Lagrange or Newton polynomial interpolations are useful interpolation techniques to have in your sleeves, but they don’t always give the best or desired result. As the degree of the polynomial increases, so do the wiggles. Therefore, it is often advantageous to use piecewise interpolation, also known as spline interpolation.

A spline is simply a curve that connects two or more specific points. Originally, spline was a term for elastic rulers that were bent to pass through a number of predefined points (“knots”). These were used to make technical drawings for shipbuilding and construction by hand.

I recently wrote a post on a Linear Spline program. You can check that out here.

In this post I am sharing with you a C program that performs cubic spline interpolation. The user is asked to enter a set of x and y-axis data-points, and then each of these is joined by a cubic polynomial. So the code would involve finding the equation of cubic polynomial connecting the two successive points.

I won’t be deriving the equations that we would need to solve to get the cubic splines, but I am giving you the equations that we will be using straight away.

So let’s say you two x and y axis points as $x_i$ and $y_i$ respectively, and the intervals between successive x points are $h_i$.

Then, first of all you would need to solve the following system of equations to get the values of $S_i$.

$$
\begin{bmatrix}
  h_0 & 2(h_0 + h_1) & h_1 \\
  h_1 & 2(h_1 + h_2) & h_2 \\
  & h_2 & 2(h_2 + h_3) \\
  & & \ddots \\
  & & 2(h_{n-2} + h_{n-1}) \\
  & & h_{n-1}
\end{bmatrix}
\begin{bmatrix}
  S_0 \\
  S_1 \\
  S_2 \\
  \vdots \\
  S_n
\end{bmatrix} =
\begin{bmatrix}
  f[x_1, x_2] - f[x_0, x_1] \\
  f[x_2, x_3] - f[x_1, x_2] \\
  \vdots \\
  f[x_{n-1}, x_n] - f[x_{n-2}, x_{n-1}]
\end{bmatrix}
$$

In this post I will consider Natural cubic splines for which $S_0 = S_n = 0$, therefore the system remaining to be solved is,
Once you have those you can find the equation of cubic polynomial, $g_i(x)$ in the $i$th interval between the points $(x_i, y_i), (x_{i+1}, y_{i+1})$, given by:

$$g_i(x) = a_i(x - x_i)^3 + b_i(x - x_i)^2 + c_i(x - x_i) + d_i$$

where

$$a_i = \frac{S_{i+1} - S_i}{6h_i}$$
$$b_i = \frac{S_i}{2}$$
$$c_i = \frac{y_{i+1} - y_i}{h_i} - \frac{2h_i S_i + h_i S_{i+1}}{6}$$
$$d_i = y_i$$

CODE:

```c
#include<stdio.h>
#include<math.h>

//************************************************
//*************CUBIC SPLINE PROGRAM*****************
//************************************************

The program asks the user to enter the data-points and then returns the cubic splines equations
for each interval
Equation for ith interval being:
a_i(x-x_i)^3+b_i(x-x_i)^2+c_i(x-x_i)+d_i*/
#include<stdio.h>
#include<math.h>

/******
Function that performs Gauss-Elimination and returns the Upper triangular matrix and solution of equations:
There are two options to do this in C.
1. Pass the augmented matrix (a) as the parameter, and calculate and store the upperTriangular(Gauss-Eliminated Matrix) in it.
2. Use malloc and make the function of pointer type and return the pointer.
This program uses the first option.
```
void gaussEliminationLS(int m, int n, double a[m][n], double x[n-1]){
    int i,j,k;
    for(i=0;i<m-1;i++){
        /*//Partial Pivoting
        for(k=i+1;k<m;k++){
            //If diagonal element(absolute value) is smaller than any of
            the terms below it
            if(fabs(a[i][i])<fabs(a[k][i])){
                //Swap the rows
                for(j=0;j<n;j++){
                    double temp;
                    temp=a[i][j];
                    a[i][j]=a[k][j];
                    a[k][j]=temp;
                }
            }
        }*/
        //Begin Gauss Elimination
        for(k=i+1;k<m;k++){
            double term=a[k][i]/ a[i][i];
            for(j=0;j<n;j++){
                a[k][j]=a[k][j]-term*a[i][j];
            }
        }
        //Begin Back-substitution
        for(i=m-1;i>=0;i--){
            x[i]=a[i][n-1];
            for(j=i+1;j<n-1;j++){
                x[i]=x[i]-a[i][j]*x[j];
            }
            x[i]=x[i]/a[i][i];
        }
    }
}

