Scientific Computing, Spring 2012
Assignment IV: Function Approximation

Aleksandar Donev
Courant Institute, NYU, donev@courant.nyu.edu

Posted March 28th, 2012
Due **Thursday April 12th**, 2012

A total of 100 points is possible (100 points = A+) with up to 25 extra credit points.

1 [50 points] Least Squares polynomial approximations

In this problem we will consider approximating a given non-linear function $f(x) = \exp(x)$ on the interval $x \in [0, 1]$ with a polynomial of degree n,

$$p(x) = \sum_{i=0}^{n} a_i x^i.$$

1.1 [20pts] Least Squares "interpolation"

In class and the last homework we discussed least-squares fitting, where we evaluate the function $f(x)$ on a sequence of $m + 1 \geq n$ equally-spaced nodes with $x_j = j/m$, $j = 0, 1, \ldots, m$, and then try to minimize the Euclidean norm of the residual:

$$\boldsymbol{a}^{(m)} = \arg\min_{\boldsymbol{a}} e_m(\boldsymbol{a}) = \arg\min_{\boldsymbol{a}} \sum_{j=0}^{m} \left[f(x_j) - p_m(x_j)\right]^2.$$

This gives a least-squares fit polynomial $p_m(x) = \sum_{i=0}^{n} a_i^{(m)} x^i$ which depends on how many nodes were used. It is similar to interpolation, except that we do not try to force the polynomial to pass through all of the nodes, but rather, to pass as close as possible to the interpolation nodes. We thus call this "least-squares interpolation", and it is implemented in MATLAB's *polyfit* function.

[7.5 pts] Write down the linear system (normal equations) for finding the polynomial coefficients $\boldsymbol{a}_m^\star$ explicitly, that is, write down a formula for the matrix of the linear system and the right hand side for a general n, m and $f(x)$.
Hint: To find the minimum of the error $e_m(\boldsymbol{a})$ you need to look for critical points, i.e., solutions of the system of equations $\partial e_m / \partial \boldsymbol{a} = 0$.

Note: You may write this by hand and scan into the PDF or turn it in on paper during class.

[7.5pts] Now solve these equations numerically for $f(x) = \exp(x)$ for a given n and m, for example, $n = 5$ and $m = 10$, and compare to MATLAB's function *polyfit* to check your solution.

[5pts] Fix $n = 5$ and see how the error $f(x) - p_m(x)$ behaves as you increase m, that is, check whether adding nodes changes the polynomial approximation significantly.
Hint: Make a plot of the error $\epsilon_m(x) = f(x) - p_m(x)$ for several increasing m. The results for a fixed-degree least squares polynomial will be very different from the case of polynomial interpolation (see problem 2 below), where the degree of the polynomial grows with m also.

1.2 [30pts] Least Squares approximation

One may object to the least-squares fitting approach because only the error at the nodes is minimized, and it may be that the error is large at other points in the inverval $[0, 1]$. Possibly a better approach to approximating the function with a polynomial is a "least-square approximation" where we minimize the functional Euclidean (L_2) norm of the error,

$$\boldsymbol{a}^\star = \arg\min_{\boldsymbol{a}} e(\boldsymbol{a}) = \arg\min_{\boldsymbol{a}} \int_{x=0}^{1} \left[f(x) - p(x)\right]^2 dx.$$

The resulting approximation $p(x) = \sum_{i=0}^{n} a_i^\star x^i$ will presumably spread out the approximation error more evenly over the whole interval instead of focusing on just the interpolation nodes.

[15pts] Write down a linear system for finding the polynomial coefficients $\boldsymbol{a}^\star$ explicitly, for a given function $f(x)$. *Hint: You may encounter the ill-conditioned Hilbert matrix from the second homework.*
Note: You may write this by hand and scan into the PDF or turn it in on paper during class.

