Numerical Analysis Project 1

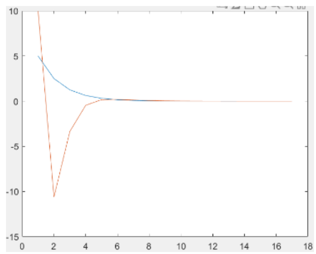
Joseph Pietroluongo 5901749

Language: MATLAB

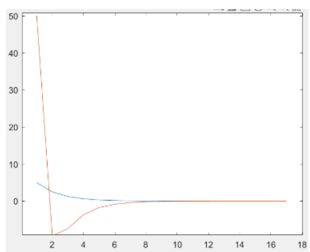
Bisection Method

Input:

Input function a	x*x*x-5*x*x+2*x
Input endpoints	0,10
Input tolerance	0.0001
maximum number of iterations	10000
Approximate solution P	0.00007629
F(P)	0.00015256
Number of iterations	17



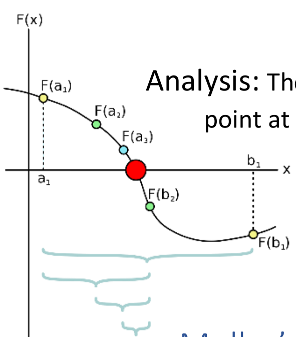
Input function b	x*x*x-2*x*x-5
Input endpoints	0,10
Input tolerance	0.0001
maximum number of iterations	10000
Approximate solution P	0.00007629
F(P)	-0.00038148
Number of iterations	17



Output:

Function	a
5	10.0000
2.5000	-10.6250
1.2500	-3.3594
0.6250	-0.4590
0.3125	0.1672
0.1562	0.1942
0.0781	0.1262
0.0390	0.0706
0.0195	0.0372
0.0097	0.0191
0.0048	0.0096
0.0024	0.0049
0.0012	0.0024
0.0006	0.0012
0.0003	0.0006
0.0001	0.0003
0.00007	0.0002

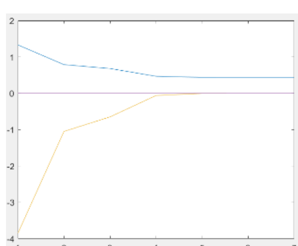
Function	b
5.0000	50.0000
2.5000	-9.3750
1.2500	-7.4219
0.6250	-3.6621
0.3125	-1.7273
0.1563	-0.8263
0.0781	-0.4024
0.0391	-0.1983
0.0195	-0.0984
0.0098	-0.0490
0.0049	-0.0245
0.0024	-0.0122
0.0012	-0.0061
0.0006	-0.0031
0.0003	-0.0015
0.0002	-0.0008
0.0001	-0.0004



Analysis: The bisection method seeks to find the midpoint of 2 points and iterate through that to determine the point at which the graph passes through the axis to determine the roots. Given the 2 functions, both are cubic and when given the same endpoints they produced the same result which is the nature of cubic functions and is expected.

Muller's Method:

Input function a	x*x*x-5*x*x+2*x
Input starting points	0,1,10
Input tolerance	0.0001
maximum number of iterations	1000
Number of iterations	9

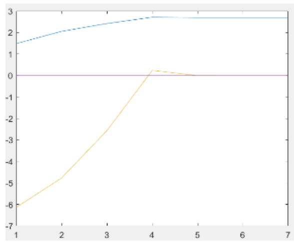


Function	a		
1.333333	0	-3.85185	0
0.790307	0	-1.0487	0
0.683223	0	-0.6486	0
0.469332	0	-0.05932	0
0.439835	0	-0.00252	0
0.438453	0	-0.00001	0
0.438447	0	0	0

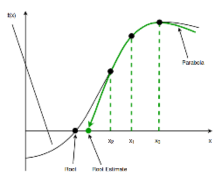
Function	a		
1.333333	0	-3.85185	0
0.790307	0	-1.0487	0
0.683223	0	-0.6486	0

