Becs-114.1100 Computational Science – exercise round 6

Kunal Ghosh, 546247

November 3, 2015

1 Solution to Question 4

1.1 Determining the natural cubic interpoland S(x) and its Derivative S'(x) from experimental data

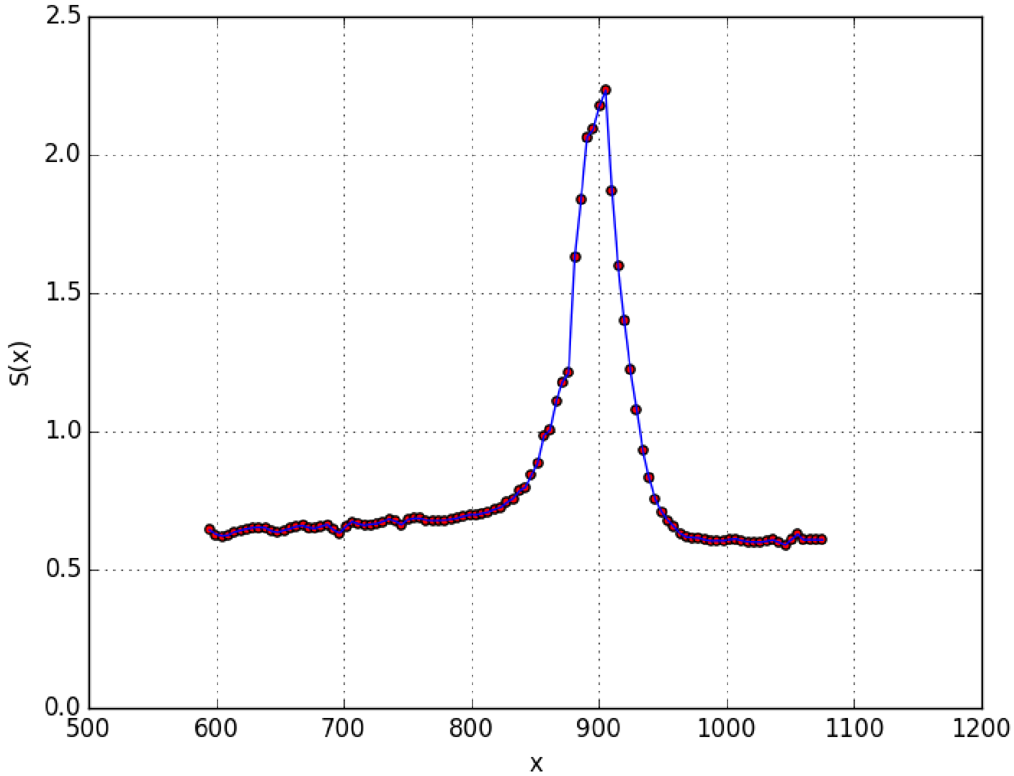

Figure 1: Plot showing the natural cubic interpoland S(X). Values of $S(X_i)$ for X_i in the range of (min(knot values), max(knot values)) are shown as red circles.

