Interpolation

Introduction to Numerical Analysis

Joaquin Cavieres

Example 1: Linear interpolation

```
# Linear interpolation
linterp <- function (x1 , y1 , x2 , y2) {
  m <- (y2 - y1) / (x2 - x1)
  b <- y2 - m * x2
  return (c(b, m))
}

inter1 <- linterp(1, 2, 0, -2)
inter1
```

```
## [1] -2  4
```

The function "`inter1`" estimate the coefficients for the slope and intercept.

Observation

There are potential sources of uncertainty in the estimation here because, if the two points of the sample are particularly close between them, then the operations $y2 - y1$ o $x2 - x1$ can have an error falsely alarming.

This can be a major problem in the second subtraction, since it is used as the denominator in a fraction, and there are other much more convenient methods to avoid this type of problem.

Example 2: Higher-Order Polynomial Interpolation

```
# Function taken from the book "Computational Methods for Numerical Analysis with R"
polyinterp <- function (x, y) {
if( length (x) != length (y))
stop (" Length of x and y vectors must be the same ")
n <- length (x) - 1
vandermonde <- rep (1, length (x))
for(i in 1:n) {
xi <- x^i
vandermonde <- cbind ( vandermonde , xi)
}
beta <- solve ( vandermonde , y)
names ( beta ) <- NULL
return ( beta )
}
```

Now we can create a vector of observations ($\boldsymbol{y}$) and the points where were taken that observations ($\boldsymbol{x}$)

```
x <- c(-1, 1, 0)
y <- c(-2, 2, -1)
(p <- polyinterp(x, y))
```

```
## [1] -1  2  1
```

Example 3:

The process for piecewise linear interpolation uses the same process as our linear interpolation model, except it repeats it for each consecutive pair of data points along the x-axis.

```
pwiselinterp <- function (x, y) {
n <- length (x) - 1
y <- y[order(x)]
x <- x[order(x)]
mvec <- bvec <- c()
for(i in 1:n) {
p <- linterp (x[i], y[i], x[i + 1], y[i + 1])
mvec <- c(mvec , p [2])
bvec <- c(bvec , p [1])
}
return ( list (m = mvec , b = bvec ))
}
```

```
x <- c(-2, -1, 0, 1)
y <- c(-1, -2, -1, 2)
pwiselinterp(x, y)
```

```
## $m
## [1] -1  1  3
##
## $b
## [1] -3 -1 -1
```

Example 4: Cubic Spline

```
tridiagmatrix <- function (L, D, U, b) {
n <- length (D)
L <- c(NA , L)
## The forward sweep
U[1] <- U[1] / D[1]
b[1] <- b[1] / D[1]
for(i in 2:(n - 1)) {
U[i] <- U[i] / (D[i] - L[i] * U[i - 1])
b[i] <- (b[i] - L[i] * b[i - 1]) /
(D[i] - L[i] * U[i - 1])
}
b[n] <- (b[n] - L[n] * b[n - 1]) /
(D[n] - L[n] * U[n - 1])
## The backward sweep
x <- rep.int (0, n)
x[n] <- b[n]
for(i in (n - 1) :1)
x[i] <- b[i] - U[i] * x[i + 1]
return (x)
}

cubicspline <- function (x, y) {
n <- length (x)
dvec <- bvec <- avec <- rep (0, n - 1)
vec <- rep (0, n)
deltax <- deltay <- rep (0, n - 1)
## Find delta values and the A- vector
for(i in 1:(n - 1)) {
avec [i] <- y[i]
deltax [i] = x[i + 1] - x[i]
deltay [i] = y[i + 1] - y[i]
```

```
}
## Assemble a tridiagonal matrix of coefficients
Au <- c(0, deltax [2:(n -1) ])
Ad <- c(1, 2 * ( deltax [1:(n -2)] + deltax [2:(n -1) ]) , 1)
Al <- c( deltax [1:(n -2)], 0)
vec [0] <- vec[n] <- 0
for(i in 2:(n - 1))
vec[i] <- 3 * ( deltay [i] / deltax [i] -
deltay [i -1] / deltax [i -1])
cvec <- tridiagmatrix(Al , Ad , Au , vec)
## Compute B- and D- vectors from the C- vector
for(i in 1:(n -1)) {
bvec [i] <- ( deltay [i] / deltax [i]) -
( deltax [i] / 3) * (2 * cvec [i] + cvec [i + 1])
dvec [i] <- ( cvec [i+1] - cvec [i]) / (3 * deltax [i])
}
return ( list (a = avec , b = bvec ,
c = cvec [1:( n - 1)], d = dvec ))
}
```

```
x <- c(-2, -1, 0, 1)
y <- c(-1, -2, -1, 2)
cubicspline(x, y)
```