Cubic Spline coefficients calculator
Function that calculates the values of ai, bi, ci, and di's for the cubic splines:
ai(x-xi)^3+bi(x-xi)^2+ci(x-xi)+di

void cSCoeffCalc(int n, double h[n], double sig[n+1], double y[n+1], double a[n],
        double b[n], double c[n], double d[n]){
    int i;
    for(i=0;i<n;i++){
        d[i]=y[i];
        b[i]=sig[i]/2.0;
        a[i]=(sig[i+1]-sig[i])/(h[i]*6.0);
        c[i]=(y[i+1]-y[i])/h[i]-h[i]*(2*sig[i]+sig[i+1])/6.0;
    }
}

Function to generate the tridiagonal augmented matrix
for cubic spline for equidistant data-points
Parameters:
n: no. of data-points
h: array storing the successive interval widths
a: matrix that will hold the generated augmented matrix
y: array containing the y-axis data-points
*************************/

void tridiagonalCubicSplineGen(int n, double h[n], double a[n-1][n], double y[n+1]){  int i;
    for(i=0;i<n-1;i++){
        a[i][i]=2*(h[i]+h[i+1]);
    }
    for(i=0;i<n-2;i++){
        a[i][i+1]=h[i+1];
        a[i+1][i]=h[i+1];
    }
    for(i=1;i<n;i++){
        a[i-1][n-1]=(y[i+1]-y[i])*6/(double)h[i]-(y[i]-y[i-1])*6/(double)h[i-1];
    }
}
******
Function that prints the elements of a matrix row-wise
Parameters: rows(m),columns(n),matrix[m][n]
*******/

void printMatrix(int m, int n, double matrix[m][n]){  int i,j;
    for(i=0;i<m;i++){
        for(j=0;j<n;j++){
            printf("%lf	",matrix[i][j]);
        }
        printf("\n");
    }
}
******
Function that copies the elements of a matrix to another matrix
Parameters: rows(m),columns(n),matrix1[m][n], matrix2[m][n]
*******/

void copyMatrix(int m, int n, double matrix1[m][n], double matrix2[m][n]){  int i,j;
    for(i=0;i<m;i++){
        for(j=0;j<n;j++){
            matrix2[i][j]=matrix1[i][j];
        }
    }
}
main(){
    int m,i;
    printf("Enter the no. of data-points:\n");
    scanf("%d",&m);
    int n=m-1; //Now (n+1) is the total no. of data-points, following our convention
double x[n+1]; //array to store the x-axis points
double y[n+1]; //array to store the y-axis points
double h[n];   /////array to store the successive interval widths
printf("Enter the x-axis values:\n");
for(i=0;i<n+1;i++){
    scanf("%lf",&x[i]);
}
printf("Enter the y-axis values:\n");
for(i=0;i<n+1;i++){
    scanf("%lf",&y[i]);
}
for(i=0;i<n;i++){
    h[i]=x[i+1]-x[i];
}
double a[n]; //array to store the ai's
double b[n]; //array to store the bi's
double c[n]; //array to store the ci's
double d[n]; //array to store the di's
double sig[n+1]; //array to store Si's
double sigTemp[n-1]; //array to store the Si's except S0 and Sn
sig[0]=0;
sig[n]=0;
double tri[n-1][n]; //matrix to store the tridiagonal system of equations that
will solve for Si's
tridiagonalCubicSplineGen(n,h,tri,y); //to initialize tri[n-1][n]
printf("The tridiagonal system for the Natural spline is:\n\n");
printMatrix(n-1,n,tri);
//Perform Gauss Elimination
gaussEliminationLS(n-1,n,tri,sigTemp);
for(i=1;i<n;i++){
    sig[i]=sigTemp[i-1];
}
//Print the values of Si's
for(i=0;i<n+1;i++){
    printf("\nSig[%d] = %lf\n",i,sig[i]);
}
//calculate the values of ai's, bi's, ci's, and di's
cSCoeffCalc(n,h,sig,y,a,b,c,d);
printf("The equations of cubic interpolation polynomials between the successive
intervals are:\n\n");
for(i=0;i<n;i++){
    printf("P%d(x) b/w [%lf,%lf] = %lf*(x-%lf)^3+%lf*(x-%lf)^2+%lf*(x-%lf)+%lf\n",i,x[i],x[i+1],a[i],x[i],b[i],x[i],c[i],x[i],d[i]);
}

