# MATH661 Homework 1 - Univariate approximation

SOLUTION

Posted: Aug 23 Due: 11:55PM, Sep 6

Note: Please read through the solution carefully. The solution procedures are meant to efficiently and correctly solve the assignment, maintaining clarity of presentation

## 1 Problem statement

Approximating a complicated function $f\colon \mathbb{R} \to \mathbb{R}$, by a simpler to compute function $g\colon \mathbb{R} \to \mathbb{R}$ is a fundamental operation within scientific computation. This homework investigates various techniques to solve this problem in the context of describing the 2D curve $(x(t), y(t))$ depicted in Fig. 1, with sample points listed in Table 1.

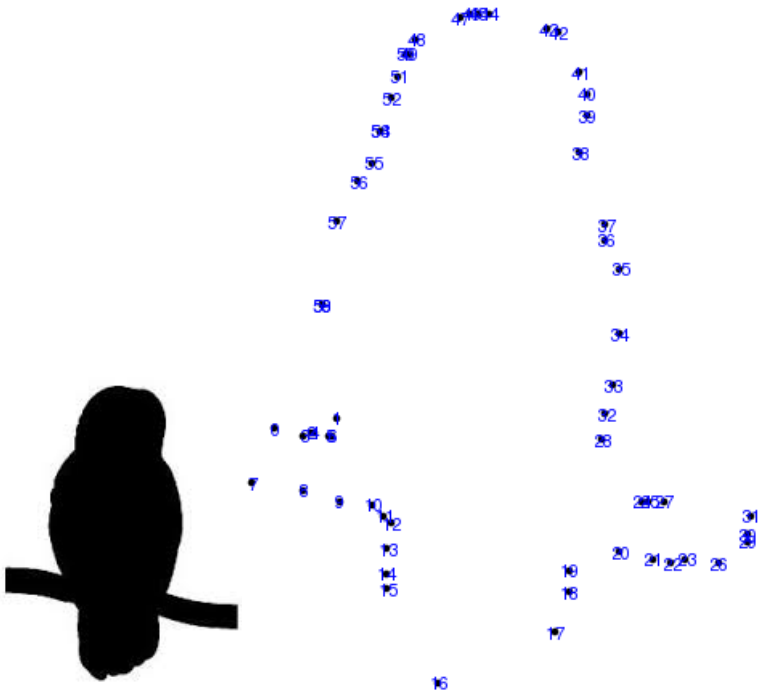

**Figure 1.** Chosen contour: owl silhouette (left), sample points (right)

| | $x_i$ | $y_i$ | $x_i$ | $y_i$ | $x_i$ | $y_i$ | $x_i$ | $y_i$ |
|---|---|---|---|---|---|---|---|---|
| 0 | 27 | 80 | 44 | 83 | 37 | 79 | 35 | 78 |
| 1 | 38 | 79 | 42 | 78 | 43 | 78 | 21 | 65 |
| 2 | 35 | 63 | 45 | 60 | 54 | 59 | 57 | 56 |
| 3 | 59 | 54 | 58 | 47 | 58 | 40 | 58 | 36 |
| 4 | 72 | 10 | 104 | 24 | 108 | 35 | 108 | 41 |
| 5 | 122 | 46 | 131 | 44 | 136 | 43 | 140 | 44 |
| 6 | 128 | 60 | 130 | 60 | 149 | 43 | 134 | 60 |
| 7 | 117 | 77 | 157 | 49 | 157 | 51 | 158 | 56 |
| 8 | 118 | 84 | 120 | 92 | 122 | 106 | 122 | 124 |
| 9 | 118 | 132 | 118 | 136 | 111 | 156 | 113 | 166 |
| 10 | 113 | 172 | 111 | 178 | 105 | 189 | 102 | 190 |
| 11 | 86 | 194 | 83 | 194 | 81 | 194 | 78 | 193 |
| 12 | 66 | 187 | 64 | 183 | 63 | 183 | 61 | 177 |
| 13 | 59 | 171 | 56 | 162 | 56 | 162 | 54 | 153 |
| 14 | 50 | 148 | 44 | 137 | 40 | 114 | 40 | 114 |

**Table 1.** Contour coordinates. Tangent vectors: $\boldsymbol{t}_{16} = (1, 0)$ $\boldsymbol{t}_{34} = (0, 1)$, $\boldsymbol{t}_{58} = (0, -1)$

