MA 322: Scientific Computing - Midsem Project

Due on Sunday, October 5, 2015

Jiten Chandra Kalita

Tushar Sircar - 130123038

Overview

A cubic spline is a spline constructed of piecewise third-order polynomials which pass through a set of control points. The second derivative of each polynomial is commonly set to zero at the endpoints, since this provides a boundary condition that completes the system of equations.
In two-dimensions, two cubic spline functions can be used together to form a parametric representation of a complicated curve that turns and twists. Select points on the curve and label them t = 0,1,2....n. For each value of t, read-off the x- and y-coordinates of the point, thus producing a table:

t	0	1	...	n
x	x_0	x_1	...	x_n
y	y_0	y_1	...	y_n

Then you can fit x = S(t) and y = F(t), where S and F are natural cubic spline interpolants. S and F give a parametric representation of the curve.

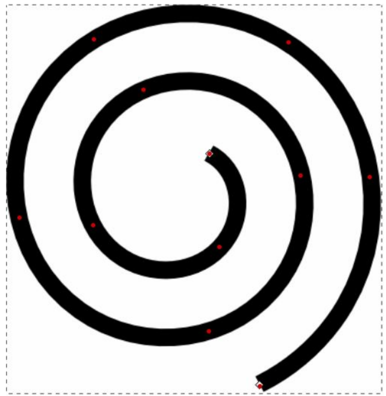
Procedure

(1) Extract co-ordinates or points using which we wish to interpolate the given curve. This is done using an online web plot digitaliser.
(2) These points are hardcoded for part (a) of the project and are read from a text file for part (b).
(3) a[],b[],c[],d[] represent the coefficients of the cubic polynomial found using the cubic spline algorithm.
(4) tA[],tB[],tC[],tD[] are temporary arrays used at that the time of applying Thomas's Algorithm.
(5) Firstly all x-coordinates are taken and x = S(t) is interpolated and then y = F(t) is interpolated. This gives us two functions, one for x and one for y.
(6) We are using t as a parameter t = 0,1,2,3 ... and so h = 1.
(7) Each interval t = i to t = i+1 is divided into 10 points and using the generated functions we can get values for x and y.
(8) Multiple values are generated for x and y using the cubic spline and plotted as x_i, y_i to get the original curve.

Question 1(a)

Draw a spiral like above and reproduce it by way of parametric spline functions.

I have used the following image of a spiral to extract data points which are then fed into the cubic spline algorithm.

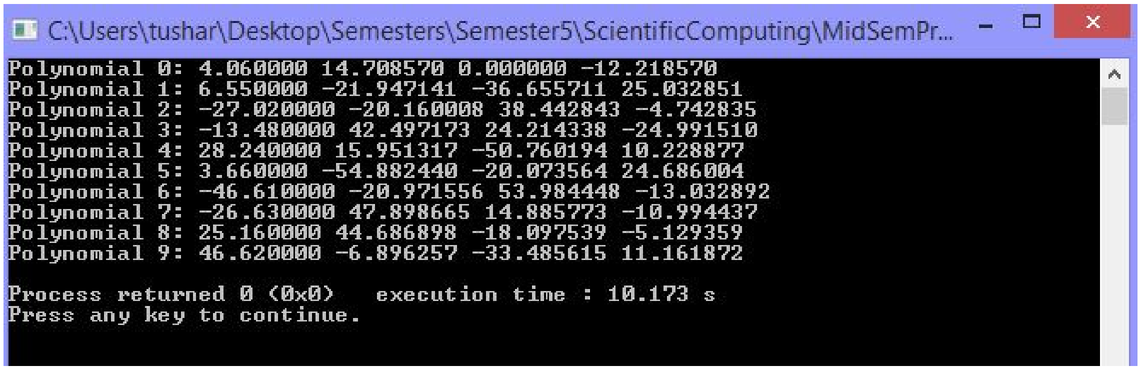


```
//TUSHAR SIRCAR
//130123038
#include<iostream>
#include<stdio.h>
#include<cmath>
using namespace std;

void solveThomas(double a[],double b[],double c[],double t[],double f[],int k);

int main()
{
    int noOfNodes = 11;
    double h = 1;

    double X[] = {4.06,6.55,-27.02,-13.48,28.24,3.66,-46.61,-26.63,25.16,46.62,17.40};
    double Y[] = {11.62,-12.20,-6.97,28.10,6.007,-34.10,-4.84,41.08,40.31,5.62,-48.25};
    double plotX[100];
    double plotY[100];
    double t[noOfNodes];
    double a[noOfNodes],b[noOfNodes],c[noOfNodes],d[noOfNodes];
    int k = noOfNodes - 1;
    double tA[k],tB[k],tC[k],tF[k],tX[k];


    FILE *output = fopen("output.txt","w");

    //initialise the x-axis points
    t[0] = 0;
    for(int i=1; i<noOfNodes; i++)
        t[i] = t[i-1] + 1;


    //first parametrise the x-coordinates
    for(int i=0; i<noOfNodes; i++)
        a[i] = X[i];
```