0.469332	0	-0.05932	0
0.439835	0	-0.00252	0
0.438453	0	-0.00001	0
0.438447	0	0	0

Input function b	x*x*x-2*x*x-5
Input starting points	0,1,10
Input tolerance	0.0001
maximum number of iterations	1000
Number of iterations	9

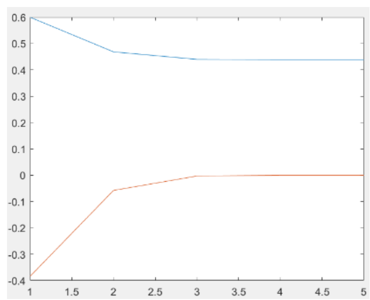


Analysis: Muller's method attempts to find the root estimate by projecting a parabola to the x axis through 3 function values. These values go through the function to determine the root by iterating through and then estimating. With the points supplied the root found is 0 for both input functions, this is because they are both cubic and would naturally pass through the origin.

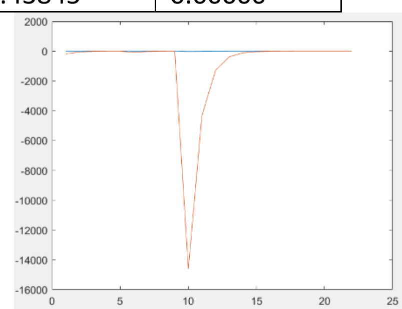


Newton's Method

Input function a	x*x*x-5*x*x+2*x
Input initial value	1
Input tolerance	0.0001
maximum number of iterations	10000
Approximate solution P	0.43845
F(P)	-1.0251578753e-10
Number of iterations	5



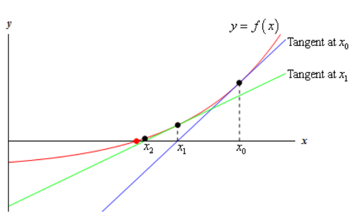
Function	**a**
0.60000	-0.38400
0.46849	-0.05762
0.44006	-0.00293
0.43845	-0.00001
0.43845	0.00000



Input function b	x*x*x-2*x*x+5
Input initial value	1
Input tolerance	0.0001
maximum number of iterations	10000
Approximate solution P	2.69065
F(P)	1.6342482922e-13
Number of iterations	22

Function	**b**
-5	-180.00000
-3.10526	-54.22831
-1.79379	-17.20714
-0.77126	-6.64848
0.59404	-5.49614
-3.57758	-76.38766
-2.12830	-23.69978
-1.05602	-8.40798
0.05474	-5.00583
-23.78451	-14591.37600
-15.64313	-4322.40290

Function	**B continued**
-10.21771	-1280.54680
-6.60111	-379.79057
-4.18404	-113.25902
-2.54864	-34.54609
-1.38474	-11.49029
-0.36714	-5.31907
2.47283	-2.10868
2.72228	0.35268
2.69119	0.00593
2.69065	0.00000
2.69065	0.00000



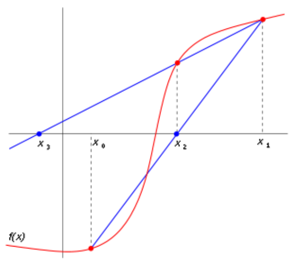
Analysis: The newton's method seeks to find the roots of a continuous function by continuously drawing secant lines and iterating through until an estimated root can be obtained. In the first function this occurred much faster than in the second function because a secant line was found much faster as the function did not have a large difference in its derivative. When the second function's derivative was opposite the original function which needed more iterations.