The corresponding python code can be found at 2

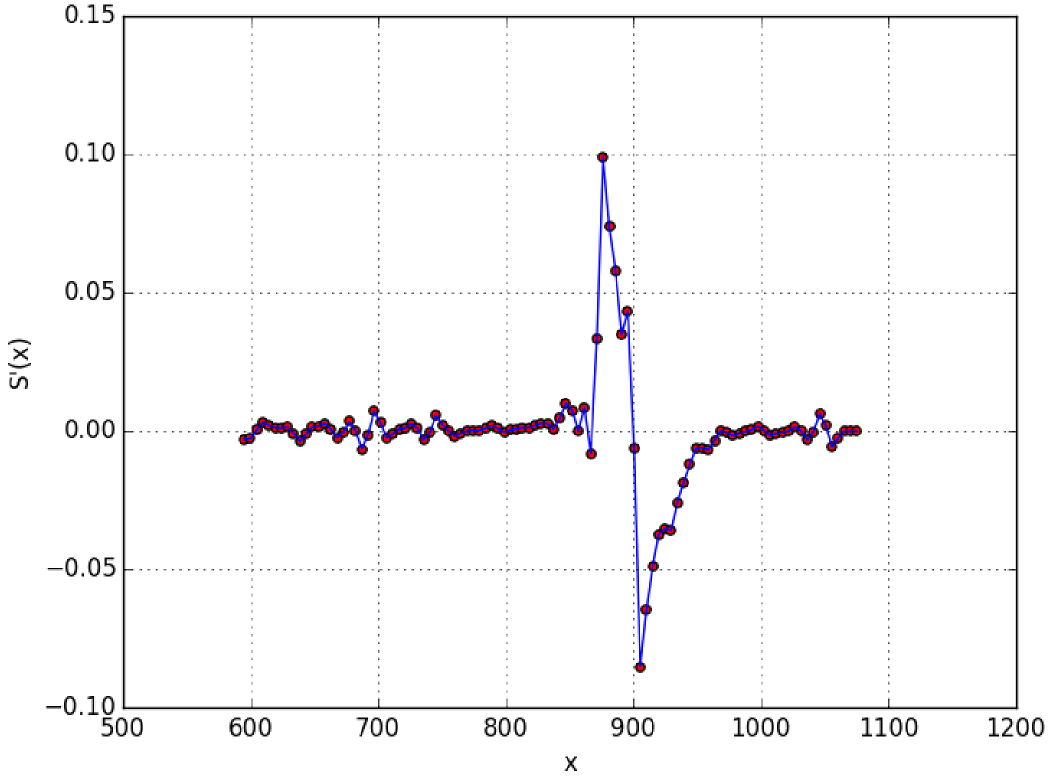

Figure 2: Plot showing the first derivative of the natural cubic interpoland S'(X). Values of $S'(X_i)$ for X_i in the range of (min(knot values), max(knot values)) are shown as red circles.

1.2 Determining the natural cubic interpoland S(x) and its Derivative S'(x) from experimental data

The corresponding plots created by matlab are much more smoother because matlab implements an additional constraint that the third derivatives of the piecewise polynomials are also equal at the second and the last knots. This are apparently called the "Not-A-Knot" end conditions. This is only done when the length of **t** and **y** are same.

Natural splines are a good choice only when the functions have 0 second derivative at the end points. I read it up from here. `http://www.mathworks.com/matlabcentral/newsreader/view_thread/172988` would really appreciate some more information about why using the Not-A-Knot condition is better.
The corresponding matlab code can be found at 3

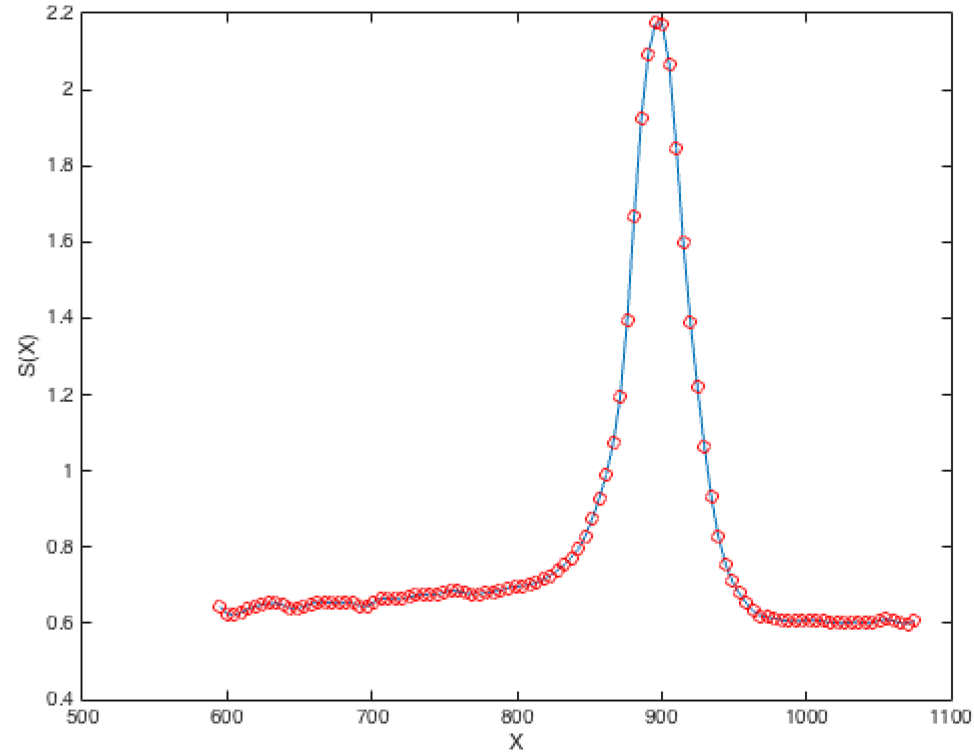

Figure 3: Plot showing the natural cubic interpoland S(X) as calculated using Matlab. Values of $S(X_i)$ for X_i in the range of (min(knot values), max(knot values)) are shown as red circles.

