Chapter 5: Non-linear equations

Joseph Sepich

Contents

1 Problem 1 Bisection Method

We know from our convergence analysis of the Bisection Method that the error tolerance is bounded by the following equation:

$$\frac{b_0 - a_0}{2^{n+1}} \leq \epsilon$$

And we can solve this equation to get:

$$n \geq \frac{ln(b-a) - ln(2\epsilon)}{ln2}$$

So to solve our problem for a = 0, b = 1, and $\epsilon = 10^{-8}$:

$$n \geq \frac{ln(1-0) - ln(2 * 10^{-8})}{ln2}$$
$$n \geq \frac{17.7275}{0.6931}$$
$$n \geq 25.5754$$

So to get our error tolerance of 10^{-8} using the bisection method we need at least 26 steps.

2 Problem 2 Fixed Point Iteration

We are trying to find the root of $f(x) = e^{-x} - cos(x)$.

2.1 a. First show there is a root in [1.1, 1.6]

$$f(1.1) = e^{-1.1} - cos(1.1) = -0.1207$$
$$f(1.6) = e^{-1.6} - cos(1.6) = 0.2311$$

Since f(1.1) and f(1.6) are opposite signs, and f is a continuous function, it must pass through 0 between x = 1.1 and 1.6.

2.2 b. Locate root by fixed point iteration

Start at $x_0 = 1.6$ and perform 4 iterations to compute x_1, x_2, x_3, x_4. What is observed and does it converge?

$$x_1 = g_1(x_0) = e^{-1.6} - cos(1.6) + 1.6 = 1.8311$$
$$x_2 = g_1(x_1) = e^{-1.8311} - cos(1.8311) + 1.8311 = 1.7340$$
$$x_3 = g_1(x_2) = e^{-1.7340} - cos(1.7340) + 1.7340 = 1.7481$$
$$x_4 = g_1(x_3) = e^{-1.7481} - cos(1.7481) + 1.7481 = 1.7458$$

The numbers appear to be converging to a root, that is not contained within our original interval. We can verify if it is converging by looking at the absolute value of the derivative of $g_1(x)$. $g_1'(x) = -sin(x) - e^{-x} + 1$, which at x = 1.6 is $g_1'(x) = -sin(1.6) - e^{-1.6} + 1 = -0.2015$, which is $|g_1'(x_0)| < 1$, so it must definitely converge.

2.3 c. Fixed point iteration with equation 2

Now use $g_2(x) = x - f(x)$ to do fixed point iteration for 4 iterations starting at $x_0 = 1.6$. How does this differ from part b?

$$x_1 = g_2(x_0) = -(e^{-1.6} - cos(1.6)) + 1.6 = 1.3689$$
$$x_2 = g_2(x_1) = -(e^{-1.3689} - cos(1.3689)) + 1.3689 = 1.3150$$
$$x_3 = g_2(x_2) = -(e^{-1.3150} - cos(1.3150)) + 1.3150 = 1.2995$$
$$x_4 = g_2(x_3) = -(e^{-1.2995} - cos(1.2995)) + 1.2995 = 1.2948$$

This time you can see that the fixed point iteration is heading towards the root to the left of the initial point. The first equation we used for fixed point iteration went to the root on the right of the initial point.

3 Problem 3 more on fixed point iteration

Consider the iteration scheme:

$$x_{n+1} = \frac{1}{2}x_n + \frac{1}{x_n}, x_0 = 1$$

3.1 a. If converges, what is lim as n goes to inf

Let's just write out the first few terms starting at $x_0 = 1$:

$$1, 1.5, 1.4167, 1.4142, 0.9336, 1.5379$$

From this pattern you can see that this function oscillates. Plugging in the limit to Matlab you get that it converges on 1.4142, which when you plug this number into the equation you can confirm it converges here.