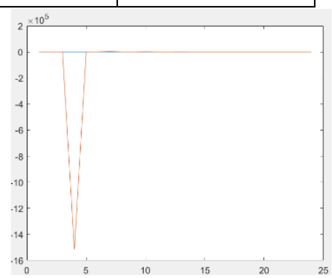
Secant Method

Input function a	x*x*x-5*x*x+2*x
Input initial values	0,1
Input tolerance	0.0001
maximum number of iterations	10000
Approximate solution P	0
F(P)	0
Number of iterations	3

Input function b	x*x*x-2*x*x+5
Input initial values	0,1
Input tolerance	0.0001
maximum number of iterations	10000
Approximate solution P	2.69064745
F(P)	-0.00000001
Number of iterations	25

Function	a
0.000000	0.000000
0.000000	0.000000





Function	b
-5.00000	-180.00000
1.20690	-6.15523
1.42666	-6.16695
-114.22837	-1516566.75
1.42713	-6.16676
1.42760	-6.16657
16.74223	4127.27730
1.45045	-6.15615
1.47322	-6.14331
12.37177	1582.50973
1.51537	-6.11288
1.55714	-6.07379
8.04861	386.82905
1.65749	-5.94096
1.75416	-5.75646
4.77027	58.03876
2.02632	-4.89194
2.23962	-3.79809
2.98025	3.70653
2.61445	-0.79997
2.67939	-0.12257
2.69114	0.00539
2.69064	-0.00003
2.69065	0.00000

Analysis: The secant method finds roots by drawing a linear line from the secant line it creates with the function. With the first function it is very easy to find the root of 0 as both the function and its derivative pass through the origin, but the second function takes much more time as the secant line does not easily draw towards the origin with opposite functions.

BiSection Method

```
syms('OK','A','B','X','FA','FB','TOL','NO','FLAG','NAME','OUP','I')
syms('C','P','FP','x','s')
TRUE = 1;
FALSE = 0;
fprintf(1,'This is the Bisection Method.\n');
fprintf(1,'Input the function F(x) in terms of x\n');
fprintf(1,'For example: cos(x)\n ');
s = input(' ');
F = inline(s,'x');
OK = FALSE;
while OK == FALSE
fprintf(1,'Input endpoints A < B on separate lines\n');
A = input(' ');
B = input(' ');
if A > B
X = A;
A = B;
B = X;
end
if A == B
fprintf(1,'A cannot equal B\n');
else
FA = F(A);
FB = F(B);
if FA*FB > 0
fprintf(1,'F(A) and F(B) have same sign\n');
else
OK = TRUE;
end
```

Muller's Method

```
syms('P', 'OK', 'TOL', 'M', 'X', 'FLAG', 'NAME', 'OUP', 'F', 'H');
syms('r','DEL1', 'DEL', 'I', 'B', 'D', 'E', 'J','x','s','N');
TRUE = 1;
FALSE = 0;
F = zeros(1,4);
X = zeros(1,4);
H = zeros(1,3);
DEL1 = zeros(1,2);
fprintf(1,'This is Mullers Method.\n');
fprintf(1,'Input the Polynomial P(x)\n');
fprintf(1,'For example: to input x^3-2*x+4 enter \n');
fprintf(1,' [ 1 0 -2 4 ] \n');
P = input(' ');
OK = TRUE;
N = length(P);
if N == 2
r = -P(N)/P(N-1);
fprintf(1,'Polynomial is linear: root is %11.8f\n', r);
OK = FALSE;
end
if OK == TRUE
OK = FALSE;
while OK == FALSE
fprintf(1,'Input tolerance\n');
TOL = input(' ');
if TOL <= 0
fprintf(1,'Tolerance must be positive\n');
else
OK = TRUE;
```

Newton's

```
syms('OK', 'P0', 'TOL', 'NO', 'FLAG', 'NAME', 'OUP', 'F0');
syms('I', 'FP0', 'D','x','s');
TRUE = 1;
FALSE = 0;
fprintf(1,'This is Newtons Method\n');
fprintf(1,'Input the function F(x) in terms of x\n');
fprintf(1,'For example: cos(x)\n');
s = input(' ');
F = inline(s,'x');
fprintf(1,'Input the derivative of F(x) in terms of x\n');
s = input(' ');
FP = inline(s,'x');
OK = FALSE;
fprintf(1,'Input initial approximation\n');
P0 = input(' ');
while OK == FALSE
fprintf(1,'Input tolerance\n');
TOL = input(' ');
if TOL <= 0
fprintf(1,'Tolerance must be positive\n');
else
OK = TRUE;
end
end
OK = FALSE;
while OK == FALSE
fprintf(1,'Input maximum number of iterations - no decimal
point\n');
NO = input(' ');
```