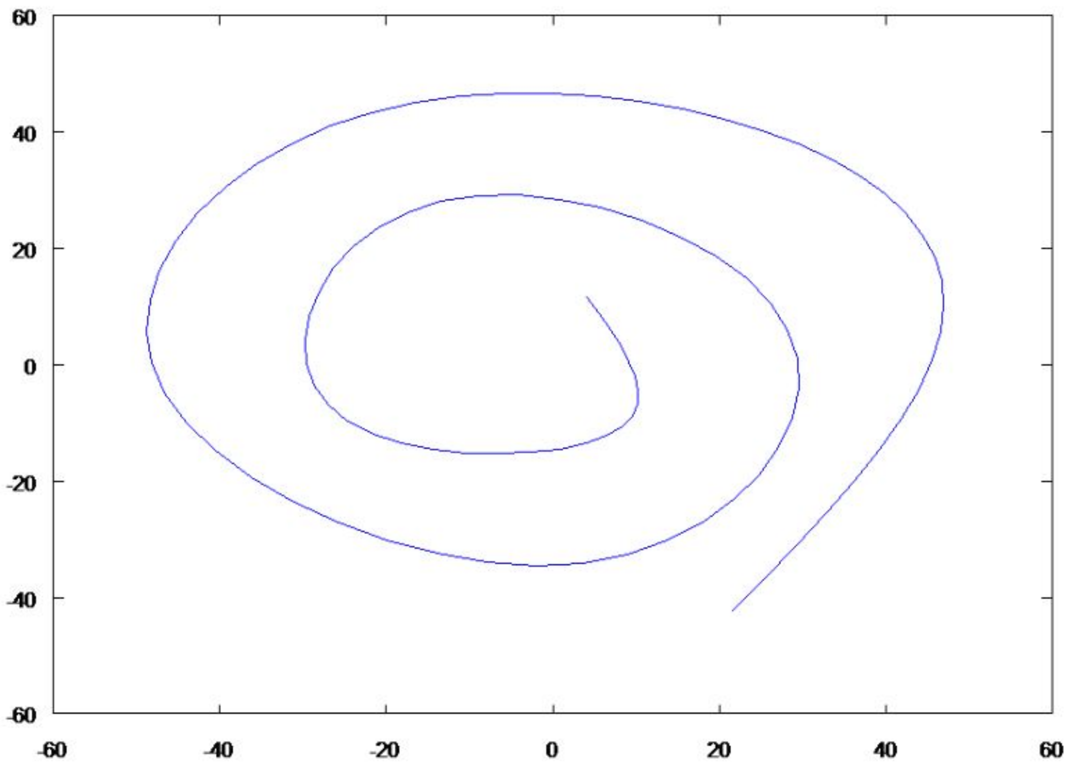
```
    c[0] = c[noOfNodes-1] = 0;


    for(int i=1; i<k; i++)
    {
        tA[i] = h;
        tB[i] = (4*h);
        tC[i] = h;
        tF[i] = ((double)3/h)*(a[i+1] - a[i] - a[i] + a[i-1]);

    }
    tX[0] = tX[k] = 0;
    tF[0] = tF[k] = 0;
    solveThomas(tA,tB,tC,tX,tF,k);


    for(int i=0; i<=k; i++)
        c[i] = tX[i];

    for(int i=0; i<k; i++)
    {
        b[i] = ((a[i+1]-a[i])/h) - (h/3)*((2*c[i])+c[i+1]);
        d[i] = (c[i+1]-c[i])/(3*h);
    }

    for(int i=0; i<100; i++)
    {
        int polyNumber = i/10;
        double p = (double)i*(0.1);
        plotX[i] = a[polyNumber] + (b[polyNumber]*
        (p - (double)polyNumber)) + (c[polyNumber]*pow(p-(double)polyNumber,2)) +
        (d[polyNumber]*pow(p-(double)polyNumber,3));
    }

    for(int i=0; i<10; i++)
       printf("Polynomial %d: %f %f %f %f\n",i,a[i],b[i],c[i],d[i]);



    //second parametrise the y-coordinates
    for(int i=0; i<noOfNodes; i++)
        a[i] = Y[i];
    c[0] = c[noOfNodes-1] = 0;

    for(int i=1; i<k; i++)
    {
        tA[i] = h;
        tB[i] = (4*h);
        tC[i] = h;