Processing of the image to obtain sample points is carried out in the attached ContourPoints.nb Mathematica notebook, that also outputs node coordinates as a Matlab format mat file. The data points are loaded into a Python session for processing.

```
Python] from pylab import *; from os import *; from scipy.io import *;
Python] chdir('/home/student/courses/MATH661/homework');
```

```
Python] data=loadmat('/home/student/courses/MATH661/homework/HW1data.mat');
Python] xy=data.values()[0];
Python] xy[0:10].T
    [[ 27.  44.  37.  35.  38.  42.  43.  21.  35.  45.]
    [ 80.  83.  79.  78.  79.  78.  78.  65.  63.  60.]]
Python] x=xy[:,0]; y=xy[:,1];
Python]
```

## 2 Theoretical exercises

1. K&C, 1.2.14, p.26 (include plots of the ODE solution for various $c$, and use Sage and Maxima to try to solve the ODE)

   Solution. Let $F(x,\,y) = 2x^3y^2 + x^2y + e^x - c = 0$, and since $F \in C^1(\mathbb{R}\times\mathbb{R})$, i.e., $F$ is differentiable in both variables, by the implicit function theorem $\mathrm{d}y/\mathrm{d}x$ exists in any open set where $\partial F/\partial y$ is invertible. Differentiation leads to

   $$\mathrm{d}F = \frac{\partial F}{\partial x}\mathrm{d}x + \frac{\partial F}{\partial y}\mathrm{d}y = 0 \Rightarrow \frac{\mathrm{d}y}{\mathrm{d}x} = -\frac{\partial F}{\partial x}/\frac{\partial F}{\partial y}.$$

   Calculate:

   $$\frac{\partial F}{\partial x} = 6x^2y^2 + 2xy + e^x,\, \frac{\partial F}{\partial y} = 4x^3y + x^2,$$

   to obtain

   $$\frac{\mathrm{d}y}{\mathrm{d}x} = -\frac{6x^2y^2 + 2xy + e^x}{4x^3y + x^2},$$

   everywhere except $(x,\,y) = (0,0)$.

   Verify in Mathematica:

   ```
   Mathematica
   In[2]:= F[x_,y_]=2 x^3 y^2 + x^2 y + Exp[x]; -D[F[x,y],x]/D[F[x,y],y]
   ```

   $$-\frac{6\,x^2\,y^2 + 2\,x\,y + e^x}{4\,x^3\,y + x^2}$$

   ```
   In[3]:=
   ```

   Solution notes:

   - The implicit function theorem is extensively used in scientific computation: ensure you are familiar with the hypotheses and conclusions of the theorem
   - Carry out such calculations by hand to reinforce mastery of differentiation of complicated functions. Subsequently verify in symbolic computation packages (e.g., Mathematica, Maple, Sage).
   - *1 bonus extra credit point awarded to recognition of the singular point (0,0).*

2. K&C, 1.2.25, p.26

Solution. By complete induction. General statement:

$$\mathcal{P}(n)\colon p_n(x) = a_0 + a_1 x + \ldots + a_n x^n = a_0 + x(a_1 + \ldots + x(a_{n-1} + x\,a_n)\ldots)$$

Verify $\mathcal{P}(0)$: $p_0(x) = a_0 = a_0$ ✓.

Assume $\mathcal{P}(n)$ is true for any $p_n(x)$. The statement for $\mathcal{P}(n+1)$ is:

$$\mathcal{P}(n+1)\colon p_{n+1}(x) = a_0 + a_1 x + \ldots + a_{n+1} x^{n+1} = a_0 + x(a_1 + \ldots + x(a_n + x\,a_{n+1})\ldots)$$

By $\mathcal{P}(n)$, $q_n(x) = a_1 + a_2 x + \ldots + a_{n+1} x^n = a_1 + x(a_2 + \ldots + x(a_n + x\,a_{n+1})\ldots)$, hence

$$a_0 + x(a_1 + \ldots + x(a_n + x\,a_{n+1})\ldots) = a_0 + x\,q_n(x) = a_0 + a_1 x + \ldots + a_{n+1} x^{n+1} \; q.e.d. \square$$

Solution notes:

- Proofs by induction are frequently used to establish validity of basic algorithms in scientific computing theory. Become familiar with the mechanics:
  - → formally present the statement identifying the induction variable $n \in \mathbb{N}$
  - → verify for small $n$
  - → recognize the pattern to move from $n$ to $n+1$ (if necessary by trying out $n = 2, 3, 4$), and use it to prove $\mathcal{P}(n+1)$
- In above proof, key is to recognize that $a_2 + \ldots + x(a_n + x\,a_{n+1})$ is an $n^{\text{th}}$ degree polynomial to which $\mathcal{P}(n)$ applies.