We could also look to see what the original function here was as this is an iteration scheme.

$$x_{n+1} = \frac{1}{2}x_n + \frac{1}{x_n}$$
$$x_{n+1} = x_n - (\frac{x_n^2}{2x_n} - \frac{2}{2x_n})$$
$$\frac{f(x_n)}{f'(x_n)} = \frac{x_n^2 - 2}{2x_n}$$

Meaning $f(x) = x^2 - 2$, which we know has the roots +/- 1.4142, which we got with our Matlab iteration.

3.2 b. Does the method converge? Why?

In this instance our function g(x) is $\frac{1}{2}x_n + \frac{1}{x_n}$. We know from our convergence analysis that if the condition $|g'(x)| < 1$, then the function converges.

$$g'(x) = \frac{1}{2} - \frac{1}{x^2}, g'(x_0) = g'(1) = \frac{1}{2} - 1 = -\frac{1}{2}$$

So it is clear that the condition is met and the iteration scheme converges.

4 Problem 4 Newton's Method

4.1 a. Check the iteration scheme

Check that Newton's iteration scheme is $x_{n+1} = \frac{1}{2}(x_n + \frac{R}{x_n})$ for $f(x) = x^2 - R$

Let's take it back to the general form for Newton's iteration formula:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

With our function f(x), we know $f'(x) = 2x$, so we get:

$$x_{n+1} = x_n - \frac{x_n^2 - R}{2x_n} = x_n - \frac{x^n}{2} + \frac{R}{2x_n} = \frac{1}{2}(x_n + \frac{R}{x_n})$$

This corresponds with the given iteration scheme.

4.2 b. Show that the sequence satisfies:

$$x_{n+1}^2 - R = [\frac{x_n^2 - R}{2x_n}]^2$$

The sequence here shows the concept of quadratic convergece, because it demonstrates the eror function of the actual root. And the error gets smaller by a power of two each iteration through the fixed point iteration.

4.3 c. Now we test R = 10

Choose $x_0 = 3$ and do 4 iterations. Use 8 digits.

$$x_1 = g(x_0) = \frac{1}{2}(3 + \frac{10}{3}) = 3.16666667$$
$$x_2 = g(x_1) = \frac{1}{2}(3.1666667 + \frac{10}{3.16666667}) = 3.16228070$$
$$x_3 = g(x_2) = \frac{1}{2}(3.16228070 + \frac{10}{3.16228070}) = 3.16227766$$
$$x_4 = g(x_3) = \frac{1}{2}(3.16227766 + \frac{10}{3.16227766}) = 3.16227766$$

This result converged very fast. We got eight digits of precision with just 3 calculations.

5 Problem 5 Newton's method not always good

5.1 i. m is a postive integer

Let's consider the function:

$$f(x) = (x - 1)^m$$

It is clear that in this instance r = 1 is a root. Let's try to find this by Newton's iteration for m = 8.

5.1.1 a. Set up the iteration scheme

We have the function:

$$f(x) = (x-1)^8$$
$$f'(x) = 8(x-1)^7$$

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$
$$x_{n+1} = x_n - \frac{(x_n-1)^8}{8(x_n-1)^7} = x_n - \frac{x_n - 1}{8} = \frac{7}{8}x_n + \frac{1}{8}$$

So the iteration scheme is:

$$x_{n+1} = \frac{7}{8}x_n + \frac{1}{8}$$

5.1.2 b.

Use $x_0 = 1.1$ as the initial guess. Complete 4 iterations.

$$x_1 = g(x_0) = \frac{7}{8}(1.1) + \frac{1}{8} = 1.0875$$
$$x_2 = g(x_1) = \frac{7}{8}(1.0875) + \frac{1}{8} = 1.0765$$
$$x_3 = g(x_2) = \frac{7}{8}(1.0765) + \frac{1}{8} = 1.06699$$
$$x_4 = g(x_3) = \frac{7}{8}(1.06699) + \frac{1}{8} = 1.05862$$

5.1.3 c.

How does the method work for this problem? Do we still have quadratic convergence? What is causing this?