        tF[i] = ((double)3/h)*(a[i+1] - a[i] - a[i] + a[i-1]);

    }
    tX[0] = tX[k] = 0;
```

```
    tF[0] = tF[k] = 0;
    solveThomas(tA,tB,tC,tX,tF,k);


    for(int i=0; i<=k; i++)
        c[i] = tX[i];

    for(int i=0; i<k; i++)
    {
        b[i] = ((a[i+1]-a[i])/h) - (h/3)*((2*c[i])+c[i+1]);
        d[i] = (c[i+1]-c[i])/(3*h);
    }

    for(int i=0; i<100; i++)
    {
        int polyNumber = i/10;
        double p = (double)i*(0.1);
        plotY[i] = a[polyNumber] + (b[polyNumber]*(p - (double)polyNumber)) + (c[polyNumber]*pow(p-(
    }

    for(int i=0; i<100; i++)
        fprintf(output,"%f,%f\n",plotX[i],plotY[i]);

    fclose(output);

    return 0;
}

void solveThomas(double a[],double b[],double c[],double t[],double f[],int k)
{

    //forward sweep
    f[1] = f[1] - (a[1]*t[0]);
    f[k-1] = f[k-1] - (c[k-1]*t[k]);

    b[1] = b[1]; //unchanged

    for(int i=2; i<=k-1; i++)
    {
        b[i] = b[i] - ((c[i-1]*a[i])/b[i-1]);
        f[i] = f[i] - ((f[i-1]*a[i])/b[i-1]);
    }

    //find solutions using backward sweep
    t[k-1] = f[k-1]/b[k-1];
    for(int i=k-2; i>=1; i--)
        t[i] = (f[i] - (c[i]*t[i+1]))/b[i];


}
```