Secant

```
syms('OK', 'P0', 'P1', 'TOL', 'NO', 'FLAG', 'NAME', 'OUP',
'F0');
syms('I', 'F1', 'P', 'FP','s','x');
TRUE = 1;
FALSE = 0;
fprintf(1,'This is the Secant Method\n');
fprintf(1,'Input the function F(x) in terms of x\n');
fprintf(1,'For example: cos(x)\n');
s = input(' ');
F = inline(s,'x');
OK = FALSE;
while OK == FALSE
fprintf(1,'Input initial approximations P0 and P1 on
separate lines.\n');
P0 = input(' ');
P1 = input(' ');
if P0 == P1
fprintf(1,'P0 cannot equal P1\n');
else
OK = TRUE;
end
end
OK = FALSE;
while OK == FALSE
fprintf(1,'Input tolerance\n');
TOL = input(' ');
if TOL <= 0
fprintf(1,'Tolerance must be positive\n');
else
```

```
end
end
OK = FALSE;
while OK == FALSE
fprintf(1,'Input tolerance\n');
TOL = input(' ');
if TOL <= 0
fprintf(1,'Tolerance must be positive\n');
else
OK = TRUE;
end
end
OK = FALSE;
while OK == FALSE
fprintf(1,'Input maximum number of iterations - no decimal point\n');
NO = input(' ');
if NO <= 0
fprintf(1,'Must be positive integer\n');
else
OK = TRUE;
end
end
if OK == TRUE
fprintf(1,'Select output destination\n');
fprintf(1,'1. Screen\n');
fprintf(1,'2. Text file\n');
fprintf(1,'Enter 1 or 2\n');
FLAG = input(' ');
if FLAG == 2
fprintf(1,'Input the file name in the form - drive:\\name.ext\n');
fprintf(1,'For example:   A:\\OUTPUT.DTA\n');
NAME = input(' ','s');
OUP = fopen(NAME,'wt');
else
OUP = 1;
end
fprintf(1,'Select amount of output\n');
fprintf(1,'1. Answer only\n');
fprintf(1,'2. All intermediate approximations\n');
fprintf(1,'Enter 1 or 2\n');
FLAG = input(' ');
fprintf(OUP,'Bisection Method\n');
if FLAG == 2
fprintf(OUP, '  I    P                F(P)\n');
end
I = 1;
OK = TRUE;
while I <= NO & OK == TRUE
C = (B - A) / 2.0;
P = A + C;
FP = F(P);
if FLAG == 2
fprintf(OUP,'%3d   %15.8e   %15.7e \n',I,P,FP);
end
if abs(FP) < 1.0e-20 | C < TOL
fprintf(OUP,'\nApproximate solution P = %11.8f \n',P);
fprintf(OUP,'with F(P) = %12.8f\n',FP);
fprintf(OUP,'Number of iterations = %3d',I);
fprintf(OUP,' Tolerance = %15.8e\n',TOL);
OK = FALSE;
else
I = I+1;
if FA*FP > 0
A = P;
FA = FP;
else
B = P;
FB = FP;
end
end
end
if OK == TRUE
fprintf(OUP,'\nIteration number %3d',NO);
fprintf(OUP,' gave approximation %12.8f\n',P);
fprintf(OUP,'F(P) = %12.8f not within tolerance : %15.8e\n',FP,TOL);
end
if OUP ~= 1
fclose(OUP);
fprintf(1,'Output file %s created successfully \n',NAME);
end
end
```

```
end
end
OK = FALSE;
while OK == FALSE
fprintf(1,'Input maximum number of iterations - no decimal point\n');
M = input(' ');
if M <= 0
fprintf(1,'Must be positive integer\n');
else
OK = TRUE;
end
end
