PW1 – Computational Methods – M1 E3A International Track

Names: BELLATRECHE Maroua

IFTAKHER Mohammed Akib

The objective:

implement and compare different methods to solve non-linear equations of the type: f(x) = 0.

I- 1-variable Functions

i- Dichotomy Method

Here's an explanation of how it works:

It is also known as the Bisection Method, and it is a numerical method for determining the root of a function. It is commonly used in continuous functions where two values must be known. In our scenario, these two values are 1 and 2. Along with the values, the sign of the values is vital in this procedure. Before iterating, the multiplication of both functions must be compared to zero to see whether the selected interval is within the right range. Otherwise, the procedure will not commence. Then the iteration begins where t he middle value of the intervals is determined, which is 1.50000 in this instance . Following that, two new subintervals are considered. The first interval points are the recently calculated middle and lower interval points, whereas the other interval points are the middle and higher interval points. At this stage, function multiplication must be implemented for both subintervals. If any of the function's multiplications change sign, it suggests that it must have a root. As a result, the other subinterval (where the sign did not change) will be excluded in subsequent iterations. A new middle point must be determined from the previously acceptable subinterval, same as in the first iteration. And the process is repeated until the correct root is identified by bisecting the interval. To find the appropriate root, the function compares the difference of subintervals with the desired precision in each iteration. If the desired precision is achieved, the iteration will be terminated immediately. The iteration number is 10 in this example, and the real root is 1.87598.

If the starting values are 4 and 5, the first middle point is 4.5. Following the methods outlined above, the final root yields 4.69434. These intervals, like the last input, need ten iterations.

```
% This function prints all of iteration, the length of the last element and
% the number of iterations needed
clc; clear;
x = 0:0.1:5;
y = cos(x) .* cosh(x) +1;
subplot(2,2,1);
plot(x,y);
title('By Dichotomy Method');
dich(1, 2, 0.001); % here we can see that P10 is 1.87598 => precision respected
```

```
P1 = 1.50000
P2 = 1.75000
P3 = 1.87500
P4 = 1.93750
```

```
P5 = 1.90625
P6 = 1.89062
P7 = 1.88281
P8 = 1.87891
P9 = 1.87695
P10 = 1.87598
the length of the last sigment is -9.765625e-04
the number of iterations needed is 10
```

```
dich(4, 5, 0.001); % here we can see that P10 is 4.69434 => precision respected
```

```
P1 = 4.50000
P2 = 4.75000
P3 = 4.62500
P4 = 4.68750
P5 = 4.71875
P6 = 4.70312
P7 = 4.69531
P8 = 4.69141
P9 = 4.69336
P10 = 4.69434
the length of the last sigment is -9.765625e-04
the number of iterations needed is 10
```

ii- Regula Falsi Method

here an explanation of how it works:

It's also referred to as the "false position approach." It's quite similar to the bisection technique algorithm. A root exists between two intervals if the function is continuous between intervals a and b and the result of the function multiplication is negative, like in the prior instance. The choice of the median point, however, is the most significant change in this case. Unlike the Dichotomy Approach, this method requires the intersection point to be computed along the abcissae axis. Following that, the methods are identical to the dichotomy technique.

$$c = b - \frac{f(b)(b-a)}{f(b)-f(a)} = \frac{af(b)-bf(a)}{f(b)-f(a)}$$

The considered intervals in this case, for example, are [1,2]. The conditions are tested in the first step to see if the specified interval has a root between them. The iteration process then commences, with the intersaction point being calculated using the above equation. The itersaction point in this case is 1.76426. Following the selection of the intersection point, the subinterval function multiplication is used to detect the changes in sign. The proper sub interval is kept for subsequent repetition while the incorrect one is eliminated by checking the changes in sign. Iteration continues after that until the desired accuracy is obtained. The correct root, in this case 1.87503.

Again, if the interval is changed to 4 and 5, the final root value would be 4.69409. Although the iteration number is increased compared to previous input, it is still faster comapared to dichotomy method.

```
% This function prints all of iteration, the length of the last element and
% the number of iterations needed
subplot(2,2,2);
plot(x,y);
title('By Regula Falsi Method');
reg_fal(1, 2, 0.001);
```

```
C1 = 1.76426
C2 = 1.86504
C3 = 1.87423
C4 = 1.87503
the length of the last sigment is -1.249712e-01
the number of iterations needed is 4
```

```
reg_fal(4, 5, 0.001);
```

```
C1 = 4.43315
C2 = 4.61724
C3 = 4.67361
C4 = 4.68880
C5 = 4.69273
C6 = 4.69374
C7 = 4.69400
C8 = 4.69407
C9 = 4.69409
the length of the last sigment is -3.059147e-01
the number of iterations needed is 9
```

