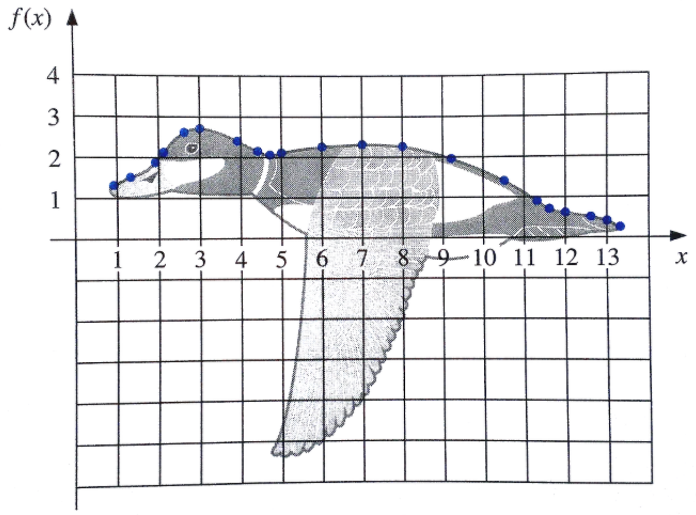
Math475a, Fall 2017. Assignment 5. Due November 2nd

In this homework we will use various methods to approximate the line of duck's back from your textbook. Data for the marked sample points are in the file 'duck.dat'. It is a text file with two columns; first contains the x_n — x-coordinates; second, f_n — y coordinates. A summary of various interpolation ideas that we discussed in class is outlined in the end of this file. *You do not need to program your own solver for linear systems for this assignment and should use the built-in facilities in your programming environment.*



Least Squares. In order to find the best-fitting m-th degree polynomial, $p_m(x) = a_0 + a_1 x + \cdots + a_m x^m$, first construct the matrix

$$X = \begin{pmatrix} 1 & x_1 & x_1^2 & \dots & x_1^m \\ 1 & x_2 & x_2^2 & \dots & x_2^m \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_N & x_N^2 & \dots & x_N^m \end{pmatrix}.$$

Then solve the *least squares* problem,

$$X^T X \boldsymbol{a} = X^T \boldsymbol{f},$$

where the unknown vector $\boldsymbol{a}$ contains the coefficients a_k, $k = 0, \dots, m$; and vector $\boldsymbol{f}$ contains the y-coordinates of the data. Plot your approximations for $m = 5, 10, 15, 20$ (use the same plot and display the original data points as well).

Interpolating Polynomials. The least squares approximation with $m = 20$ is, in fact, the exact interpolating polynomial which passes through all the data points. Verify that you get the same approximation using Lagrange interpolation (as in the previous assignment). In your opinion, does it provide a good approximation?

Cubic Splines. Construct cubic splines for the same data and plot the resulting approximation. Use the *natural splines* with free boundary conditions, i.e., the second derivatives of the constructed function must vanish at the end points. The outline below contains some of the theory behind the cubic spline approximation, however it does not have the two equations coming from the vanishing second derivative condition at the end points, x_1 and x_N. Part of your task is to carry out this derivation on your own.

Summary: Interpolating Polynomials

Take some N points, $x_1 < \cdots < x_N$; consider a function $f \in \mathbf{C}^1[x_1, x_N]$. Let $f_n = f(x_n)$, $g_n = f'(x_n)$; $n = 1, \ldots, N$. Let us look at various (piecewise) polynomial approximations to $f(x)$.

Lagrange Polynomial matches the values f_n, but not (necessarily) the derivatives:

$$\hat{f}_{\mathrm{L}}(x) = \sum_{n=1}^{N} f_n p_n(x), \quad \text{where} \quad p_n(x) = \prod_{\substack{k=1 \\ k \neq n}}^{N} \frac{x - x_k}{x_n - x_k}. \tag{1}$$

Osculating (Hermite) polynomial matches the values f_n and the derivatives, g_n:

$$\hat{f}_{\mathrm{H}}(x) = \sum_{n=1}^{N} f_n \left[1 - 2(x - x_n) \sum_{\substack{k=1 \\ k \neq n}}^{N} \frac{1}{x_n - x_k} \right] p_n^2(x) \; + \; \sum_{n=1}^{N} g_n (x - x_n) p_n^2(x). \tag{2}$$

Here the polynomials $p_n(x)$ are as defined in equation (1).