OUTPUT



```
Polynomial 0: 4.060000 14.708570 0.000000 -12.218570
Polynomial 1: 6.550000 -21.947141 -36.655711 25.032851
Polynomial 2: -27.020000 -20.160008 38.442843 -4.742835
Polynomial 3: -13.480000 42.497173 24.214338 -24.991510
Polynomial 4: 28.240000 15.951317 -50.760194 10.228877
Polynomial 5: 3.660000 -54.882440 -20.073564 24.686004
Polynomial 6: -46.610000 -20.971556 53.984448 -13.032892
Polynomial 7: -26.630000 47.898665 14.885773 -10.994437
Polynomial 8: 25.160000 44.686898 -18.097539 -5.129359
Polynomial 9: 46.620000 -6.896257 -33.485615 11.161872

Process returned 0 (0x0)   execution time : 10.173 s
Press any key to continue.
```



RESULTS

(a) Number of data points taken are 11.

(b) Between t = i and t =i+1, 10 points are taken to draw the spline.

Question 1(b)

Using at most 20 knots and cubic splines, plot on a computer plotter an outline of your own SIGNATURE. (The following signature has been used)

```
//TUSHAR SIRCAR
//130123038
#include<iostream>
#include<stdio.h>
#include<cmath>
using namespace std;

void solveThomas(double a[],double b[],double c[],double t[],double f[],int k);
int main()
{
    int noOfNodes = 33;
    double h = 1;

    FILE *output = fopen("output.txt","w");
    FILE *input = fopen("input.txt","r");


    float X[noOfNodes];
    float Y[noOfNodes];
    for(int i=0; i<noOfNodes; i++)
        fscanf(input,"%f, %f",&X[i],&Y[i]);


    double plotX[noOfNodes * 10];
    double plotY[noOfNodes * 10];
    double t[noOfNodes];
    double a[noOfNodes],b[noOfNodes],c[noOfNodes],d[noOfNodes];
    int k = noOfNodes - 1;
    double tA[k],tB[k],tC[k],tF[k],tX[k];



    //initialise the x-axis points
    t[0] = 0;
    for(int i=1; i<noOfNodes; i++)
        t[i] = t[i-1] + 1;


    //first parametrise the x-coordinates
    for(int i=0; i<noOfNodes; i++)
```

```
        a[i] = X[i];
    c[0] = c[noOfNodes-1] = 0;


    for(int i=1; i<k; i++)
    {
        tA[i] = h;
        tB[i] = (4*h);
        tC[i] = h;
        tF[i] = ((double)3/h)*(a[i+1] - a[i] - a[i] + a[i-1]);

    }
    tX[0] = tX[k] = 0;
    tF[0] = tF[k] = 0;
    solveThomas(tA,tB,tC,tX,tF,k);


    for(int i=0; i<=k; i++)
        c[i] = tX[i];

    for(int i=0; i<k; i++)
    {
        b[i] = ((a[i+1]-a[i])/h) - (h/3)*((2*c[i])+c[i+1]);
        d[i] = (c[i+1]-c[i])/(3*h);
    }

    for(int i=0; i<(noOfNodes-1)*10; i++)
    {
        int polyNumber = i/10;
        double p = (double)i*(0.1);
        plotX[i] = a[polyNumber] + (b[polyNumber]*(p - (double)polyNumber)) + (c[polyNumber]*pow(p-(d
    }

     for(int i=0; i<33; i++)
       printf("X: Polynomial %d: %f %f %f %f\n",i,a[i],b[i],c[i],d[i]);



    //second parametrise the y-coordinates
    for(int i=0; i<noOfNodes; i++)
        a[i] = Y[i];
    c[0] = c[noOfNodes-1] = 0;

    for(int i=1; i<k; i++)
    {
        tA[i] = h;
        tB[i] = (4*h);
        tC[i] = h;
        tF[i] = ((double)3/h)*(a[i+1] - a[i] - a[i] + a[i-1]);

    }
    tX[0] = tX[k] = 0;
    tF[0] = tF[k] = 0;
```