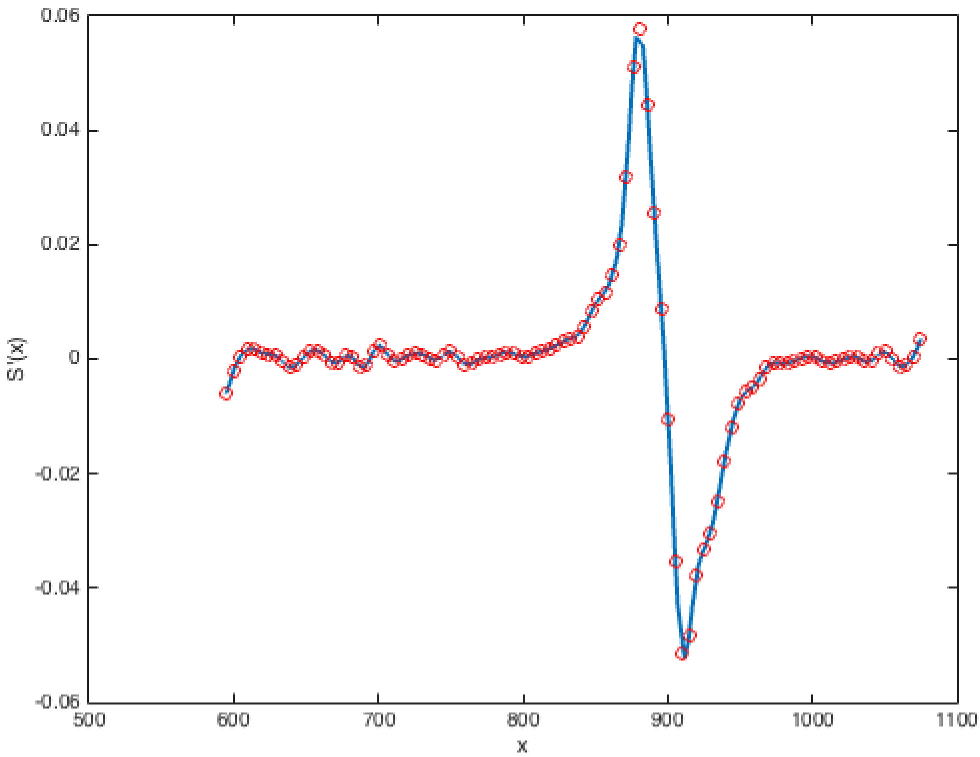

Figure 4: Plot showing the first derivative of the natural cubic interpoland S'(X) as calculated using Matlab. Values of $S'(X_i)$ for X_i in the range of (min(knot values), max(knot values)) are shown as red circles.

2 Appendix A

Python source code for 1.1.

```
from __future__ import division
import numpy as np
import pylab as pl
import string

def get_data(filename):
    retVal = None
    with open(filename) as f:
        retVal = f.readlines()
        retVal = map(string.strip, retVal)
        retVal = map(string.split, retVal)
        retVal = zip(*retVal)
        # Assuming data just has columns of floats
        for idx,_ in enumerate(retVal):
            retVal[idx] = map(float, retVal[idx])
    return retVal

def evaluate(t,y,z,x):
    i = -1
    for idx in range(len(t)-1):
        if x - t[idx] <= 0:
            i = idx
            break
    # Reduce the number of list accesses
    yi = y[i]
    ti = t[i]
    zi = z[i]
    zi1 = z[i+1]
    hi = t[i+1] - ti
    # Evaluate the value of Spline at x
    Bi = -(hi*zi1)/6 -(hi*zi)/3 +(y[i+1]-yi)/hi
    #Common product being used in Ai and Ai_dash
    Ai_dash_common = (x-ti)*(zi1-zi)
    Ai = 0.5*zi + (1/(6*hi))*Ai_dash_common
    Ri = Bi+(x-ti)*Ai
    S = yi + (x-ti)*Ri
    #Calculating the derivative now
    Ai_dash = zi + (0.5/hi)*Ai_dash_common
    S_dash = Bi + (x-ti)*Ai_dash
    return S,S_dash

if __name__ == "__main__":
    t,y = get_data("titanium.dat")
    h = [t[i+1] - t[i] for i in range(len(t)-1)]
    b = [(1/h[i])*(y[i+1]-y[i]) for i in range(len(t)-1)]
    u = [2*(h[0]+h[1])]
    v = [6*(b[1]-b[0])]
    # intermediate points
    for i in range(1,len(y)-1):
        u.append(2*(h[i]+h[i-1]) - (h[i-1]*h[i-1])/u[i-1])
        v.append(6*(b[i] - b[i-1]) - (h[i-1]*v[i-1])/u[i-1])
    z = np.zeros(len(y))
    try:
        for i in range(len(y)-2,0,-1):
            z[i] = (v[i] - h[i]*z[i])/u[i]
    except Exception,e:
        print i,len(y),len(z),len(v),len(u)

    minx,maxx = min(t),max(t)
    x_range = np.linspace(minx,maxx,100)
    S,S_dash = zip(*[evaluate(t,y,z,x) for x in x_range])

    pl.figure()
    pl.plot(x_range,S)
    pl.scatter(x_range,S,c="r")
    pl.grid()
    pl.xlabel("x")
    pl.ylabel("S(x)")
    pl.savefig("plotS.png")
    pl.figure()
    pl.plot(x_range,S_dash,c="b")
    pl.scatter(x_range,S_dash,c="r")
    pl.grid()
    pl.xlabel("x")
    pl.ylabel("S'(x)")
    pl.savefig("plotS_dash.png")
    pl.show()
    pl.close()
```

3 Appendix B

Matlab source code for 1.2.

```
% Getting the data
a = load('titanium.dat')
y = a(:,2)
t = a(:,1)
x = linspace(min(t),max(t),100)

% Calculating and Plotting the spline
plot(x,spline(t,y,x))
hold on
scatter(x,spline(t,y,x),'red')
xlabel('X')
ylabel('S(X)')

% Now Calculating and Plotting derivative of the Spline
f = fnder(spline(t,y))
fnplt(fnder(spline(t,y)))
hold on
scatter(x,ppval(f,x),'r')
xlabel('x')
ylabel('S'(x)')
```