Numerical Algorithms

## Introduction

This document covers

Introduction

## Root Finding

### Newton Rhapson

Given a function we want to find that is to say the value of such that . In simple situations we calculate this analytically using algebra. In many other situations such as finding we need to turn to numerical algorithms. A slightly simpler scenario is when we want to find such that . 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 , say then we can use an infinite Taylor series expansion to get the value of as follows

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

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

Letting such that we are finding the value of that cuts the x-axis.

Letting

And

Now, is an approximation of the root of and the error is given by

It can be shown that under certain circumstances if we let

Then the sequence converges to the root The following simple algorithm can be used to find roots using this procedure.

Listing Root 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 such that for any value of c. Thankfully a small amendment to our algorithm extends it for any c. If we use original function to form a new function then finding a root of will give us the value of x that makes . We can use the derivative of as the derivative of 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));

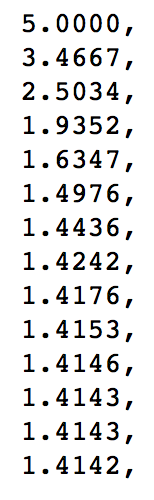
y = g(x);

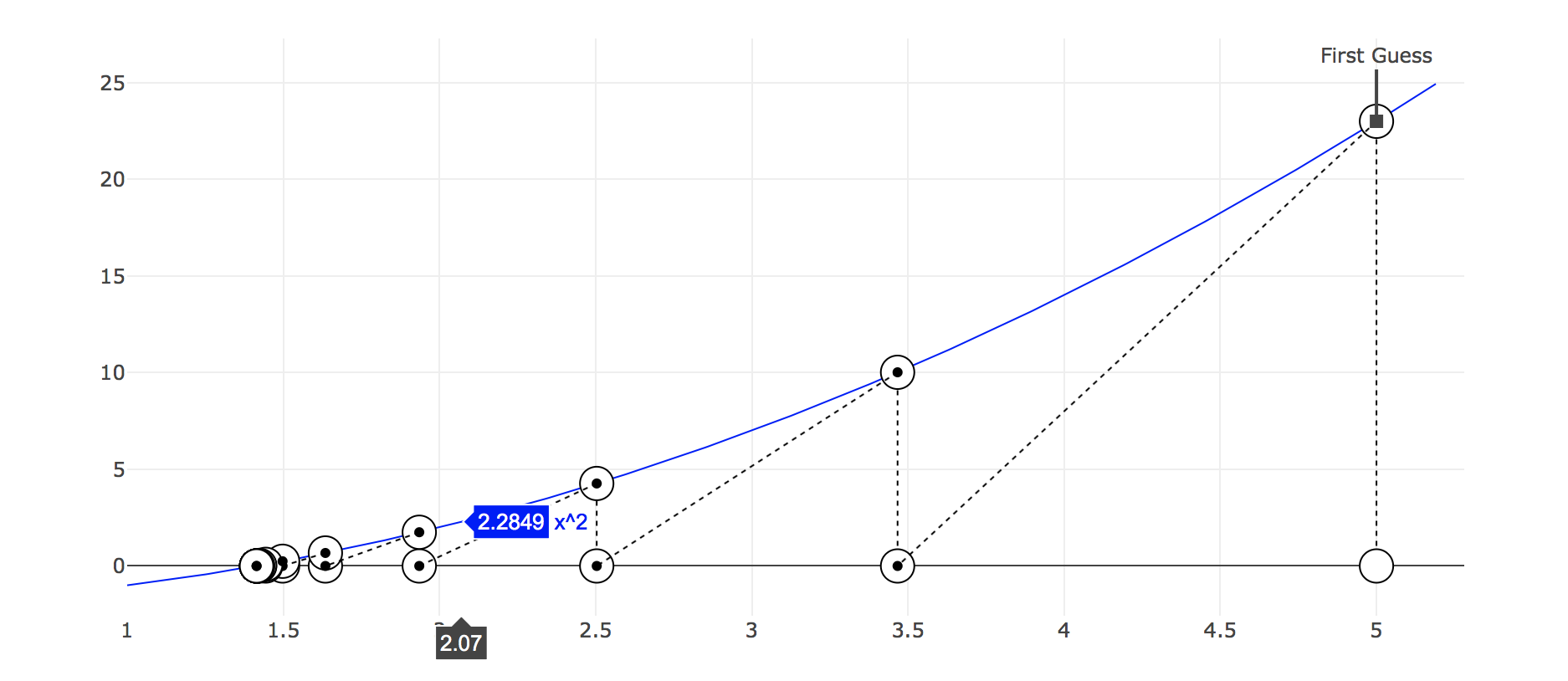
}

return x;

}

The converges looks as follows in tabular and graphical form





## Linear Regression

## Beta of Portfolio