2.3 The Newton-Raphson Method

**Question 1**

Using newtons method with initial guess P0 = 3

f = function(x) { x^3 - 2\*x^2 -5 }

fp = function(x) { 3\*x^2-4\*x }

> newtons (3, f, fp)

[1] 2.690647

> newtons (3, f, fp, iter = true)

p\_n abs (P - P0)

0 3.000000 NA

1 2.733333 2.666667e-01

2 2.691625 4.170861e-02

3 2.690648 9.767488e-04

4 2.690647 5.289012e-07

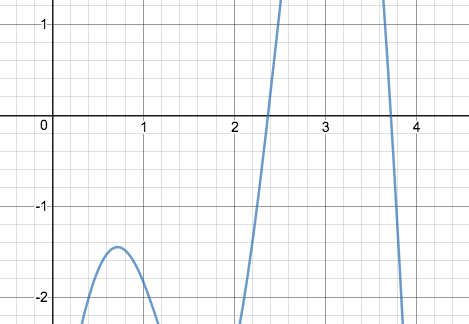
5 2.690647 1.553235e-13

The solution accurate to within 10-4 is P3 = 2.690648

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ----------------------\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Question 2**

Graph the function, we get



f = function(x) { 2\*x\*cos(2\*x) - (x-2)^2 }

fp = function(x) { 2\*cos(2\*x) - 4\*x \*sin(2\*x) - 2\*(x-2) }

using initial guess 2

> newtons (2, f, fp, iter = true)

P\_value abs (P - P0)

0 2.000000 NA

1 2.550769 5.507692e-01

2 2.371359 1.794107e-01

3 2.370687 6.714557e-04

4 2.370687 1.391643e-07

5 2.370687 5.773160e-15

using initial guess 3.5

> newtons (3.5, f, fp, iter = true)

P\_value abs (P - P0)

0 3.500000 NA

1 3.783191 2.831912e-01

2 3.724165 5.902576e-02

3 3.722115 2.049904e-03

4 3.722113 2.724167e-06

5 3.722113 4.825029e-12

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ----------------------\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Question 3**

f = function(x) { x^3 - 2\*x^2 -5 }

Using secant method with initial guess: 2.5, 3

> secant (2.5, 3, f)

[1] 2.690647

> secant (2.5, 3, f, iter=true)

new\_p a\_n b\_n err (b-a)

1 2.659574 2.500000 3.000000 5.000000e-01

2 2.685853 3.000000 2.659574 3.404255e-01

3 2.690731 2.659574 2.685853 2.627898e-02

4 2.690647 2.685853 2.690731 4.877722e-03

5 2.690647 2.690731 2.690647 8.394698e-05

6 2.690647 2.690647 2.690647 2.228408e-07

Using false position method w/ initial guess: 2.5, 3

> falseposition(2.5, 3, f)

[1] 2.690647

> falseposition(2.5, 3, f, iter = true)

new\_p [p0 p1] err (p-p1)

1 2.659574 2.5 3.000000 3.404255e-01

2 2.685853 3.0 2.659574 2.627898e-02

3 2.689914 3.0 2.685853 4.060910e-03

4 2.690535 3.0 2.689914 6.211384e-04

5 2.690630 3.0 2.690535 9.485675e-05

6 2.690645 3.0 2.690630 1.448249e-05

7 2.690647 3.0 2.690645 2.211070e-06

8 2.690647 3.0 2.690647 3.375663e-07

9 2.690647 3.0 2.690647 5.153656e-08

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ----------------------\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Question 4**

f = function(x) { 2\*x\*cos(2\*x) - (x-2)^2 }

Using secant method with initial guess: 2, 3.5

> secant (2, 3.5, f)

[1] 2.370687

> secant (2, 3.5, f, iter = true)

new\_p a\_n b\_n err (b-a)

1 2.695133 2.000000 3.500000 1.500000000

2 -15.232432 3.500000 2.695133 0.804867422

3 2.531565 2.695133 -15.232432 17.927564832

4 2.449682 -15.232432 2.531565 17.763997697

5 2.372182 2.531565 2.449682 0.081883213

6 2.370716 2.449682 2.372182 0.077500689

7 2.370687 2.372182 2.370716 0.001465807

8 2.370687 2.370716 2.370687 0.000028803

Using false position method w/ initial guess: 2.5, 3

> falseposition (2, 3.5, f)

[1] 2.370687

> falseposition(2, 3.5, f, iter = true)

new\_p [p0 p1] err (p-p1)

1 2.695133 2.000000 3.500000 8.048674e-01

2 2.329742 2.000000 2.695133 3.653909e-01

3 2.369639 2.695133 2.329742 3.989715e-02

4 2.370671 2.695133 2.369639 1.032006e-03

5 2.370687 2.695133 2.370671 1.582532e-05

6 2.370687 2.695133 2.370687 2.383754e-07

7 2.370687 2.695133 2.370687 3.589633e-09

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ----------------------\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Question 6**

Use Newton’s method to approximate, within 10-4, the value of x that produces the point on the graph of that closest to (2,1)