Note: it can be seen that the number of iterations needed by Regula Falsi is less in iterations than the dichotomy method.

iii- Secant Method

here an explanation of how it works

The Secant technique is a numerical approach for solving single-unknown problems. It's a modification on the Regula falsi approach. After each iteration, f(a)f(b)<0 does not need to be tested again and again. If f(x) is continuous on [a,b], the secant technique always converges to a root of f(x)=0. Two points are specified at the start of the process. These points were chosen at random. The two points in this example are 1 and 2. The interval's root can be then calculated by looping the equation below for numerous times and comparing the root function with the required accuracy.

$$x = a - f(a)\frac{b-a}{f(b)-f(a)}$$

After replacing the values with the chosen points, the first value of the secant method is 1.76426. The desired precision has been checked in the following question which in this case 10^-9. If the function of newly calculated value is over this precision, a loop will continue until the correct root point is determined. After four iteration, the correct root (1.8750 was found using the secant method.

In case of changing the input, once again it can be seen that the iteration number will increase but compared previous two the increment is still low.

```
% This function prints all of iteration, the length of the last element and
% the number of iterations needed
subplot(2,2,3);
plot(x,y);
title('By Secant Method');
secant(1, 2, 0.001);
```

```
P1 = 1.76426
P2 = 1.86504
P3 = 1.87597
P4 = 1.87510
the length of the last sigment is -8.752912e-04
the number of iterations needed is 4
```

```
secant(4, 5, 0.001);
```

```
P1 = 4.43315
P2 = 4.61724
P3 = 4.72040
P4 = 4.69196
P5 = 4.69403
P6 = 4.69409
the length of the last sigment is 5.626134e-05
the number of iterations needed is 6
```

Note: it can be seen that all of the above methods diverges to the same value.

iv- Newton Method

here an explanation of how it works

Newton's method, often known as the Newton–Raphson method, is a numerical analysis root-finding strategy that provides successively better approximations to the roots (or zeroes) of a real-valued function. In this scenario, it's important to keep in mind that the function needs to be continually derivable. As a result, the first condition of this technique is that the function's derivative should not be zero. Then, in the following loops, a specific equation is applied for the purpose of determining root that used a given point. The equation can be found below.

$$x = x_1 - \frac{y_1}{f'(x_1)} = x_1 - \frac{f(x_1)}{f'(x_1)}.$$

After each iteration, the required precision is compared to the absolute difference between the newly computed point and the prior point. The final point can be defined as the proper root point if the precession is matched. The very first point in this situation is 1. 2. The next computed point was 2.651913 after the first iteration. The accuracy of the new value is then verified. Because the precision was not met, the loop iterated four more times before arriving at the final precised root value of 1.875104.

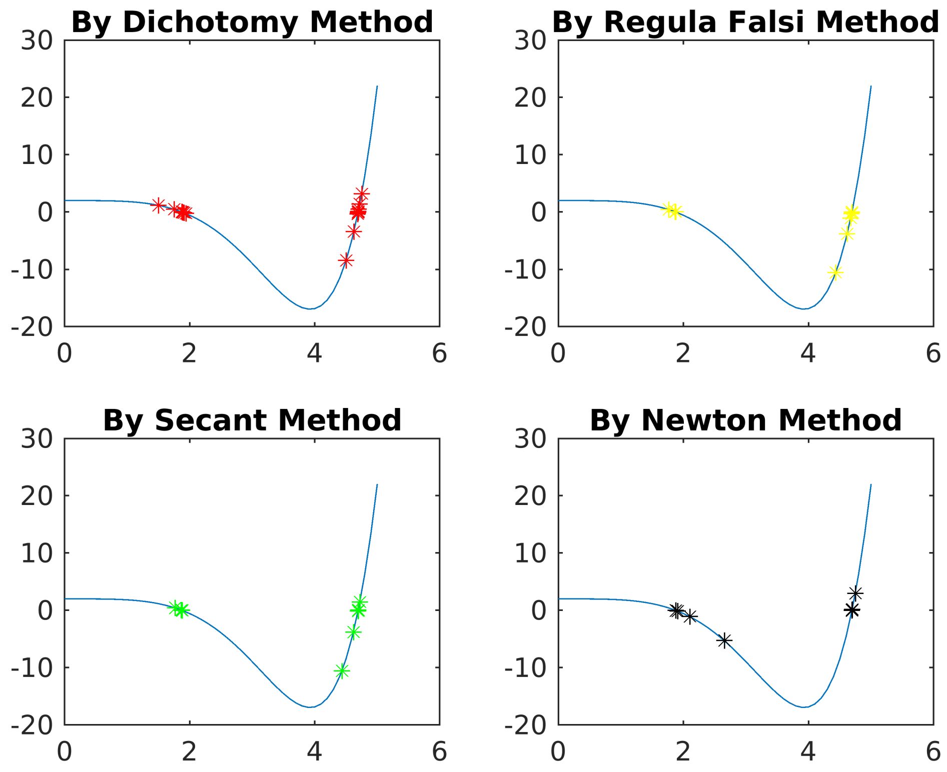
```
% This function prints all of iteration, the length of the last element and
% the number of iterations needed
```

```
subplot(2,2,4);
plot(x,y);
title('By Newton Method');
newton(1.2, 0.001); %here we need to give a value close to the root so it does not dive
```

```
x1 = 2.651913
x2 = 2.099757
x3 = 1.905171
x4 = 1.875749
x5 = 1.875104
the length of the last sigment is -6.442258e-04
the number of iterations needed is 5
```

```
%newton(1, 0.001);   % if we give 1 as starting value it diverges to -51!
newton(4.5, 0.001);
```

```
x1 = 4.745936
x2 = 4.696648
x3 = 4.694098
x4 = 4.694091
```



```
the length of the last sigment is -6.634843e-06
the number of iterations needed is 4
```

```
%newton(4, 0.001);   % same for this at 4 diverges we need to specifu 4.5 or so
```

