29.02.2016
Elif Ak
150130009

Istanbul Technical University
Department of Computer Engineering

BLG 202E – Numerical Methods
Assignment 1
Solutions

Solution 1.a

$\sin\alpha - \sin\beta = 2\sin(\alpha-\beta)\cos(\alpha+\beta)$ is given as trigonometric equation and

$f_1(x_0, h) = \sin(x_0 + h) - \sin(x_0)$ is given as a function.

In the same time,

$f_2(x_0, h) = \sin(x_0 + h) - \sin(x_0) = 2\cos\left(\frac{x_0+h+x_0}{2}\right)\sin\left(\frac{x_0+h-x_0}{2}\right)$ then

$f_2(x_0, h) = 2\cos\left(\frac{2x_0+h}{2}\right)\sin\left(\frac{h}{2}\right)$ can be transformed like this.

Solution 1.b

Derivative of $f(x) = \sin(x)\ at\ x = x_0$:

$f'(x) = \frac{f(x_0+h)-f(x_0)}{h} = \frac{sin(x_0+h)-sin(x_0)}{h}$

However, in this case there can be cancellation errors for computing the approximation $\frac{f(x_0+h)-f(x_0)}{h}$ to the derivative of $f(x) = \sin(x)\ at\ x = x_0$. In some value of h, $\frac{sin(x_0+h)-sin(x_0)}{h}$ will be $\frac{sin(x_0)-sin(x_0)}{h}$ so that $\frac{0}{h}$. For avoiding from this situation, $\frac{sin(x_0+h)-sin(x_0)}{h}$ can be written as $\frac{2\cos\left(\frac{x_0+h+x_0}{2}\right)\sin\left(\frac{x_0+h-x_0}{2}\right)}{h} = \frac{2\cos\left(\frac{2x_0+h}{2}\right)\sin\left(\frac{h}{2}\right)}{h}$ 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 x_0 = 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 x_0 = 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.

```
h : 1.000000e-20    Without cancellation error: 0.362358
h : 1.000000e-19    Without cancellation error: 0.362358
h : 1.000000e-18    Without cancellation error: 0.362358
h : 1.000000e-17    Without cancellation error: 0.362358
h : 1.000000e-16    Without cancellation error: 0.362358
h : 1.000000e-15    Without cancellation error: 0.362358
h : 1.000000e-14    Without cancellation error: 0.362358
h : 1.000000e-13    Without cancellation error: 0.362358
h : 1.000000e-12    Without cancellation error: 0.362358
h : 1.000000e-11    Without cancellation error: 0.362358
h : 1.000000e-10    Without cancellation error: 0.362358
h : 1.000000e-09    Without cancellation error: 0.362358
h : 1.000000e-08    Without cancellation error: 0.362358
h : 1.000000e-07    Without cancellation error: 0.362358
h : 1.000000e-06    Without cancellation error: 0.362357
h : 1.000000e-05    Without cancellation error: 0.362353
h : 1.000000e-04    Without cancellation error: 0.362311
h : 1.000000e-03    Without cancellation error: 0.361892
h : 1.000000e-02    Without cancellation error: 0.357692
h : 1.000000e-01    Without cancellation error: 0.315191
h : 1               Without cancellation error: -0.123543
```

29.02.2016
Elif Ak
150130009

Solution 2

Horner's Method says that polynomial of degree n requires n multiplications and n additions which be totally $2n$ elementary operations. For example;

$f(x) = 4x^2 + 5x + 9$ can be written as following

$f(x) = x(4x + 5) + 9$ as nested. In this case function can be calculated 4 elementary operations (2 multiplications and 2 additions). The operation will iterate n times to reach polynomial of degree n. Hence complexity will be $O(n)$.

Solution 3

The finite floating numbers can be represented using floating point system (i.e., some finite integer base β and precision t). However, some values continue indefinitely like $\frac{1}{3}$, π, e. These values would not represent precisely. Briefly, no.

Solution 4.a

In the fixed point iteration there is $f(x) = 0 \ \ can\ be\ rewritten\ as\ x = g(x)$.

$g(x) = x^2 + \frac{3}{16}$ has two fixed points:

$g(x) = x = x^2 + \frac{3}{16} \quad \rightarrow \quad x_1 = 0.25\ ,\ x_2 = 0.75$

Solution 4.b

Iterations can converge to fixed point if only if $|g'(x^*)| < \rho,\ \rho < 1$. In this case,

$|g'(0.25)| = 0.5$ and $|g'(0.75)| = 1.5$. Converging fixed point will be $x_1 = 0.25$.

For example, 0.55 can be chosen as initial guess as x_0.

$x_0 = 0.55, \qquad x_{n+1} = g(x_n)$

$x_1 = 0.49,$

$x_2 = 0.4276$

.

.

.

$x_{17} = 0.25001250092667$

$x_{18} = 0.25000625061961$

0.55 is converge to the $\mathbf{0.25}$ which be fixed point of $|g'^{(x^*)}| < \rho,\ \rho < 1$, not $\mathbf{0.75}$

Solution 4.c

$|x_{k+1} - x^*| = |g(x_k) - g(x^*)| \leq \rho|x_k - x^*|$ This is a **contraction** by the factor of ρ. So

$|x_{k+1} - x^*| \leq \rho|x_k - x^*| \leq \rho^2|x_k - x^*| \leq \cdots \leq \rho^{k+1}|x_0 - x^*|$. Since ρ < 1, we have $\rho^{k+1} \to 0$ as k$\to\infty$.