OUTPUT:
If you know that your points will be equidistant, that is all hi’s are equal to h, then the above code can be modified to the following:

\[
\begin{bmatrix}
4 & 1 \\
1 & 4 & 1 \\
1 & 4 & 1 \\
\vdots & \ddots & 1 \\
1 & 4 \\
\end{bmatrix}
\begin{bmatrix}
S_1 \\
S_2 \\
S_3 \\
\vdots \\
S_{n-1} \\
\end{bmatrix}
= \frac{6}{h^2}
\begin{bmatrix}
y_2 - 2y_1 + y_0 \\
y_3 - 2y_2 + y_1 \\
y_4 - 2y_3 + y_2 \\
\vdots \\
y_n - 2y_{n-1} + y_{n-2} \\
\end{bmatrix}
\]

The ai’s, bi’s, ci’s and di’s will be modified accordingly, so that hi’s become h.

CODE:

```c
#include<stdio.h>
#include<math.h>

/**********************CUBIC SPLINE FOR EQUIDISTANT POINTS**********************/
#include<stdio.h>
#include<math.h>

Function that performs Gauss-Elimination and returns the Upper triangular matrix and solution of equations:
There are two options to do this in C.
1. Pass the augmented matrix (a) as the parameter, and calculate and store the upperTriangular(Gauss-Eliminated Matrix) in it.
2. Use malloc and make the function of pointer type and return the pointer.
This program uses the first option.

```c
void gaussEliminationLS(int m, int n, double a[m][n], double x[n-1]){
    int i,j,k;
    for(i=0;i<m-1;i++){
        //Partial Pivoting
        for(k=i+1;k<m;k++){
            //If diagonal element (absolute value) is smaller than any of
            the terms below it
            if(fabs(a[i][i])<fabs(a[k][i])){
                //Swap the rows
                for(j=0;j<n;j++){
                    double temp;
                    temp=a[i][j];
                    a[i][j]=a[k][j];
                    a[k][j]=temp;
                }
            }
        }
        //Begin Gauss Elimination
        for(k=i+1;k<m;k++){
            double term=a[k][i]/a[i][i];
            for(j=0;j<n;j++){
                a[k][j]=a[k][j]-term*a[i][j];
            }
        }
    }
    //Begin Back-substitution
    for(i=m-1;i>=0;i--){
        x[i]=a[i][n-1];
        for(j=i+1;j<n-1;j++){
            x[i]=x[i]-a[i][j]*x[j];
        }
        x[i]=x[i]/a[i][i];
    }
}
```

Cubic Spline coefficients calculator

```c
void cSCoeffCalc(int n, double h, double sig[n+1], double y[n+1], double a[n], double b[n], double c[n], double d[n]){
    int i;
    for(i=0;i<n;i++){
        d[i]=y[i];
        b[i]=sig[i]/2.0;
        a[i]=(sig[i+1]-sig[i])/(h*6.0);
        c[i]=(y[i+1]-y[i])/h-h*(2*sig[i]+sig[i+1])/6.0;
    }
}
```

Function to generate the tridiagonal augmented matrix for cubic spline for equidistant data-points