fprintf(1,'Input the first of three starting values\n');
X(1) = input(' ');
fprintf(1,'Input the second of three starting values\n');
X(2) = input(' ');
fprintf(1,'Input the third starting value\n');
X(3) = input(' ');
end
if OK == TRUE
fprintf(1,'Select output destination\n');
fprintf(1,'1. Screen\n');
fprintf(1,'2. Text file\n');
fprintf(1,'Enter 1 or 2\n');
FLAG = input(' ');
if FLAG == 2
fprintf(1,'Input the file name in the form - drive:\\name.ext\n');
fprintf(1,'For example:   A:\\OUTPUT.DTA\n');
NAME = input(' ','s');
OUP = fopen(NAME,'wt');
else
OUP = 1;
end
fprintf(OUP, 'MULLERS METHOD\n');
fprintf(OUP, 'The output is i, approximation x(i), f(x(i))\n\n');
fprintf(OUP,'The real and imaginary parts of x(i) are\n');
fprintf(OUP,'followed by real and imaginary parts of f(x(i)).\n\n');
F(1) = polyval(P,X(1));
F(2) = polyval(P,X(2));
F(3) = polyval(P,X(3));
H(1) = X(2)-X(1);
H(2) = X(3)-X(2);
DEL1(1) = (F(2)-F(1))/H(1);
DEL1(2) = (F(3)-F(2))/H(2);
DEL = (DEL1(2)-DEL1(1))/(H(2)+H(1));
I = 3;
while I <= M & OK == TRUE
B = DEL1(2)+H(2)*DEL;
D = B*B-4*F(3)*DEL;
if abs(DEL) <= 1.0e-20
if abs(DEL1(2)) <= 1.0e-20
fprintf(1,'Horizontal Line\n');
OK = FALSE;
else
X(4) = (F(3)-DEL1(2)*X(3))/DEL1(2);
H(3) = X(4)-X(3);
end
else
D = sqrt(D);
E = B+D;
if abs(B-D) > abs(E)
E = B-D;
end
H(3) = -2*F(3)/E;
X(4) = X(3)+H(3);
end
if OK == TRUE
F(4) = polyval(P,X(4));
fprintf(OUP, '%d %f %f %f %f\n',I,X(4),imag(X(4)),F(4),imag(F(4)));
end
if abs(H(3)) < TOL
fprintf(OUP, '\nMethod Succeeds\n');
fprintf(OUP, 'Approximation is within %.10e\n', TOL);
fprintf(OUP, 'in %d iterations\n', I);
OK = FALSE;
else
for J = 1:2
H(J) = H(J+1);
X(J) = X(J+1);
F(J) = F(J+1);
end
X(3) = X(4);
F(3) = F(4);
DEL1(1) = DEL1(2);
DEL1(2) = (F(3)-F(2))/H(2);
DEL = (DEL1(2)-DEL1(1))/(H(2)+H(1));
end
I = I+1;
end
if I > M & OK == TRUE
fprintf (OUP, 'Method Failed\n');
end
if OUP ~= 1
fclose(OUP);
fprintf(1,'Output file %s created sucessfully\n',NAME);
end
end
```

```
if NO <= 0
fprintf(1,'Must be positive integer\n');
else
OK = TRUE;
end
end
if OK == TRUE
fprintf(1,'Select output destination\n');
fprintf(1,'1. Screen\n');
fprintf(1,'2. Text file\n');
fprintf(1,'Enter 1 or 2\n');
FLAG = input(' ');
if FLAG == 2
fprintf(1,'Input the file name in the form - drive:\\name.ext\n');
fprintf(1,'For example:   A:\\OUTPUT.DTA\n');
NAME = input(' ','s');
OUP = fopen(NAME,'wt');
else
OUP = 1;
end
fprintf(1,'Select amount of output\n');
fprintf(1,'1. Answer only\n');
fprintf(1,'2. All intermediate approximations\n');
fprintf(1,'Enter 1 or 2\n');
FLAG = input(' ');
fprintf(OUP, 'Newtons Method\n');
if FLAG == 2