3. K&C, 1.3.14-16, p.35

Solution. 1.3.14: By definition, $\triangle x_i = x_{i+1} - x_i$, $Ex_i = x_{i+1}$, $Ix_i = E^0 x_i = x_i$. Verify

$$Ex_i = x_{i+1} = (I + \triangle)x_i = x_i + x_{i+1} - x_i = x_i \checkmark$$

for any $x_i$, hence $E = I + \triangle$. Establish that for any polynomial $p \in \Pi_m$

$$\mathcal{P}(m)\colon p(E) = \frac{1}{0!}p_m(I) + \frac{1}{1!}p'(I)\triangle + \frac{1}{2!}p''(I)\triangle^2 + \ldots + \frac{1}{m!}p^{(m)}(I)\triangle^m,$$

by complete induction. Verify for $m = 1$,

$$\begin{array}{ll} & p(E) = a_0 E^0 + a_1 E^1 \\ \mathcal{P}(1)\colon & p(I) + p'(I)\triangle = a_0 I^0 + a_1 I^1 + a_1 \triangle \\ & p(E) = p(I) + p'(I)\triangle \Leftrightarrow a_0 I + a_1 E = a_0 I + a_1 (I + \triangle), \end{array} \tag{1}$$

verified for any $a_0$,$a_1$. Assume $\mathcal{P}(m)$ true for any polynomial of degree $m$. For $m+1$,

$$\mathcal{P}(m+1)\colon p(E) = \frac{1}{0!}p(I) + \frac{1}{1!}p'(I)\triangle + \frac{1}{2!}p''(I)\triangle^2 + \ldots + \frac{1}{(m+1)!}p^{(m+1)}(I)\triangle^{m+1}.$$

4. K&C, 6.1.31-32, p.326

Solution. The Lagrange polynomial is

$$p_n(x) = \sum_{i=0}^{n} \ell_i(x)\, y_i, \ell_i(x) = \frac{\prod_{j=0, j\neq i}^{n} (x - x_j)}{\prod_{j=0, j\neq i}^{n} (x_i - x_j)}. \tag{2}$$

Note the $i^{\text{th}}$ Lagrange polynomial can be written as

$$\ell_i(x) = \frac{\ell(x)}{x - x_i} w_i, \ell(x) = \prod_{j=0}^{n} (x - x_j), w_i = \frac{1}{\prod_{j=0, j\neq i}^{n} (x_i - x_j)} = \frac{1}{\ell'(x_i)},$$

and obtain

$$p_n(x) = \sum_{i=0}^{n} \frac{\ell(x)}{x - x_i} w_i\, y_i = \ell(x) \sum_{i=0}^{n} \frac{w_i\, y_i}{x - x_i}.$$

The barycentric form is obtained by dividing the above with the interpolation of $f(x) = 1$

$$1 = \sum_{i=0}^{n} \frac{\ell(x)}{x - x_i} w_i\, y_i = \ell(x) \sum_{i=0}^{n} \frac{w_i}{x - x_i},$$

leading to

$$p_n(x) = \frac{\sum_{i=0}^{n} \frac{w_i\, y_i}{x - x_i}}{\sum_{i=0}^{n} \frac{w_i}{x - x_i}},$$

a form that requires only $\mathcal{O}(n)$ operations to evaluate versus the $\mathcal{O}(n^2)$ operations required for (2).