Conclusion f the first part of the lab:

It can be noted that, by the newton method it's faster but the main disadantage is that it can diverge and the choice of the starting point is very important.

Dichotomy	9.65625e-04
Regula-Falsi	-1.249712e-01
Secant	-8.752912e-04
Newton	-6.442258e-04

The second part: II- Non-linear systems of equations

Newton's technique is a method for locating the roots of differentiable functions that use iterative local linearization to approximate the roots. To identify two unknown roots in nonlinear equations, two starting values are necessary. The process starts with the equation below, which includes the Jacobian matrix and starting values.

$$u^{(k+1)} = u^{(k)} - \emptyset^{-1}(u^{(k)}F(u^{(k)}))$$

$$\emptyset = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix}$$

A recursive function has been utilised in this approach, which accepts as input a 0.6 and 0.5 and required precision number (10^-9) and an iteration number (0). The programme then employs the previously specified equations to determine new points in the following phase. At first sys function is called, which produces total values of the mentioned equations after replacing with the initial points, and then the inverse Jacobian called to substract from the previous function. Afterwards a condition is triggered to compare the desired precision with the absolute value of the function of the system. The recursion will continue if the condition is not satisfied. Meanwhile, the recursion number will be kept in a counter. In this scenario, the precision was achieved after 4 iterations, with the newly computed points being 0.3938494528..... and 0.03214273894.....

```
u = [0.6;0.5]; % here is our input
new1(u,10^-9,0);
```

```
The number of iterations is 4
ans =
```

$$\begin{pmatrix} 0.39384945283486890987012476681305 \\ 0.032142738943740672584184748582291 \end{pmatrix}$$

Remark:

Newton method in non-linear system can converge to root very fastly. When the value goes near to the root, number of significant digits double and it also provide precise roots. But it has some issue regarding the intial point selection. If the intial points are away from the actual root, it may take indefinite amount of time.

The third part: III- Zeros of polynomials

Using the function newton_p which takes a starting point for the root of the polynomial with the given precision.

```
newton_p(1.2,1e-6);
```

```
x1 = 0.992795
x2 = 0.999992
x3 = 1.000000
x4 = 1.000000
the length of the last sigment is 1.075973e-11
the number of iterations needed is 4
```

```
newton_p(-10.2,1e-6);
```

```
x1 = -10.006016
x2 = -10.000006
x3 = -10.000000
x4 = -10.000000
the length of the last sigment is 5.112355e-12
the number of iterations needed is 4
```

```
newton_p(5.2,1e-6);
```

```
x1 = 5.011459
x2 = 5.000041
x3 = 5.000000
x4 = 5.000000
the length of the last sigment is -5.409246e-10
the number of iterations needed is 4
```

Remark:

We can see that it's very fast after 3 or 4 iterations it directly diverges and the length of the last segment is in terms of e-12 means it is very precise too.

