

Istanbul Technical University

Department of Computer Engineering

BLG 202E – Numerical Methods

Assignment 1

Solutions

**Solution 1.a**

is given as trigonometric equation and

is given as a function.

In the same time,

= then

can be transformed like this.

**Solution 1.b**

Derivative of :

=

However, in this case there can be cancellation errors for computing the approximation to the derivative of . In some value of h, will be so that . For avoiding from this situation, can be written as = using trigonometric equations.

**In Matlab:**

x0 = 1.2; %%Initial Value

i = -20:1:0;

h = 10.^i; %%Desired value of h form 1e-20 to 1

syms x

derivative = inline(diff(sin(x), 'x')); %%derivative of sin(x) without approximation

f0 = cos((2\*x0 + h) / 2);

f1 = sin(h/2);

approximationDerivativeWithoutCancellationError = (2.\*f0.\*f1)./h;

err1 = abs(approximationDerivativeWithoutCancellationError - derivative(x0));

approximationDerivativeWithCancellationError = (sin(x0 + h) - sin(x0)) ./ h;

err2 = abs(approximationDerivativeWithCancellationError - derivative(x0));

fprintf('h : %d\n\n', h')

fprintf('Without cancellation error: %f\n\n', approximationDerivativeWithoutCancellationError')

fprintf('With cancellation error: %f\n\n', approximationDerivativeWithCancellationError')

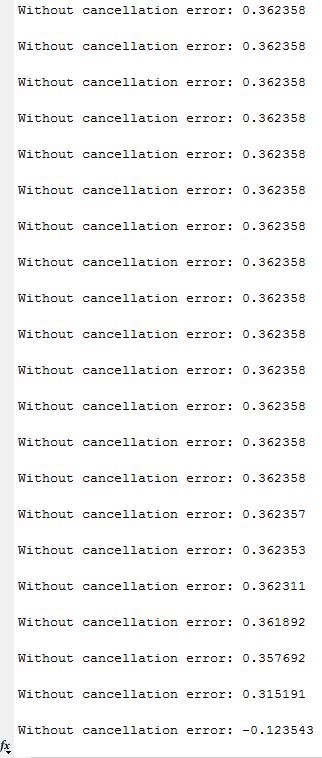
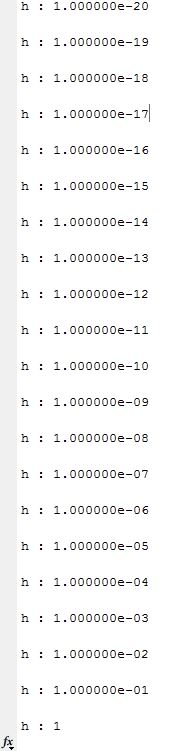
Note: Some required explanation about variables are given following:

**derivative** = derivative of sin(x) without approximation

**approximationDerivativeWithoutCancellationError** = derivative of sin(x) with respect to h = 1e-20; 1e-19; … 1 and = 1.2. In order to get rid of cancellation error, definition of derivative transformed to another function using trigonometric equations.

**approximationDerivativeWithCancellationError** = derivative of sin(x) with respect to h = 1e-20; 1e-19; … 1 and = 1.2. Definition of derivative is used.

With this Matlab script, we can see that err1 is zero for the first six values of h; on the other hand, err2 is never zero because of the cancellation error.



**Solution 2**

Horner’s Method says that polynomial of degree requires multiplications and additions which be totally elementary operations. For example;

can be written as following

as nested. In this case function can be calculated elementary operations (2 multiplications and 2 additions). The operation will iterate n times to reach polynomial of degree . Hence complexity will be .

**Solution 3**

The finite floating numbers can be represented using floating point system (i.e., some finite integer base and precision ). However, some values continue indefinitely like , π, . These values would not represent precisely. Briefly, no.

**Solution 4.a**

In the fixed point iteration there is .

has two fixed points:

**Solution 4.b**

Iterations can converge to fixed point if only if . In this case,

and . Converging fixed point will be .

For example, 0.55 can be chosen as initial guess as .

0.55 is converge to the which be fixed point of , not

**Solution 4.c**

This is a **contraction** by the factor of . So

. Since ρ < 1, we have → 0 as k→∞.

By a factor of 10, we get. So,

Taking of both sides yields Rate of converges can be defined as **k** was the iteration numbers so .

= ρ

This yields

Briefly, 4 iterations will be required to reduce the convergence error by a factor of 10.

**Solution 5.a**

In the Newton-Raphson Method, the theorem of the equation can be written following:

Let take the initial guess

Absolute relative approximate error:

Absolute relative approximate error:

Still absolute relative approximate error greater than 1%.

Absolute relative approximate error:

Absolute relative approximate error is less than 1%.