5. K&C, 6.1.37, p327. Compare prediction to current prices. Present an analysis of any discrepancy

   Solution.

```
Python] from pylab import *;
Python] def dd(x,y):
          n = size(x)-1
          a=zeros(size(y)); a=copy(y)
          for j in range(1,n+1):
            for i in range(n,j-1,-1):
              a[i] = (a[i]-a[i-1])/(x[i]-x[i-j])
          return a
        def newton(xi,dd,x):
          n = size(xi)-1
          px = zeros(size(x)); q = ones(size(x))
          for i in range(n+1):
            px = px + dd[i]*q
            q = q*(x-xi[i])
          return px
Python] x=array([1885,1917,1919,1932,1958,1963,1968,1971,1974,1978,
        1981+(31+28+15.)/365,1981+(365-31-30-15.)/365,1985,1988,1991,1995,
        1999,2001]);
Python] y=array([2.,3,2,3,4,5,6,8,10,15,18,20,22,25,29,32,33,34]);
Python] c=dd(x,y);
```

```
Python] print c

   [  2.00000000e+00   3.12500000e-02  -1.56250000e-02   1.15077741e-03
     -2.89440714e-05   6.78486768e-07  -1.52159492e-08   8.18727496e-10
     -9.10167759e-11   9.40668614e-12  -9.21878598e-13   1.70466546e-13
     -2.95063652e-14   3.88067800e-15  -3.85086922e-16   2.78456821e-17
     -1.57306784e-18   8.01205780e-20]

Python] xf=arange(2001,2003,0.1); yf=newton(x,c,xf);
Python] yf/100

    [   0.34          2.758225      5.76866736    9.45624828   13.91479256
      19.2477652    25.56905774   33.00382715   41.68939017   51.77617614
      63.42874159   76.8268498    92.16661895  109.6617424   129.54478503
     152.06855951  177.50758673  206.15964471  238.34741047  274.42019974]

Python]
```

Evaluation of the interpolating polynomial outside the data range is known as extrapolation and is fraught with error. In this case the prediction is within a year of 2001 for 1 dollar and 10 dollars.

6. K&C, 6.3.13, p. 349. Repeat for $n = 3, 4$, using symbolic computation.

   Solution. First for just two nodes $n = 1$

```
In[36]:= n=1; xn=Table[Subscript[x,k],{k,0,n}]
```

$\{x_0, x_1\}$

```
In[37]:= l[t_]=Product[(t-xn[[i+1]]),{i,0,n}]
```

$(t - x_0)\,(t - x_1)$

```
In[38]:= w[i_,t_]:=1/(D[l[t],t] /. t->xn[[i+1]]);
         l[i_,t_]:=l[t] w[i,t]/(t-xn[[i+1]]);
         Table[l[i,t],{i,0,n}]
```

$$\left\{\frac{t - x_1}{x_0 - x_1}, \frac{t - x_0}{x_1 - x_0}\right\}$$

```
In[39]:= A[i_,t_]:=(1-2(t-xn[[i+1]]) (D[l[i,t],t] /. t->xn[[i+1]])) l[i,
         t]^2;
         Table[Simplify[A[i,t]],{i,0,n}]
```

$$\left\{-\frac{(t - x_1)^2\,(2\,t - 3\,x_0 + x_1)}{(x_0 - x_1)^3}, \frac{(t - x_0)^2\,(2\,t + x_0 - 3\,x_1)}{(x_0 - x_1)^3}\right\}$$

```
In[40]:= B[i_,t_]:=(x-xn[[i+1]]) l[i,t]^2;
         Table[Simplify[B[i,t]],{i,0,n}]
```

$$\left\{\frac{(x - x_0)\,(t - x_1)^2}{(x_0 - x_1)^2}, \frac{(x - x_1\_)(t - x_0)^2}{(x_0 - x_1)^2}\right\}$$

```
In[41]:=
```

Repeat for other $n$, e.g. $n = 3$

```
In[42]:= n=3; xn=Table[Subscript[x,k],{k,0,n}]
```