We most definitely do not have quadratic convergence. We went from an error of 1 * 10 $^{-1}$ to an error of 6 * 10 $^{-2}$ in 4 steps. You would expect it to be around 10^{-8} in four iterations with quadratic convergence. The iteration scheme here does not allow for quick convergence. Since we are keeping 7/8 of the original value over each iteration, it does not converge quickly.

5.1.4 d.

What if m is 20? This goes back to what I mentioned in the previous part. If m is 20, you are keeping 19/20 of the previous value each time, so the convergence will be even slower. The greater m is, the slower the converge is with this method.

5.2 ii. Use Newton's method to solve the equation

$$0 = \frac{1}{2} + \frac{1}{4}x^2 - xsin(x) - \frac{1}{2}cos(2x), x_0 = \frac{\pi}{2}$$

Iterate until error is within 3 digits.

$$f'(x) = sin(2x) - sin(x) - xcos(x) + \frac{1}{2}x$$

$$x_{n+1} = x_n - \frac{\frac{1}{2} + \frac{1}{4}x^2 - xsin(x) - \frac{1}{2}cos(2x)}{sin(2x) - sin(x) - xcos(x) + \frac{1}{2}x}$$

$$x_1 = g(x_0) = 1.785398$$
$$x_2 = g(x_1) = 1.844561$$
$$x_3 = g(x_2) = 1.870834$$
$$x_4 = g(x_3) = 1.883346$$
$$x_5 = g(x_4) = 1.889464$$

Until we get 1.895493. This is unusally slow convergence for Newton's method.

6 Problem 6 Newton in Matlab

Newton Function:

```
function r = mynewton(f,df,x0,tol,nmax)
%MYNEWTON matlab implementation of newton fixed point iteration
% f - function f(x)
% df - function f'(x)
% x0 - initial guess
% tol - error tolerance
% nmax - maximum number of iterations
x = x0;
n = 0;
dx = feval(f,x)/feval(df,x);
while ((dx>tol) & (feval(f,x)>tol)) | (n<nmax)
    fVal = feval(f,x);
    disp(sprintf('I have n=%d and x=%g, but f=%f.\n',n,x,fVal))
    n = n+1;
    x = x-dx;
    dx = feval(f,x)/feval(df,x);
end
r = x-dx;
```

Script:

```
% Calculate square root of 2
mynewton('func','f',1.4,10^-8,10)

% Root of e^-x-cos(x) on [1.1, 1.6]
% tol = 1e-12
% nmax = 10
mynewton('func2', 'f2',1.3,1e-12,10)
```

Output:

```
problem6Script
I have n=0 and x=1.4, but f=-0.040000.
I have n=1 and x=1.41429, but f=0.000204.
I have n=2 and x=1.41421, but f=0.000000.
I have n=3 and x=1.41421, but f=0.000000.
I have n=4 and x=1.41421, but f=-0.000000.
I have n=5 and x=1.41421, but f=0.000000.
I have n=6 and x=1.41421, but f=-0.000000.
I have n=7 and x=1.41421, but f=0.000000.
I have n=8 and x=1.41421, but f=-0.000000.
I have n=9 and x=1.41421, but f=0.000000.
ans =

        1.414213562373095

I have n=0 and x=1.3, but f=0.005033.
I have n=1 and x=1.29272, but f=0.000014.
I have n=2 and x=1.2927, but f=0.000000.
I have n=3 and x=1.2927, but f=0.000000.
I have n=4 and x=1.2927, but f=0.000000.
I have n=5 and x=1.2927, but f=0.000000.
I have n=6 and x=1.2927, but f=0.000000.
I have n=7 and x=1.2927, but f=0.000000.
I have n=8 and x=1.2927, but f=0.000000.
I have n=9 and x=1.2927, but f=0.000000.
ans =

        1.292695719373398
```

Functions for part 1:

```
function y = func(x)
%FUNC function
y = x^2 - 2;
end


function y = f(x)
%F derivative of function in func.m
y = 2*x;
end
```

Funcion for part 2:

```
function y = func2(x)
    y = exp(-x)-cos(x);
end


function y = f2(x)
    y = sin(x) - exp(-x);
end
```

7 Problem 7 The Secant Method

7.1 a.