All functions that we used are commented bellow:

```
% function return_val = f(x)
```

```
%
%     return_val = cos(x) .* cosh(x) +1;
%
% end
%
%
% function return_val = df(x)
%
%     return_val = -sin(x) .* cosh(x) + sinh(x) .* cos(x);
%
% end
%
% function [] = dich(a, b, epsilon)
%
% n = (log(b-a) - log(2 * epsilon) ) / log(2) + 2;
%
% if f(a)*f(b)<0
%     for i=1:n
%         c = (a+b)/2;
%         hold on
%         plot(c,f(c),'r*')
%         fprintf('P%d = %.5f\n',i,c)
%         if  abs(c-a)<epsilon||abs(c-b)<epsilon
%             fprintf("the length of the last sigment is %d\n", (c-b));
%             fprintf("the number of iterations needed is %d\n",i);
%             break
%         end
%         if f(a)*f(c)<0
%             b = c;
%         elseif f(c)*f(b)<0
%             a = c;
%         end
%     end
% else
%     disp("Wrong range");
% end
% end
%
%
% function [] = reg_fal (a,b,epsilon)
%
% n = (log(b-a) - log(2 * epsilon) ) / log(2) + 2;
%
% if f(a)*f(b)<0 && a<b
%     for i=1:n
%         c = (a*f(b)-b*f(a))/(f(b)-f(a));
%         fprintf('C%d = %.5f\n',i,c)
%         hold on
%         plot(c,f(c),'y*')
%         if abs(f(c))<epsilon
%             fprintf("the length of the last sigment is %d\n", (c-b));
%             fprintf("the number of iterations needed is %d\n",i);
%             break
%         end
%         if f(a)*f(c)<0
%             b = c;
%         elseif f(c)*f(b)<0
%             a = c;
%         end
%
%     end
% else
%     disp("Wrong range");
% end
% end
%
%
% function [] = secant(a, b, epsilon)
% n = 20;
%     for i = 1:n
%
%         c = b - f(b)*(b-a)/ (f(b) - f(a));
%         fprintf('P%d = %.5f\n',i,c)
%         hold on
%         plot(c,f(c),'g*')
%
%         if (abs(f(c))>epsilon)
%             a = b;
%             b = c;
%         else
%             fprintf("the length of the last sigment is %d\n", (c-b));
%             fprintf("the number of iterations needed is %d",i);
%             break
%
%
%         end
%
%     end
% end
%
%
% function [] = newton(a,epsilon)
% % choose 1.2 as starting point
% n = 20;
%
% if (df(a)~=0)
%     for i =1:n
%         b = a - f(a)/df(a);
%         fprintf('x%d = %8f\n',i,b)
%         hold on
%         plot(b,f(b),'k*')
%         if abs(b-a) < epsilon
%             fprintf("the length of the last sigment is %d\n", (b-a));
%             fprintf("the number of iterations needed is %d\n",i);
%             break
%         end
%         if df(b)==0
%             disp("Newton Raphson Failed");
%         end
%         a = b;
%     end
%  else
%     disp("Newton Raphson Failed");
% end
% function return_val = f_p(x)
%
%     return_val = x^3 + 4 * x^2 - 55 * x + 50;
%
% end
%
% function return_val = df_p(x)
%
%     return_val = 3*x^2 + 8 * x -55;
%
% end
%
% function [] = newton_p(a,epsilon)
% % choose 1.2 as starting point
% n = 20;
%
% if (df_p(a)~=0)
%     for i =1:n
%         b = a - f_p(a)/df_p(a);
%         fprintf('x%d = %8f\n',i,b)
%         if abs(b-a) < epsilon
%             fprintf("the length of the last sigment is %d\n", (b-a));
%             fprintf("the number of iterations needed is %d\n",i);
%             break
%         end
%         if df_p(b)==0
%             disp("Newton Raphson Failed");
%         end
%         a = b;
%     end
%  else
%     disp("Newton Raphson Failed");
% end
%
% function [] = newton_p(a,epsilon)
% % choose 1.2 as starting point
% n = 20;
%
% if (df_p(a)~=0)
%     for i =1:n
%         b = a - f_p(a)/df_p(a);
%         fprintf('x%d = %8f\n',i,b)
%         if abs(b-a) < epsilon
%             fprintf("the length of the last sigment is %d\n", (b-a));
%             fprintf("the number of iterations needed is %d\n",i);
%             break
%         end
%         if df_p(b)==0
%              disp("Newton Raphson Failed");
%          end
%          a = b;
%      end
%  else
%      disp("Newton Raphson Failed");
% end
%
% function [J] = Jacobian(r)
% syms x y
% u = [x;y];
% JJ=[diff(sys1(u),u(1)) diff(sys1(u),u(2))];
% J=vpa(subs(JJ,u,r));
% end
%
% function [F] = sys1(u)
% F = [exp(u(1)*u(2))+u(1)^2+u(2)-1.2;u(1)^2+u(2)^2+u(1)-0.55];
% end
%
% function [p1] = new1(u,e,c)
%
%  v = u- inv(Jacobian(u))*sys1(u);
% if abs(sys1(v))>e
%      c = c+1;
%     new1(v,e,c)
%   else
%   p1 = v;
%   counter = c;
%   fprintf("The number of iterations is %d", counter);
% end
% end
```