Programming Tutorial:

Numerical Differentiation & Passing a function as input argument

Numerical Differentiation

Part 1. From datasets

Problem

Estimate the velocity and acceleration from datasets of position of an object.

```
clear all



t = 0:0.2:4;

x = [-5.87 -4.23 -2.55 -0.89 0.67 2.09 3.31 4.31 5.06 5.55 5.78 5.77 5.52 5.08
4.46 3.72 2.88 2.00 1.10 0.23 -0.59];
```

Create a function for numerical differentiation from a set of data

vector dydx=gradient(vector x, vector y)

Input: dataset (xi,yi), i=0 to m-1

Output: vector dydx at given **x(i)**

You can also create using 1D arrays such as

void gradient(1D-array x, 1D-array y, uint m, 1D-array dydx)

Procedure

First Derivative		
Method	**Formula**	**Truncation Error**
Two-point forward difference	$f'(x_i) = \frac{f(x_{i+1}) - f(x_i)}{h}$	$O(h)$
Three-point forward difference	$f'(x_i) = \frac{-3f(x_i) + 4f(x_{i+1}) - f(x_{i+2})}{2h}$	$O(h^2)$
Two-point backward difference	$f'(x_i) = \frac{f(x_i) - f(x_{i-1})}{h}$	$O(h)$
Three-point backward difference	$f'(x_i) = \frac{f(x_{i-2}) - 4f(x_{i-1}) + 3f(x_i)}{2h}$	$O(h^2)$
Two-point central difference	$f'(x_i) = \frac{f(x_{i+1}) - f(x_{i-1})}{2h}$	$O(h^2)$
Four-point central difference	$f'(x_i) = \frac{f(x_{i-2}) - 8f(x_{i-1}) + 8f(x_{i+1}) - f(x_{i+2})}{12h}$	$O(h^4)$

- With assumption of 3 or more data points,

  3-Point forward difference: for the first point x[0]

  2-Point central difference : for x[1]~x[m-2]

  3-Point backward difference: for the last point x[m-1]

- Write down a pseudocode for the function

- What would you do if you have 2 points?

Part 2. From equation

Create a function for numerical differentiation from given equation

vector dydx=gradientFunc(vector x, …)

- Define a function that defines the target equation.

```
// Example
double myFunc(const double x){
    double y = x*x;
    return y;
}
```

- How can you use ‘myFun()’ in the function of ‘gradientFunc()’?

In this tutorial, we will learn how to pass a mathematical function f(x) as an input argument to another function.

There are several methods of calling a function within another function.

(1) Calling the function that is defined globally or in the same scope

What if you want to define the equation “myFun” in another file or in main.c (not in your NM library)?

```
// Use global function or defined in the same header file

double myFunc(const double x){
   double y = x*x;
   return y;
}

void func_call(double xin)
{
    double y = myFunc(xin);
    printf ("Y_out = %d\n", y);
}

. . .
// in Main() function
double   xin=2.5;
func_call(xin);
```

(2) Pass a function as an input argument to another function.

```
// Defined in myNM.h, myNM.c

// Pass function as input argument
void func_call(double func(const double x), double xin)
{
        double yout = func(xin);
        printf("Y_out by my_func1= %f\n", yout);
}

….


// Defined in Main.cpp
double myFunc(const double x)
{
   double y = x*x;
}

// in Main()
double xin = 2;
double yout = 0;
func_call(myFunc, xin);
```

Procedure

- Define a function that defines the target equation.

  `double myFunc(const double x)`

- Modify ‘ gradient( )’ in Part 1 to get **y** data from the equation function (myFunc).

  `vector gradientFunc(double func(const double x), vector xin)`

- Check your function with the test equation

  “ y=x^3 “ at x=0:0.2:4