```c
**Cubic Spline (Piecewise Interpolation) – C PROGRAM**

Manas Sharma (c) Bragitoff.com
Parameters:
n:
a:
y:

```
void tridiagonalCubicSplineGen(int n, double h, double a[n-1][n], double y[n+1]){
    int i;
    for(i=0;i<n-1;i++){
        a[i][i]=4;
    } 
    for(i=0;i<n-2;i++){
        a[i][i+1]=1;
        a[i+1][i]=1;
    } 
    for(i=0;i<n-1;i++){
        a[i][n-1]=(y[i+2]-2*y[i+1]+y[i])*6/h/h;
    } 
}
```

```
function that prints the elements of a matrix row-wise
Parameters: rows(m), columns(n), matrix[m][n]
```

```
void printMatrix(int m, int n, double matrix[m][n]){
    int i,j;
    for(i=0;i<m;i++){
        for(j=0;j<n;j++){
            printf("%lf	",matrix[i][j]);
        }
        printf("\n");
    }
}
```

```
function that copies the elements of a matrix to another matrix
Parameters: rows(m), columns(n), matrix1[m][n], matrix2[m][n]
```

```
void copyMatrix(int m, int n, double matrix1[m][n], double matrix2[m][n]){
    int i,j;
    for(i=0;i<m;i++){
        for(j=0;j<n;j++){
            matrix2[i][j]=matrix1[i][j];
        }
    }
}
```

```
main()
{
    int m,i;
    printf("Enter the no. of data-points:\n");
    scanf("%d", &m);
    int n=m-1;        //Now (n+1) is the total no. of data-points, following our
                     //convention
    double x[n+1];
    double y[n+1];
    printf("Enter the x-axis values:\n");
}
```
for(i=0;i<n+1;i++){
    scanf("%lf",&x[i]);
}
printf("Enter the y-axis values:\n");
for(i=0;i<n+1;i++){
    scanf("%lf",&y[i]);
}
double h=x[1]-x[0]; //space interval
double a[n];
double b[n];
double c[n];
double d[n];
double sig[n+1];
double sigTemp[n-1];
sig[0]=0;
sig[n]=0;
double tri[n-1][n];
tridiagonalCubicSplineGen(n,h,tri,y);
printf("The tridiagonal system for the Natural spline is:\n");
printMatrix(n-1,n,tri);
//Perform Gauss Elimination
gaussEliminationLS(n-1,n,tri,sigTemp);
for(i=1;i<n;i++){
    sig[i]=sigTemp[i-1];
}
for(i=0;i<n+1;i++){
    printf("\nSig[%d] = %lf\n",i,sig[i]);
}
cSCoeffCalc(n,h,sig,y,a,b,c,d);
printf("The equations of cubic interpolation polynomials between the successive
intervals are:\n");
for(i=0;i<n;i++){
    printf("P%d(x) b/w [%lf,%lf] = %lf*(x-%lf)^3+%lf*(x-%lf)^2+%lf*(x-%lf)+%lf\n",i,x[i],x[i+1],a[i],x[i],b[i],x[i],c[i],x[i],d[i]);
}

OUTPUT:

References and Resources:

https://tools.timodenk.com/cubic-spline-interpolation
http://mathworld.wolfram.com/CubicSpline.html
http://www.maths.lth.se/na/courses/FMN081/FMN081-06/lecture11.pdf
Manas Sharma

PhD researcher at Friedrich-Schiller University Jena, Germany. I'm a physicist specializing in theoretical, computational and experimental condensed matter physics. I like to develop Physics related apps and softwares from time to time. Can code in most of the popular languages. Like to share my knowledge in Physics and applications using this Blog and a YouTube channel.

Consider donating if you found the information useful

Appreciate your blog: $3