**Solution 5.b**

syms x

x0 = 50; %%Initial x

xn = 0; %%Next value of x

i = 0; %%Iteration step

err = 1; %% Absolute relative approximate error

fprintf('Initial x = %.20f\n', x0)

while err > 10^-2

f = inline((40\*x^1.5)-875\*x+35000, 'x'); %%f(x)

derivative = inline(diff((40\*x^1.5)-875\*x+35000, 'x')); %%f'(x)

xn = x0 - f(x0)/derivative(x0); %%Newton-Raphson Method Iteration

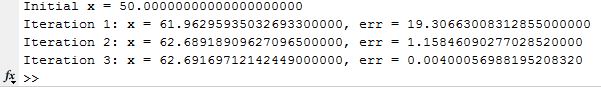
err = (xn-x0)/xn\*100;

x0 = xn;

i = i +1;

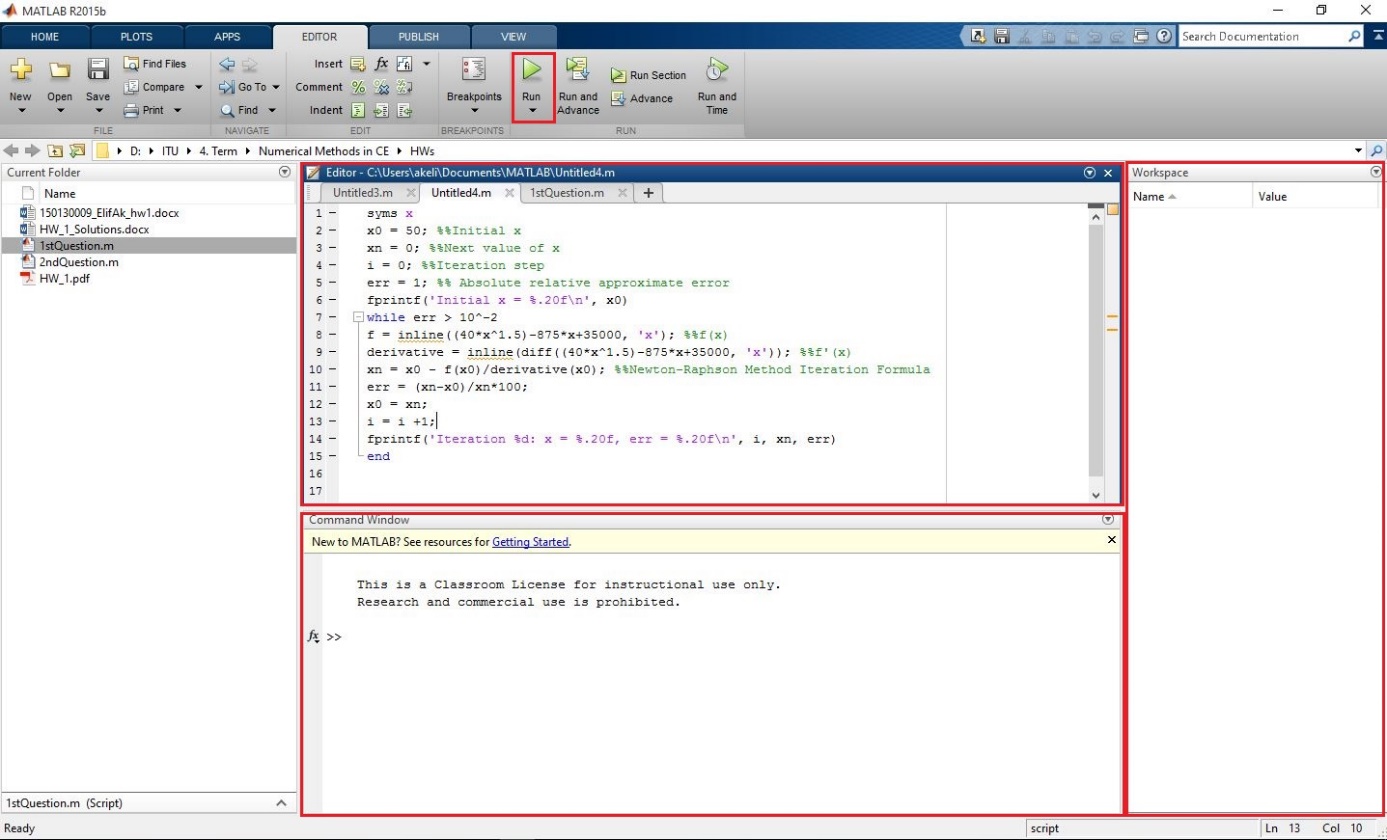
fprintf('Iteration %d: x = %.20f, err = %.20f\n', i, xn, err)

end

****

**How to run codes?**

To run extension of “.m” files must be in the folder which MATLAB path. If the MATLAB files are in another path, either MATLAB current folder must be changed or is added to MATLAB path. After all these done, while current window is “Editor Window” script is run like as following:



Using “Run” button which placed the top of the program, MATLAB script is run. Outputs can be followed from “Command Window” which is the bottom of the program. And also, in the right side, values can be seen using in the script.