Calculate the approximate value for $4^{3/5}$ using the secant method with $x_0 = 3$ and $x_1 = 2$. Make 3 steps to get to x_4.

Recall with the secant method we follow the scheme:

$$x_{n+1} = x_n - \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})} f(x_n)$$

Using $f(x) = x^{5/3} - 4$

$$x_2 = g(x_1) = 2.26919$$
$$x_3 = g(x_2) = 2.2987$$
$$x_4 = g(x_3) = 2.9739$$

This result appears to converge within the first 3 computations.

7.2 b. Find root

$f(x) = x^3 - 2x + 1, x_0 = 4, x_1 = 2$. Compute the next 3 values.

$$x_2 = g(x_1) = 1.80769$$
$$x_3 = g(x_2) = 1.43714$$
$$x_4 = g(x_2) = 1.25269$$

This method converges to the exact number.

7.3 c. Consider the iteration scheme. What does it converge to?

$$x_{n+1} = x_n + (2 - e^{x_n})(x_n - x_{n-1})/(e^{x_n} - e^{x_{n-1}})$$

You can see that this iteration scheme follows the secant method for the function $f(x) = 2 - e^{x_n}$. Knowing this we know that f(x) is zero when $2 = e^x$, whic gives us the value $ln(2) = 0.69314$. So if the iteration converges the limit would converge to natural log of 2.

8 Problem 8 Secant Method in Matlab

Script:

```
function r = mysecant(f,x0,x1,tol,nmax)
%MYSECANT secant method
xPrev = x0;
x = x1;
n = 0;
dx = ((x - xPrev) / (feval(f,x) - feval(f,xPrev))) * feval(f,x);
while ((dx>tol) & (feval(f,x)>tol)) | (n<nmax)
    fVal = feval(f,x);
    disp(sprintf('I have n=%d and x=%g, but f=%f.\n',n,x,fVal))
    n = n + 1;
    temp = x;
    x = x - dx;
    xPrev = temp;
    dx = ((x - xPrev) / (feval(f,x) - feval(f,xPrev))) * feval(f,x);
end
r = x-dx;
```

Output:

```
mysecant('func',1,2,10^-8,10)

I have n=0 and x=2, but f=2.000000.

I have n=1 and x=1.33333, but f=-0.222222.

I have n=2 and x=1.4, but f=-0.040000.

I have n=3 and x=1.41463, but f=0.001190.

I have n=4 and x=1.41421, but f=-0.000006.

I have n=5 and x=1.41421, but f=-0.000000.

I have n=6 and x=1.41421, but f=0.000000.

I have n=7 and x=1.41421, but f=0.000000.

I have n=8 and x=1.41421, but f=-0.000000.

I have n=9 and x=1.41421, but f=-0.000000.

ans =

1.41421

mysecant('func2',1.6,1.1,1e-12,10)

I have n=0 and x=1.1, but f=-0.120725.
```

```
I have n=1 and x=1.27157, but f=-0.014389.
I have n=2 and x=1.29479, but f=0.001439.
I have n=3 and x=1.29268, but f=-0.000012.
I have n=4 and x=1.2927, but f=-0.000000.
I have n=5 and x=1.2927, but f=0.000000.
I have n=6 and x=1.2927, but f=0.000000.
I have n=7 and x=1.2927, but f=0.000000.
I have n=8 and x=1.2927, but f=0.000000.
I have n=9 and x=1.2927, but f=0.000000.
ans =

    1.2927
```

The secant method does not converge as fast as the Newton method. Here the converge happens around n = 4, but happens around n = 2 with Newton's method. This shows the super linear vs quadaratic convergence in the two methods.