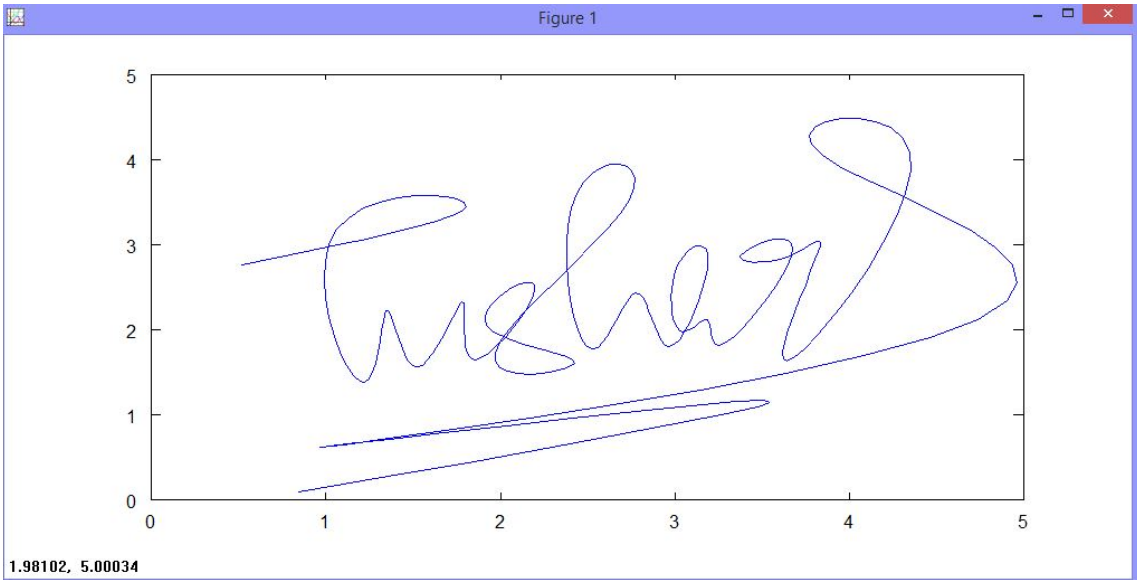
```
    solveThomas(tA,tB,tC,tX,tF,k);


    for(int i=0; i<=k; i++)
        c[i] = tX[i];

    for(int i=0; i<k; i++)
    {
        b[i] = ((a[i+1]-a[i])/h) - (h/3)*((2*c[i])+c[i+1]);
        d[i] = (c[i+1]-c[i])/(3*h);
    }

    for(int i=0; i<(noOfNodes-1)*10; i++)
    {
        int polyNumber = i/10;
        double p = (double)i*(0.1);
        plotY[i] = a[polyNumber] + (b[polyNumber]*(p - (double)polyNumber)) + (c[polyNumber]*pow(p-(
    }

     for(int i=0; i<33; i++)
       printf("Y: Polynomial %d: %f %f %f %f\n",i,a[i],b[i],c[i],d[i]);



    for(int i=0; i<(noOfNodes-1)*10; i++)
        fprintf(output,"%f,%f\n",plotX[i],plotY[i]);

    fclose(output);

    return 0;
}

void solveThomas(double a[],double b[],double c[],double t[],double f[],int k)
{

    //forward sweep
    f[1] = f[1] - (a[1]*t[0]);
    f[k-1] = f[k-1] - (c[k-1]*t[k]);

    b[1] = b[1]; //unchanged

    for(int i=2; i<=k-1; i++)
    {
        b[i] = b[i] - ((c[i-1]*a[i])/b[i-1]);
        f[i] = f[i] - ((f[i-1]*a[i])/b[i-1]);
    }

    //find solutions using backward sweep
    t[k-1] = f[k-1]/b[k-1];
    for(int i=k-2; i>=1; i--)
        t[i] = (f[i] - (c[i]*t[i+1]))/b[i];

