Numerical Algorithms

Introduction

THIS DOCUMENT COVERS

- Introduction

Root Finding

Newton Rhapson

Given a function f we want to find $f^{-1}(c)$ that is to say the value of x such that $f(x) = c$. In simple situations we calculate this analytically using algebra. In many other situations such as finding $\sqrt{c}$ we need to turn to numerical algorithms. A slightly simpler scenario is when we want to find x such that $f(x) = 0$. This is known as root finding. First, we look at how we might use the Taylor to derive a root finding algorithm

The Taylor series tells us that if we know the value of a function at point x_0, say $f(x_0)$then we can use an infinite Taylor series expansion to get the value of $f(x_0 + \delta x)$as follows

$$f(x_0 + \delta x) = f(x_0) + \frac{f^1(x_0)\delta x}{1!} + \frac{f^2(x_0)(\delta x)^2}{2!} \dots$$

A finite Taylor series expansion gives an approximation of the original function. The less terms we use the greater the error in the approximation. A first order approximation is given by

$$f(x_0 + \delta x) \approx f(x_0) + f^1(x_0)\delta x$$

In our case we want to find δx so we re-arrange to get

$$\delta x \approx \frac{f(x_0 + \delta x) - f(x_0)}{f^1(x_0)}$$

Letting $f(x_0 + \delta x) = 0$ such that we are finding the value of $f(x_0 + \delta x)$ that cuts the x-axis.

$$\delta x \approx \frac{-f(x_0)}{f^1(x_0)}$$

Letting $\delta x = (x_1 - x_0)$

$$(x_1 - x_0) \approx \frac{-f(x_0)}{f^1(x_0)}$$

And

$$x_1 \approx x_0 + \frac{-f(x_0)}{f^1(x_0)}$$

Now, $x_1$is an approximation of the root of $f$and the error is given by $f(x_1) - 0 = f(x_1)$

It can be shown that under certain circumstances if we let

$$x_n = x_{n-1} + \frac{-f(x_0)}{f^1(x_0)}$$

Then the sequence x_1, x_2, x_3 ... converges to the root x The following simple algorithm can be used to find roots using this procedure.

Listing 1Root Finding Newton

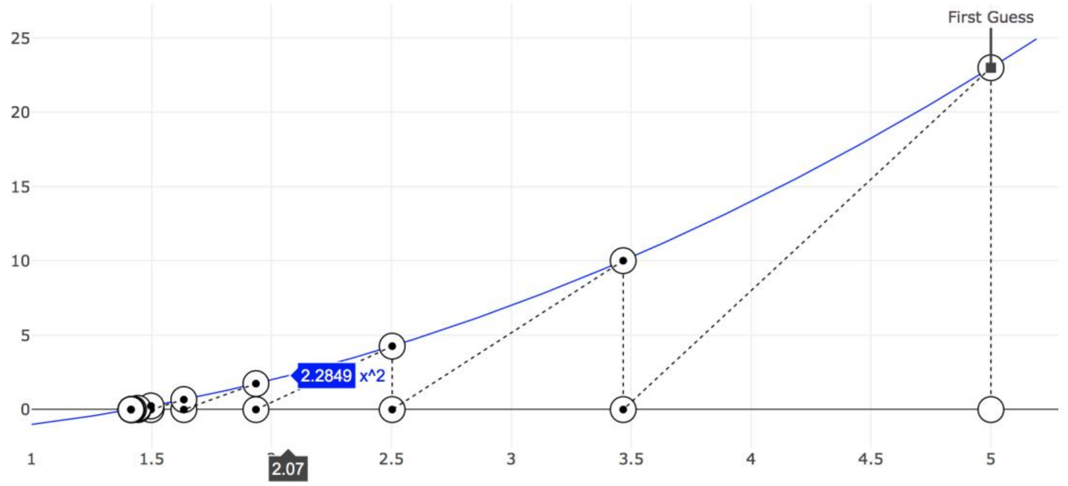
```
public double Solve(Func<double,double>f, Func<double,double> fd,
    double initialGuess)
{
    double x = initialGuess;
    double y = f(x);
    double error = 0.0000001;

    while (Math.Abs(y) > error)
    {
          x = x - (y/fd(x));
          y = f(x);
    }

    return x;
}
```

While finding roots is of course useful our original desire was to find the value of x such that $f(x) = c$ for any value of c. Thankfully a small amendment to our algorithm extends it for any c. If we use original function f to form a new function $g = f - c$ then finding a root of g will give us the value of x that makes $f(x_0) = c$. We can use the derivative of f as the derivative of g because c is a constant and hence gets dropped during differentiation.

```
public double Solve(Func<double,double>f, Func<double,double> fd,
    double initialGuess, double c)
{
    Func<double,double> g = r => f(r)-c;
    double x = initialGuess;

    double y = g(x);
    double error = 0.0000001;

    while (Math.Abs(y) > error)
    {
          x = x - (y/fd(x));
          y = g(x);
    }

    return x;
}
```

The converges looks as follows in tabular and graphical form

```
5.0000,
3.4667,
2.5034,
1.9352,
1.6347,
1.4976,
1.4436,
1.4242,
1.4176,
1.4153,
1.4146,
1.4143,
1.4143,
1.4142,
```



Linear Regression

Beta of Portfolio