$\{x_0, x_1, x_2, x_3\}$

```
In[44]:= l[t_]=Product[(t-xn[[i+1]]),{i,0,n}]
```

$$(t-x_0)\,(t-x_1)\,(t-x_2)\,(t-x_3)$$

```
In[45]:= w[i_,t_]:=1/(D[l[t],t] /. t->xn[[i+1]]);
         l[i_,t_]:=l[t] w[i,t]/(t-xn[[i+1]]);
         Table[l[i,t],{i,0,n}]
```

$$\left\{\frac{(t-x_1)\,(t-x_2)\,(t-x_3)}{(x_0-x_1)\,(x_0-x_2)\,(x_0-x_3)},\ \frac{(t-x_0)\,(t-x_2)\,(t-x_3)}{(x_1-x_0)\,(x_1-x_2)\,(x_1-x_3)},\ \frac{(t-x_0)\,(t-x_1)\,(t-x_3)}{(x_2-x_0)\,(x_2-x_1)\,(x_2-x_3)},\ \frac{(t-x_0)\,(t-x_1)\,(t-x_2)}{(x_3-x_0)\,(x_3-x_1)\,(x_3-x_2)}\right\}$$

```
In[46]:= A[i_,t_]:=(1-2(t-xn[[i+1]]) (D[l[i,t],t] /. t->xn[[i+1]])) l[i,t]^2;
         Table[Simplify[A[i,t]],{i,0,n}]
```

$$\left\{\frac{(t-x_1)^2\,(t-x_2)^2\left(1-2\left(\frac{1}{x_0-x_2}+\frac{1}{x_0-x_3}+\frac{1}{x_0-x_1}\right)(t-x_0)\right)(t-x_3)^2}{(x_0-x_1)^2\,(x_0-x_2)^2\,(x_0-x_3)^2},\right.$$
$$\frac{(t-x_0)^2\,(t-x_2)^2\left(1-2\left(\frac{1}{x_1-x_2}+\frac{1}{x_1-x_3}+\frac{1}{x_1-x_0}\right)(t-x_1)\right)(t-x_3)^2}{(x_0-x_1)^2\,(x_1-x_2)^2\,(x_1-x_3)^2},$$
$$\frac{(t-x_0)^2\,(t-x_1)^2\left(1-2\left(\frac{1}{x_2-x_1}+\frac{1}{x_2-x_3}+\frac{1}{x_2-x_0}\right)(t-x_2)\right)(t-x_3)^2}{(x_0-x_2)^2\,(x_1-x_2)^2\,(x_2-x_3)^2},$$
$$\left.\frac{(t-x_0)^2\,(t-x_1)^2\,(t-x_2)^2\left(1-2\left(\frac{1}{x_3-x_1}+\frac{1}{x_3-x_2}+\frac{1}{x_3-x_0}\right)(t-x_3)\right)}{(x_0-x_3)^2\,(x_1-x_3)^2\,(x_2-x_3)^2}\right\}$$

```
In[47]:= B[i_,t_]:=(x-xn[[i+1]]) l[i,t]^2;
         Table[Simplify[B[i,t]],{i,0,n}]
```

$$\left\{\frac{(x-x_0)\,(t-x_1)^2\,(t-x_2)^2\,(t-x_3)^2}{(x_0-x_1)^2\,(x_0-x_2)^2\,(x_0-x_3)^2},\ \frac{(x-x_1)\,(t-x_0)^2\,(t-x_2)^2\,(t-x_3)^2}{(x_0-x_1)^2\,(x_1-x_2)^2\,(x_1-x_3)^2},\right.$$
$$\left.\frac{(x-x_2)\,(t-x_0)^2\,(t-x_1)^2\,(t-x_3)^2}{(x_0-x_2)^2\,(x_1-x_2)^2\,(x_2-x_3)^2},\ \frac{(x-x_3)\,(t-x_0)^2\,(t-x_1)^2\,(t-x_2)^2}{(x_0-x_3)^2\,(x_1-x_3)^2\,(x_2-x_3)^2}\right\}$$

```
In[48]:=
```

Note: Symbolic computation can alleviate the tedium of manual manipulation of formulas as shown above. Furthermore, the expressions can be translated into TeX, Fortran, C to write documents or computer programs, e.g.

```
In[48]:= TeXForm[B[3,t]]
```

frac{\left(t-x_0right){}^2 left(t-x_1right){}^2 left(t-x_2right){}^2 left(x-x_3right)}{\left(x_3-x_0right){}^2 left(x_3-x_1right){}^2 left(x_3-x_2right){}^2}

# 3 Implementation and analysis

1. Construct the Newton interpolation of the contour and compare to original.
2. Construct the Hermite interpolation of the contour and compare to original.
3. Construct the spline interpolation of the contour and compare to original.
4. Provide an analysis of the interpolation error and compare to your computational results.