```

```
}
```

OUTPUT

```
C:\Users\tushar\Desktop\Semesters\Semester5\ScientificComputing\MidSemPr...
X: Polynomial 0: 0.515567 1.890872 0.000000 -0.604133
X: Polynomial 1: 1.802306 0.078474 -1.812398 0.996173
X: Polynomial 2: 1.064555 -0.557804 1.176120 -0.489373
X: Polynomial 3: 1.193498 0.326319 -0.291997 0.118473
X: Polynomial 4: 1.346293 0.097743 0.063422 0.017169
X: Polynomial 5: 1.524627 0.276095 0.114929 -0.134381
X: Polynomial 6: 1.781270 0.102810 -0.288214 0.259743
X: Polynomial 7: 1.855608 0.305609 0.491014 -0.454709
X: Polynomial 8: 2.197522 -0.076491 -0.873114 0.674622
X: Polynomial 9: 1.922539 0.201147 1.150752 -0.851798
X: Polynomial 10: 2.422640 -0.052743 -1.404642 1.003920
X: Polynomial 11: 1.969174 0.149732 1.607118 -0.987841
X: Polynomial 12: 2.738184 0.400447 -1.356404 0.653627
X: Polynomial 13: 2.435854 -0.351480 0.604477 -0.190244
X: Polynomial 14: 2.498607 0.286742 0.033744 -0.045177
X: Polynomial 15: 2.773916 0.218699 -0.101786 0.080384
X: Polynomial 16: 2.971213 0.256277 0.139364 -0.176331
X: Polynomial 17: 3.190524 0.006013 -0.389629 0.203859
X: Polynomial 18: 3.010767 -0.161667 0.221948 -0.042770
X: Polynomial 19: 3.028277 0.153918 0.093637 -0.086642
X: Polynomial 20: 3.189190 0.081266 -0.166289 0.197967
X: Polynomial 21: 3.302135 0.342589 0.427612 -0.399519
X: Polynomial 22: 3.672817 -0.000742 -0.770944 0.478542
X: Polynomial 23: 3.379673 -0.107006 0.664681 -0.274129
X: Polynomial 24: 3.663218 0.399967 -0.157707 -0.100576
X: Polynomial 25: 3.804902 -0.217175 -0.459435 0.551547
X: Polynomial 26: 3.679840 0.518598 1.195207 -1.047606
X: Polynomial 27: 4.346038 -0.233808 -1.947612 1.680401
X: Polynomial 28: 3.845019 0.912171 3.093591 -3.111482
X: Polynomial 29: 4.739298 -2.235093 -6.240855 4.701830
X: Polynomial 30: 0.965181 -0.611312 7.864636 -4.709740
X: Polynomial 31: 3.508764 0.988740 -6.264584 2.088195
X: Polynomial 32: 0.321115 0.000000 0.000000 0.000000
```

```
Y: Polynomial 0: 2.765768 0.771523 0.000000 -0.086362
Y: Polynomial 1: 3.450929 0.512437 -0.259086 -0.523919
Y: Polynomial 2: 3.180361 -1.577493 -1.830844 1.619482
Y: Polynomial 3: 1.391506 -0.380736 3.027601 -1.818539
Y: Polynomial 4: 2.219832 0.218849 -2.428016 1.548065
Y: Polynomial 5: 1.558730 0.007011 2.216179 -1.459187
Y: Polynomial 6: 2.322732 0.061806 -2.161384 1.424431
Y: Polynomial 7: 1.647586 0.012332 2.111909 -1.251858
Y: Polynomial 8: 2.519970 0.480578 -1.643664 0.697176
Y: Polynomial 9: 2.054060 -0.715221 0.447865 -0.179354
Y: Polynomial 10: 1.607350 -0.357553 -0.090198 0.564577
Y: Polynomial 11: 1.724176 1.155781 1.603532 -0.953185
Y: Polynomial 12: 3.530304 1.503291 -1.256022 -0.179835
Y: Polynomial 13: 3.597739 -1.548257 -1.795527 1.540292
Y: Polynomial 14: 1.794247 -0.518434 2.825350 -1.668598
Y: Polynomial 15: 2.432565 0.126472 -2.180444 1.423920
Y: Polynomial 16: 1.802514 0.037344 2.091316 -1.113289
Y: Polynomial 17: 2.817885 0.880109 -1.248551 0.277215
Y: Polynomial 18: 2.726659 -0.785347 -0.416906 0.465301
Y: Polynomial 19: 1.989707 -0.223255 0.978998 -0.632874
Y: Polynomial 20: 2.112577 -0.163880 -0.919623 0.830949
Y: Polynomial 21: 1.860023 0.489721 1.573224 -0.971577
Y: Polynomial 22: 2.951391 0.721439 -1.341506 0.546292
Y: Polynomial 23: 2.877616 -0.322698 0.297369 0.020884
Y: Polynomial 24: 2.873172 0.334693 0.360022 -0.743957
Y: Polynomial 25: 2.823929 -1.177135 -1.871850 1.891071
Y: Polynomial 26: 1.666015 0.752377 3.801363 -2.134715
Y: Polynomial 27: 4.085040 1.950959 -2.602782 0.612654
Y: Polynomial 28: 4.045871 -1.416643 -0.764820 0.255222
Y: Polynomial 29: 2.119628 -2.180620 0.000844 0.671533
Y: Polynomial 30: 0.611386 -0.164333 2.015443 -1.295366
Y: Polynomial 31: 1.167129 -0.019547 -1.870657 0.623552
Y: Polynomial 32: -0.099522 0.000000 0.000000 0.000000

Process returned 0 (0x0)   execution time : 10.451 s
Press any key to continue.
```



RESULTS

(a) Number of data points taken are 33. Fewer data points were not sufficient to produce a quality interpolation because of high number of curves and intersections in the signature.

(b) Between t = i and t =i+1, 10 points are taken to draw the spline.