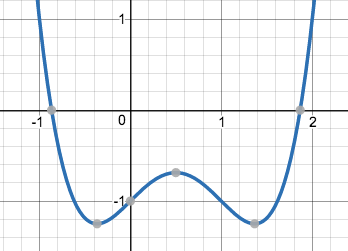
First, we need to find the minimum distance between the curve and the point (2, 1)

The distance formula:

Taking the derivatives implicitly yields

When it means that the numerator

Here is the graph of function



Using Newton’s method with initial guess of 2

f = function(x) { x^4 - 2\*x^3 - 1 + x }

fp = function(x) { 4\*x^3 - 6\*x^2 +1 }

> newtons (2, f, fp, iter = true)

P\_value abs (P - P0)

0 2.000000 NA

1 1.888889 1.111111e-01

2 1.867504 2.138453e-02

3 1.866761 7.430861e-04

4 1.866760 8.777535e-07

5 1.866760 1.223688e-12

So

Thus, the point on the graph of that closest to (2, 1) is the point (

So, the minimum distance between the curve and the point (2, 1) is

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ----------------------\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Question 10**

The function

1. Bisection method

f10 = function(x) {tan(pi\*x) -6}

bisection (0, 0.48, f10)

[1] 0.4474315

Bisection (0, 0.48, f10, n=10, iter = true)

Midpt (p) LeftB (a) RightB (b) err (b-a)

1 0.2400000 0.0000000 0.480000 0.24000000

2 0.3600000 0.2400000 0.480000 0.12000000

3 0.4200000 0.3600000 0.480000 0.06000000

4 0.4500000 0.4200000 0.480000 0.03000000

5 0.4350000 0.4200000 0.450000 0.01500000

6 0.4425000 0.4350000 0.450000 0.00750000

7 0.4462500 0.4425000 0.450000 0.00375000

8 0.4481250 0.4462500 0.450000 0.00187500

9 0.4471875 0.4462500 0.448125 0.00093750

10 0.4476562 0.4471875 0.448125 0.00046875

1. False Position method

falseposition (0, 0.48, f10)

[1] 0.4474314

falseposition (0, 0.48, f10, n=10, iter = true)

new\_p [p0 p1] err (p-p1)

1 0.1811942 0.00 0.4800000 0.298805758

2 0.2861872 0.48 0.1811942 0.104992924

3 0.3489812 0.48 0.2861872 0.062794062

4 0.3870526 0.48 0.3489812 0.038071394

5 0.4103047 0.48 0.3870526 0.023252099

6 0.4245665 0.48 0.4103047 0.014261763

7 0.4333363 0.48 0.4245665 0.008769830

8 0.4387374 0.48 0.4333363 0.005401096

9 0.4420669 0.48 0.4387374 0.003329540

10 0.4441207 0.48 0.4420669 0.002053713

1. Secant method

secant (0, 0.48, f10)

[1] 8.720436e+16

Warning message:

In secant (0, 0.48, f10) : Failed to converge after 50 iterations

> secant (0, 0.48, f10, n=10, iter= true)

current\_p p\_0 p\_1. err (p0-p1)

1 0.1811942 0.0000000 0.4800000 0.4800000

2 0.2861872 0.4800000 0.1811942 0.2988058

3 1.0919861 0.1811942 0.2861872 0.1049929

4 -3.6922967 0.2861872 1.0919861 0.8057989

5 -22.6006499 1.0919861 -3.6922967 4.7842828

6 -57.2228325 -3.6922967 -22.6006499 18.9083532

7 3.5387581 -22.6006499 -57.2228325 34.6221826

8 -113.9443980 -57.2228325 3.5387581 60.7615906

9 -195.8944309 3.5387581 -113.9443980 117.4831562

10 -2956.3767304 -113.9443980 -195.8944309 81.950032

Clearly, we can see that the bisection is the best to solve this problem.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ----------------------\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Question 11**

**The equation for secant method can also be written in the simpler form**

**Explain why this iteration equation is likely to be less accurate than the one given in Algorithm 2.4**

In general, it increases the probability of truncation error. It involves the subtraction of nearly equal entities on both the top and the bottom, which causes floating point issues.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ----------------------\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Question 16**

Find an approximation for , accurate to within 10-4, for the population equation

Use this value to predict the population at the end of the second year, assuming that the immigration rate during this year remains at 435,000 individuals per year.

Simplify the equation by Divide both sides of the equation by a million,

We can use secant method solve for with initial guess 1 and 2

f = function(x) {1.564 - exp(x) - (0.435/x) \*(exp(x)-1)}

> secant (1, 2, f, iter = true)

current\_p p\_0 p\_1 err (p0-p1)

1 0.6420562 1.0000000 2.0000000 1.000000e+00

2 0.4370293 2.0000000 0.6420562 1.357944e+00

3 0.1764536 0.6420562 0.4370293 2.050269e-01

4 0.1122397 0.4370293 0.1764536 2.605757e-01

5 0.1013928 0.1764536 0.1122397 6.421390e-02

6 0.1010000 0.1122397 0.1013928 1.084689e-02

7 0.1009979 0.1013928 0.1010000 3.927478e-04

8 0.1009979 0.1010000 0.1009979 2.088901e-06

So

The population equation (in million) is

The population at the end of the year two should be :

\*