y_{n+1} - y_{n-1}}{2h} \\ \text{forward } O(h^2) &: \frac{-3y_n + 4y_{n+1} - y_{n+2}}{2h} \\ \text{backward } O(h^2) &: \frac{3y_n - 4y_{n-1} + y_{n-2}}{2h} \end{aligned}$$

where $h = \Delta x$ and $y_n = y(x_n)$

Now we look to implement a second order centered difference using the above formulas (for a 4th order implementation see page 48 of Kutz' notes):

```
% 2nd order accurate first derivative of a function
% over a domain x = [-2,2]
dx=.4;
x = (-2:dx:2);
u = sech(x);
n=length(x); % number of points

% Use second order forward difference
ux(1)=(-3*u(1)+4*u(2)-u(3))/(2*dx);

% Use second order center difference
for j=2:n-1
    ux(j)=(u(j+1)-u(j-1))/(2*dx);
end

% Use second order backward difference
ux(n)=(3*u(n)-4*u(n-1)+u(n-2))/(2*dx);
```

```
% Compare results:
ux_exact = -tanh(x).*sech(x);
error_1norm = norm(ux_exact-ux,1)

figure(1), plot(x,ux_exact,'b',x,ux,'r*')

edit ux_2ndOrder.m

% To see the second order accurate nature
% compute the difference of the two for different
% step sizes dx = 0.2, .1, .05, .025

% Insert the loop after assigning dx = .4
dx = .4;
for i = 1:4,
    dx = dx/2;

% Code from above goes here

end
```

Now that we know the method works we can apply it to a data set and take the derivatives.

It's best to first fit a polynomial spline to a set of data before numerically differentiating so that the derivatives are smooth.

`edit ux_2ndOrder_MoreReal.m`

Built in Integration formulas in MATLAB

- trapz
- quad
- cumtrapz

In MATLAB use the last data for x and ux to see if we can numerically integrate ux over the domain x

```
trapz(x,ux)
sech(x(end))-sech(x(1))
```

Try it with cumtrapz and quad:

```
% Cumulative trapezoidal method
cumtrapz(x,ux)
```

```
% Implements a recursive Simpson's  
% rule that is accurate to within  $10^{-6}$ 
```

```
help quad
```

```
quad(@myux,x(1),x(end))
```

More built in quadrature formulas in MATLAB

- `quadl` (Adaptive Lobatto quad)
- `quadgk` (Adaptive Gauss-Kronrod quad)
- `quadv` (For complex array-valued functions)
- `dblquad` (For double integrals, specify `quad`)
- `triplequad` (For triple integrals, specify `quad`)