## 3.1 Newton interpolant

Define interplation functions

```
Python] def dd(x,y):
          n = size(x)-1
          a=zeros(size(y)); a=copy(y)
          for j in range(1,n+1):
            for i in range(n,j-1,-1):
              a[i] = (a[i]-a[i-1])/(x[i]-x[i-j])
          return a
        def newton(xi,dd,x):
          n = size(xi)-1
          px = zeros(size(x)); q = ones(size(x))
          for i in range(n+1):
            px = px + dd[i]*q
            q = q*(x-xi[i])
          return px
Python]
Python]
```

Import data, and define parameter $t$ as the node number

```
Python] from pylab import *; from os import *; from scipy.io import *;
Python] chdir('/home/student/courses/MATH661/homework');
Python] data=loadmat('/home/student/courses/MATH661/homework/HW1data.mat');
Python] xy=data.values()[0];
Python] xy[0:10].T
```

```
    [[ 27.  44.  37.  35.  38.  42.  43.  21.  35.  45.]
    [ 80.  83.  79.  78.  79.  78.  78.  65.  63.  60.]]
```

```
Python] x=xy[:,0]; y=xy[:,1];
Python] n=size(x); t=arange(0.,n);
Python]
Python]
```

Compute interpolations $x(t)$,$y(t)$, evaluate at additional points, compare to original

```
Python] cx=dd(t,x); cy=dd(t,y); tf=arange(0.,n,0.5);
Python] xf=newton(t,cx,tf); yf=newton(t,cy,tf);
Python] xf[0:8].T
```

```
    [  2.70000000e+01   2.74765883e+15   4.40000000e+01  -7.28999710e+13
      3.70000000e+01   3.28364022e+12   3.50000000e+01  -2.11007169e+11]
```

```
Python]
```

Note that the interpolation conditions are satisfied (e.g., $x(t=0)=27$, $x(t=1)=44$), but evaluation at intermediate points leads to enormous errors.

## 3.2 Hermite interpolation

A simple modification of the divided difference procedure checks for repeated nodes and places the derivative value in the coefficient table.

```
Python] def ddH(x,y,yp,eps=1.0e-6):
          n = size(x)-1
          a=zeros(size(y)); a=copy(y)
          for j in range(1,n+1):
            for i in range(n,j-1,-1):
              if abs(x[i]-x[i-j]) > eps:
                a[i] = (a[i]-a[i-1])/(x[i]-x[i-j])
              else:
                a[i] = yp[i]
          return a
        def newton(xi,dd,x):
          n = size(xi)-1
          px = zeros(size(x)); q = ones(size(x))
          for i in range(n+1):
            px = px + dd[i]*q
            q = q*(x-xi[i])
          return px
Python]
```

Check the procedure on a simple case $f(x)=(x-1)^2$, $f(0)=1$, $f'(0)=-2$, $f(1)=0$

```
Python] xtst=array([0.,0.,1.]); ytst=array([1.,1.,0.]); yptst=array([0.,-2.,0.]);
Python] ctst=ddH(xtst,ytst,yptst);
Python] ctst
    [ 1. -2.  1.]
Python]
```

The Hermite polynomial would be

$$p(x)=1-2(x-0)+1(x-0)(x-0)=1-2x+x^2=(x-1)^2\checkmark$$

Given that the Newton interpolant showed large errors between evaluation points, the Hermite polynomial can be expected to behave similarly (since it's a global interpolant). The tangent vector at $i=0$ seems to be $(0,-1)$.

```
Python] tH=zeros(n+1); tH[1:n+1]=t[0:n]; tH[0]=tH[1];
Python]
Python] xH=zeros(n+1); xH[1:n+1]=x[0:n]; xH[0]=xH[1]; xpH=zeros(n+1);
Python] yH=zeros(n+1); yH[1:n+1]=y[0:n]; yH[0]=yH[1]; ypH=zeros(n+1); ypH[1]=-1.;
Python] cxH=ddH(t,xH,xpH); cyH=ddH(t,yH,ypH); tf=arange(0.,n,0.5);
Python] xfH=newton(tH,cxH,tf); yfH=newton(tH,cyH,tf);
Python] xfH[0:2].T
    [  2.70000000e+01   1.73758751e+78]
Python]
```

Again, contour seems to be correctly interpolated, but large errors appear between nodes.