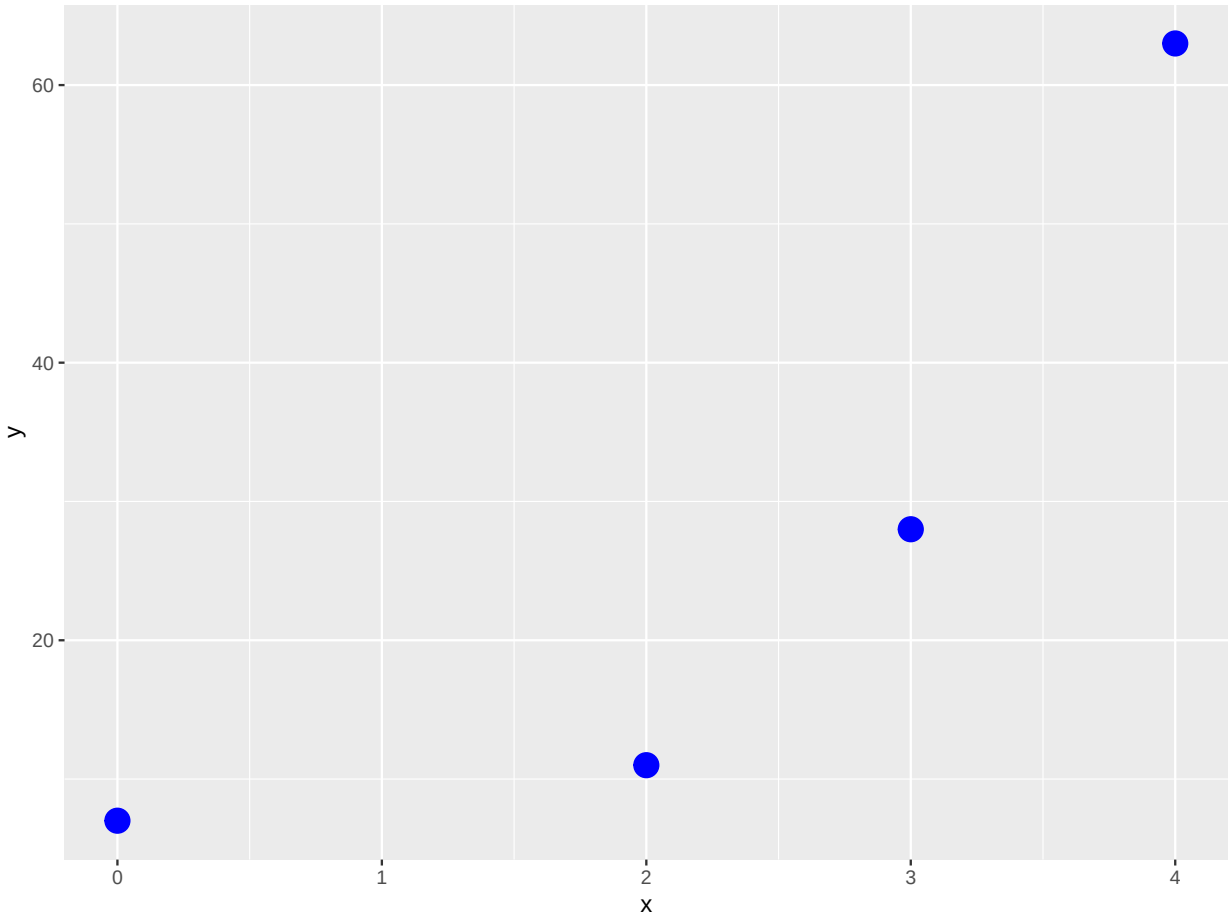
```
## $a
## [1] -1 -2 -1
##
## $b
## [1] -1.4 -0.2  2.2
##
## $c
## [1] 0.0 1.2 1.2
##
## $d
## [1]  0.4  0.0 -0.4
```