fprintf(OUP, '  I    P                F(P)\n');
end
F0 = F(P0);
% STEP 1
I = 1;
OK = TRUE;
% STEP 2
while I <= NO & OK == TRUE
% STEP 3
% compute P(I)
FP0 = FP(P0);
D = F0/FP0;
% STEP 6
P0 = P0 - D;
F0 = F(P0);
if FLAG == 2
fprintf(OUP,'%3d   %14.8e   %14.7e\n',I,P0,F0);
end
% STEP 4
if abs(D) < TOL
% procedure completed successfully
fprintf(OUP,'\nApproximate solution = %.10e\n',P0);
fprintf(OUP,'with F(P) = %.10e\n',F0);
fprintf(OUP,'Number of iterations = %d\n',I);
fprintf(OUP,'Tolerance = %.10e\n',TOL);
OK = FALSE;
% STEP 5
else
I = I+1;
end
end
if OK == TRUE
% STEP 7
% procedure completed unsuccessfully
fprintf(OUP,'\nIteration number %d',NO);
fprintf(OUP,' gave approximation %.10e\n',P0);
fprintf(OUP,'with F(P) = %.10e not within tolerance %.10e\n',F0,TOL);
end
if OUP ~= 1
fclose(OUP);
fprintf(1,'Output file %s created successfully \n',NAME);
end
end
```

```
OK = TRUE;
end
end
OK = FALSE;
while OK == FALSE
fprintf(1,'Input maximum number of iterations - no decimal point\n');
NO = input(' ');
if NO <= 0
fprintf(1,'Must be positive integer\n');
else
OK = TRUE;
end
end
if OK == TRUE
fprintf(1,'Select output destination\n');
fprintf(1,'1. Screen\n');
fprintf(1,'2. Text file\n');
fprintf(1,'Enter 1 or 2\n');
FLAG = input(' ');
if FLAG == 2
fprintf(1,'Input the file name in the form - drive:\\name.ext\n');
fprintf(1,'For example:   A:\\OUTPUT.DTA\n');
NAME = input(' ','s');
OUP = fopen(NAME,'wt');
else
OUP = 1;
end
fprintf(1,'Select amount of output\n');
fprintf(1,'1. Answer only\n');
fprintf(1,'2. All intermediate approximations\n');
fprintf(1,'Enter 1 or 2\n');
FLAG = input(' ');
fprintf(OUP, 'Secant Method\n');
if FLAG == 2
fprintf(OUP, '  I    P                F(P)\n');
end
% STEP 1
I = 2;
F0 = F(P0);
F1 = F(P1);
OK = TRUE;
% STEP 2
while I <= NO & OK == TRUE
% STEP 3
% compute P(I)
P = P1-F1*(P1-P0)/(F1-F0);
% STEP 4
FP = F(P);
if FLAG == 2
fprintf(OUP,'%3d   %15.8e   %15.8e\n',I,P,FP);
end
% STEP 4
if abs(P-P1) < TOL
% procedure completed successfully
fprintf(OUP,'\nApproximate solution P = %12.8f\n',P);
fprintf(OUP,'with F(P) = %12.8f\n',FP);
fprintf(OUP,'Number of iterations = %d\n',I);
fprintf(OUP,'Tolerance = %14.8e\n',TOL);
OK = FALSE;
% STEP 5
else
I = I+1;
% STEP 6
% update P0, F0, P1, F1
P0 = P1;
F0 = F1;
P1 = P;
F1 = FP;
end
end
if OK == TRUE
% STEP 7
% procedure completed unsuccessfully
fprintf(OUP,'\nIteration number %d',NO);
fprintf(OUP,' gave approximation %12.8f\n',P);
fprintf(OUP,'with F(P) = %12.8f not within tolerance %15.8e\n',FP,TOL);
end
if OUP ~= 1
fclose(OUP);
fprintf(1,'Output file %s created successfully\n',NAME);
end
end
```