## 3.3 Spline interpolation

Use the spline interpolation procedure within pylab

```
Python] from scipy.interpolate import interp1d
Python] fx=interp1d(t,x,kind='cubic'); fy=interp1d(t,y,kind='cubic');
Python] tf=arange(1,n-1,0.5);
```

```
Python] plot(fx(tf),fy(tf),'r-',x,y,'ko'); title('Spline interpolation');
Python] xlabel('x'); ylabel('y'); axis('equal');
Python] show();
Python]
```

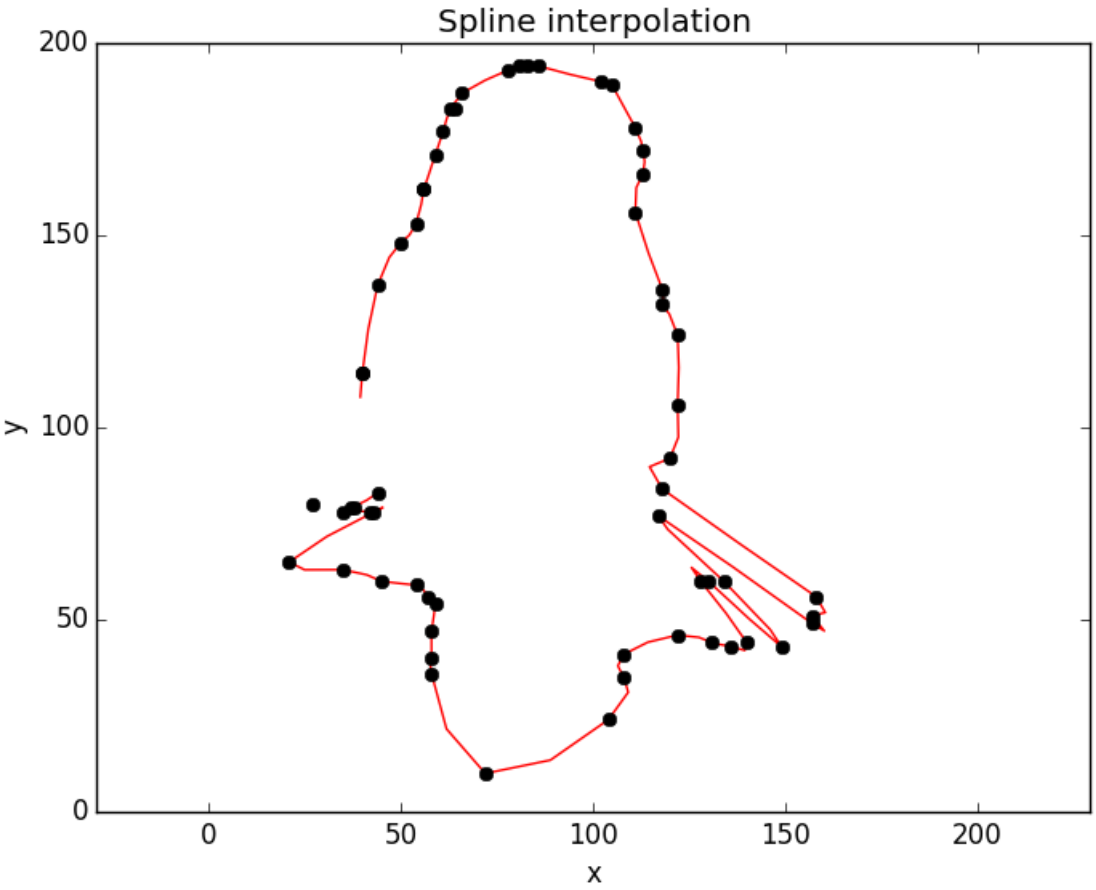


**Figure 2.** Spline interpolation of contour

### 3.4 Analysis

The interpolation error is of form

$$f(x) - p(x) = \frac{f^{(p)}(\xi_x)}{(n+1)!} \prod_{i=0}^{n} (x - t_i)$$

with $p = n + 1$ for Newton, and $p = 2n + 2$ for Hermite interpolation. For $n = 60$ the product can give large values, and the data seems to sample a discontinuous contour (e.g., sharp corners at branch), thus leading to large errors.

The piecewise spline interpolation minimizes curvature between points and an acceptable approximation is obtained.