Example 5: Lagrande interpolation

```
# Interpolacion de Lagrange (ejemplo 2)
x <- c(0, 2, 3, 4)
y <- c(7, 11, 28, 63)
```

```
df <- data.frame(cbind(x, y))

library(ggplot2)
ggplot(df, aes(x=x, y=y)) + geom_point(size=5, col='blue')
```



Since we want to find a polynomial that passes (interpolates) through these points, then we must calculate following the definition of the Lagrange interpolant:

$$L_1(x) = \frac{(x-2)(x-3)(x-4)}{(0-2)(0-3)(0-4)} = -\frac{1}{24}(x-2)(x-3)(x-4)$$

and then $L_2(x)$ is:

$$L_2(x) = \frac{(x-0)(x-3)(x-4)}{(2-0)(2-3)(2-4)} = \frac{1}{4}x(x-3)(x-4)$$

and finally $L_3(x)$ y $L_4(x)$:

$$L_3(x) = \frac{(x-0)(x-2)(x-4)}{(3-0)(3-2)(3-4)} = -\frac{1}{3}x(x-2)(x-4)$$

$$L_4(x) = \frac{(x-0)(x-2)(x-3)}{(4-0)(4-2)(4-3)} = \frac{1}{8}x(x-2)(x-3)$$

The polynomials found are multiplied by the corresponding values of y resulting in the following polynomial

$$7\left(-\frac{1}{24}(x-2)(x-3)(x-4)\right)+11\left(\frac{1}{4}x(x-3)(x-4)\right)-28\left(-\frac{1}{3}x(x-2)(x-4)\right)+63\left(\frac{1}{8}x(x-2)(x-3)\right)$$

reduced to the next expression:

$$x^3 - 2x + 7$$

Example in R

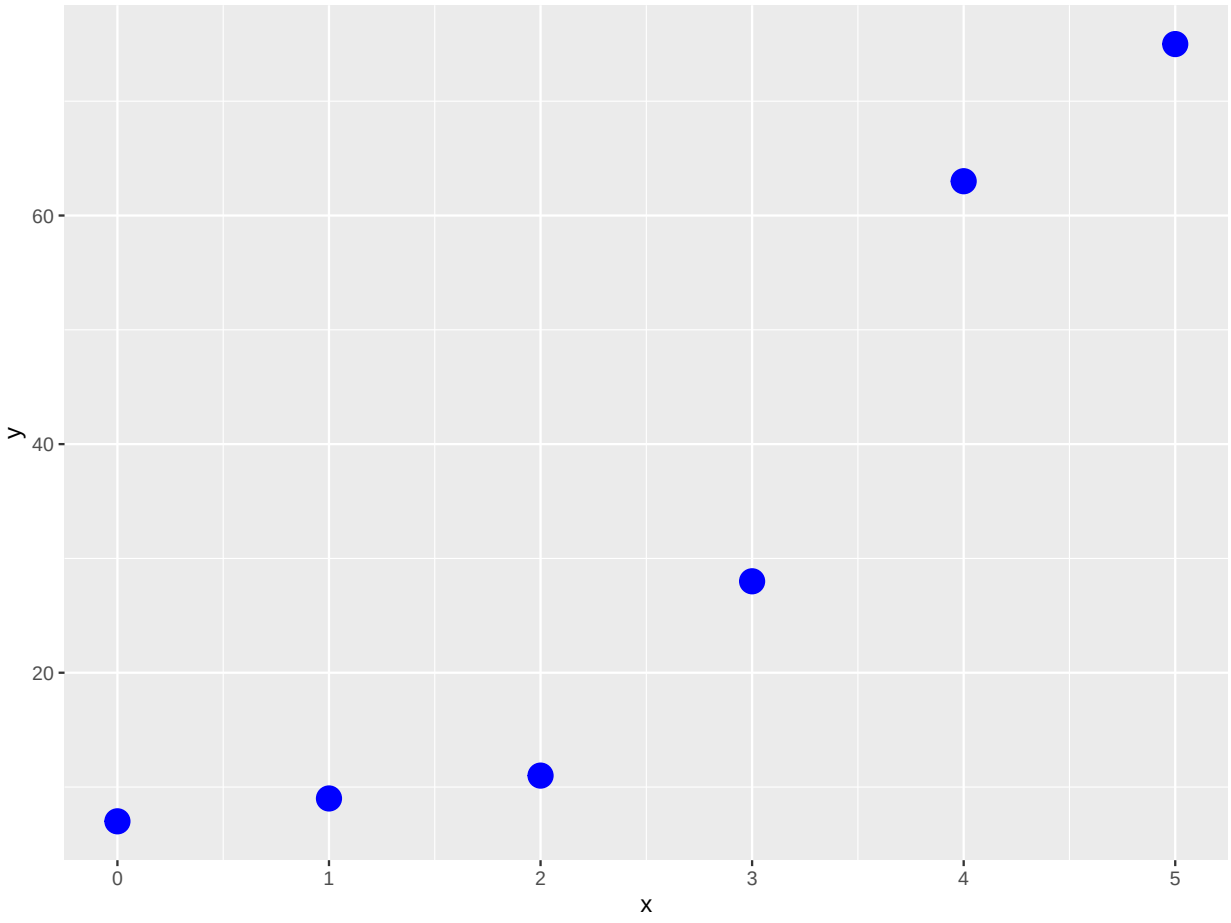
```
poly_lagrange <- function (x, y) {
if( length (x) != length (y))
stop ("The length of x and y must be the same")
n <- length (x) - 1
vandermonde <- rep (1, length (x))
for(i in 1:n) {
xi <- x^i
vandermonde <- cbind ( vandermonde , xi)
}
beta <- solve ( vandermonde , y)
names ( beta ) <- NULL
return ( beta )
}
```

We created a different set of data:

```
x <- c(0, 1, 2, 3, 4, 5)
y <- c(7, 9, 11, 28, 63, 75)

df <- data.frame(cbind(x, y))
```

```
library(ggplot2)
ggplot(df, aes(x=x, y=y)) + geom_point(size=5, col='blue')
```



We apply the function `poly_lagrange`

```
poly_lagrange(x, y)
```

```
## [1]  7.0000000  3.6000000  0.3333333 -3.8333333  2.1666667 -0.2666667
```

What the polynomial gives us:

$$7 + 3.6x + 0.3333333x^2 - 3.833333x^3 + 2.166667x^4 - 0.2666667x^5$$

We comprobe our results with the library `polynom`:

```
# install.packages("polynom")
library(polynom)
poly.calc(x,y)
```

```
## 7 + 3.6*x + 0.3333333*x^2 - 3.833333*x^3 + 2.166667*x^4 - 0.2666667*x^5
```

Finally, we can plot the polynomial at the given points:

```
x <- c(0, 1, 2, 3, 4, 5)
y <- c(7, 9, 11, 28, 63, 75)

df <- data.frame(cbind(x, y))


g <- function(x) {
     return(7 + 3.6*x + 0.333333*x^2 - 3.83333*x^3 + 2.166667*x^4 - 0.2666667*x^5)
}

ggplot(df, aes(x=x, y=y)) + geom_point(size=5, col='red') +
                            stat_function(fun = g, size=1.25, alpha=0.4)
```