$\rho|x_k - x^*| \leq \rho^{k+1}|x_0 - x^*| \quad \rightarrow \quad |x_k - x^*| \leq \rho^k|x_0 - x^*|$

By a factor of 10, we get $|x_k - x^*| \approx 0.1|x_0 - x^*|$. So,

$\rho^k \approx 0.1$ Taking $\log_{10}$ of both sides yields $k\log_{10}\rho \approx -1$. Rate of converges can be defined as rate $= -\log_{10}\rho$. **k** was the iteration numbers so $k = [1/rate]$.

$k = [1/-\log_{10}\rho]$

$\lim_{n\to\infty}\left(\frac{x_{n+1}-x^*}{x_n-x^*}\right) = |g'(x^*)|$ = ρ $\quad \rightarrow \quad |g'(x^*)| = g'(0.25) = 0.5$

$\rho = 0.5 \quad \rightarrow \quad k = [1/-\log_{10}0.5]$

$rate = -\log_{10}0.5 \approx 0.301$. This yields $k = 4$.

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:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

$$x_{k+1} = x_k - \frac{40x_k^{1.5} - 875x_k + 35000}{60x_k^{0.5} - 875}$$

Let take the initial guess $x_0 = 50$

Iteration I

$$x_1 = 50 - \frac{40 * 50^{1.5} - 875 * 50 + 35000}{60 * 50^{0.5} - 875}$$

$x_1 = 61.962959350326933$

Absolute relative approximate error:

$$\frac{61.962959350326933-50}{61.962959350326933} * 100 = 19.306630083128550\ \%$$

Iteration II

$$x_2 = 61.962959350326933 - \frac{40*61.962959350326933^{1.5}-875*61.962959350326933+35000}{60*61.962959350326933^{0.5}-875}$$

$$x_2 = 62.689189096270965$$

Absolute relative approximate error:

$$\frac{62.689189096270965 - 61.962959350326933}{62.689189096270965} * 100 = 1.158460902770285\ \%$$

Still absolute relative approximate error greater than 1%.

Iteration III

$$x_3 = 62.689189096270965 - \frac{40*62.689189096270965^{1.5}-875*62.689189096270965+35000}{60*62.689189096270965^{0.5}-875}$$

$$x_3 = 62.691697121424490$$

Absolute relative approximate error:

$$\frac{62.691697121424490 - 62.689189096270965}{62.691697121424490} * 100 = 0.004000569881952\ \%$$

Absolute relative approximate error is less than 1%.

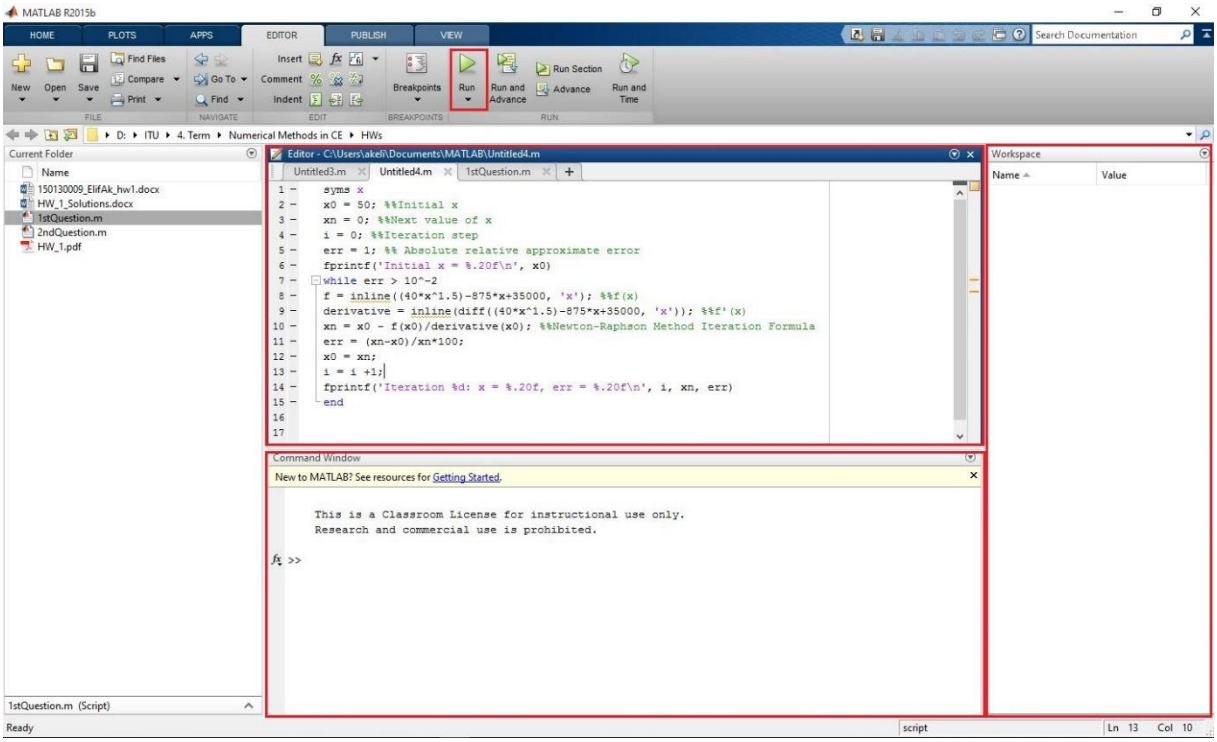
Solution 5.b

```
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
```

```
Initial x = 50.000000000000000000
Iteration 1: x = 61.962959350326933000000, err = 19.306630083128550000000
Iteration 2: x = 62.689189096270965000000, err = 1.158460902770285200000
Iteration 3: x = 62.691697121424490000000, err = 0.004000569881952083200
fx >>
```

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.