Cubic Splines. First let us construct four "building-block" polynomials as defined in equation (2) if we set $N = 2$, $x_1 = 0$, $x_2 = 1$(using t instead of x):

$$p_0(t) = (1 + 2t)(t - 1)^2; \quad p_1(t) = t^2(3 - 2t); \quad q_0(t) = t(t - 1)^2; \quad q_1(t) = t^2(t - 1). \tag{3}$$

These polynomials have the following special properties:

$$\begin{array}{llllllll} p_0(1) = 1, & p_0(1) = 0, & p_0'(0) = 0, & p_0'(0) = 0; & p_1(1) = 0, & p_1(1) = 1, & p_1'(0) = 0, & p_1'(0) = 0; \\ q_0(1) = 0, & q_0(1) = 0, & q_0'(0) = 0, & q_0'(0) = 0; & q_1(1) = 0, & q_1(1) = 0, & q_1'(0) = 0, & q_1'(0) = 0. \end{array}$$

We can use these polynomials to construct an osculating (matching the values f_n and derivatives g_n) piecewise cubic approximation to $f(x)$. Namely, for $x \in [x_n, x_{n+1}]$, set

$$\hat{f}_{\mathrm{CS}}(x) = f_n p_0(t) \; + \; f_{n+1} p_1(t) \; + \; (x_{n+1} - x_n)\Big[g_n q_0(t) \; + \; g_{n+1} q_1(t) \Big], \quad \text{where} \quad t = \frac{x - x_n}{x_{n+1} - x_n}. \tag{4}$$

Note that the function $t(x)$ is different for every sub-interval $[x_n, x_{n+1}]$. The approximation $\hat{f}_{\mathrm{CS}}(x)$ constructed this way is continuously differentiable on the entire domain $[x_1, x_N]$, however it is not twice differentiable. We can construct a $\mathbf{C}^2[x_1, x_N]$ approximation as well if we do not require matching the values of derivatives, g_n.

In order to do this we must find the values of g_n in (4) so that the second derivatives of $\hat{f}_{\mathrm{CS}}(x)$ on both sides of every x_n, $n = 2, \ldots, N - 1$ are equal. Differentiating (4) with respect to x twice, we get that for $x \in [x_n, x_{n+1}]$,

$$\Delta_n^2 \hat{f}_{\mathrm{CS}}''(x) = f_n(12t - 6) \; + \; f_{n+1}(6 - 12t) \; + \; \Delta_n\Big[g_n(6t - 4) \; + \; g_{n+1}(6t - 2) \Big], \quad \text{where} \quad \Delta_n = x_{n+1} - x_n. \tag{5}$$

After doing some algebra, we can derive that the matching condition at x_n may be written as

$$\Delta_n g_{n-1} \; + \; 2(\Delta_{n-1} + \Delta_n) g_n \; + \; \Delta_{n-1} g_{n+1} \; = \; \frac{3\Delta_n}{\Delta_{n-1}}(f_{n-1} - f_n) \; + \; \frac{3\Delta_{n-1}}{\Delta_n}(f_{n+1} - f_n). \tag{6}$$

This condition should hold for every $n \in \{2, \dots, N-1\}$ providing $N-2$ linear equations for N unknowns g_n; thus we need to supply two more equations. These may be prescribed in various manners, e.g., the so-called *free boundary* corresponds to setting $f''_{\mathrm{CS}}(x_1) = f''_{\mathrm{CS}}(x_N) = 0$. *Exercise: use equation* (5) *setting $n = 1$ and $n = N-1$ to derive the two corresponding equations for g_n. These equations will involve g_1, g_2, g_{N-1} and g_N.*

Finally, observe that equations (6) may be further simplified if all Δ_n are equal (to Δ):

$$g_{n-1} + 4g_n + g_{n+1} = \frac{3}{\Delta}(f_{n-1} - 2f_n + f_{n+1}), \qquad n = 2, \dots, N-1. \tag{7a}$$

The free boundary conditions in this case will give the following two equations:

$$2g_1 + g_2 = \frac{3}{\Delta}(f_2 - f_1), \qquad g_{N-1} + 2g_N = \frac{3}{\Delta}(f_N - f_{N-1}). \tag{7b}$$

Once these equations are solved for g_n, formula (4) will provide a $\mathbf{C}^2[x_1, x_N]$ approximation to $f(x)$. (Observe that in this case we can solve equations (7) for Δg_n rather than for g_n-s themselves.)