[10pts] Solve the above system numerically for $n = 5$ and $f(x) = \exp(x)$ and compare the solution $p(x)$ to the function $f(x)$.

Hint: The following explicit formula may be useful:

$$\int_0^1 x^j \exp(x) dx = (-1)^{j+1}\, j! + e \sum_{i=0}^{j} (-1)^{j-i} \frac{j!}{i!},$$

where $i!$ denotes the factorial of i, obtained as $factorial(i)$ in MATLAB, and $e = \exp(1)$.

[5pts] On the same plot, compare the error $f(x) - p_m(x)$ from the last part of question 1.1 (i.e., use a "large" m) to the error $f(x) - p(x)$, and comment on what you observe. Explain the result.

2 [40 points] Polynomial interpolation and [10 + 25 points] Numerical integration

In this problem we consider interpolating the following (periodic) function on the interval $x \in [-\pi, \pi]$,

$$f(x) = \exp\left(a \cos x\right),$$

where $a = 3$ is a parameter.

The goal of this exercise is to see whether and how fast the interpolation error converges to zero as the number of interpolation nodes increases, for several different types of interpolants:

1. Global polynomial $p_{\text{glob}}(x)$ of degree n, as obtained using MATLAB's *polyfit*.
2. Piecewise linear $p_1(x)$ interpolant, as obtained using MATLAB's *interp*1 function.
3. Piecewise cubic spline $p_3(x)$, as obtained using MATLAB's built-in function *spline* or, equivalently, *interp*1.

For all interpolants, we will use a grid of $n+1$ equi-spaced nodes in the interval $[-\pi, \pi]$. For this assignment we will not use the fact that the function is periodic when constructing the approximations, although this is important in practice (see forthcoming lecture on the Fourier Transform).

For a given interpolant $\phi(x)$, we can evaluate the interpolant on a fine grid of points, for example, $\tilde{x} = linspace(-\pi, \pi, N+1)$ for $N = 1000$, and then compare $\phi(x)$ to the actual function $f(x)$ visually. A quantitative measure of the approximation error is the L_2 norm of the error

$$E_2\left[\phi(x)\right] = \left[\int_{-\pi}^{\pi} \left|f(x) - \phi(x)\right|^2 dx\right]^{1/2}, \tag{1}$$

or the L_1 norm

$$E_1\left[\phi(x)\right] = \int_{-\pi}^{\pi} \left|f(x) - \phi(x)\right| dx. \tag{2}$$

Because we do not know how to do these integrals analytically, we can make our life simpler by using the L_∞ norm instead, and estimate the error using the fine grid of points as

$$E_\infty\left[\phi(x)\right] = \max_{-\pi \le x \le \pi} \left|f(x) - \phi(x)\right| \approx \max_{i=1,\ldots N} \left|f(\tilde{x}_i) - \phi(\tilde{x}_i)\right|.$$

2.1 [10pts] Comparing different interpolants

[7.5pts=2.5pts for each of p_{glob}, p_1 and p_3] For a given small n, say $n = 8$, plot the different interpolants together with the function.

[2.5pts] Plot the error $\varepsilon(x) = |f(x) - \phi(x)|$ of the different interpolants for a larger n, say $n = 32$, to visually compare the accuracy of the different interpolants. Comment on your observations.

2.2 [30pts] Interpolation Error

[10pts] For different numbers of nodes, $n = 2^k$, $k = 2, 3, \ldots$, compute the estimated interpolation error E_∞ for $p_1(x)$ and $p_3(x)$ and then plot the error versus n using an appropriate scaling of the axes.

[7.5pts] For $p_1(x)$ and $p_3(x)$, the theoretical estimates from class suggest that (for either E_2 or E_∞)

$$E\left[p_1(x)\right] \approx C_1 n^{-2} \text{ and } E\left[p_3(x)\right] \approx C_3 n^{-4}.$$

Verify this scaling from the plot of E_∞ versus n.
Hint: Assume that the error scales as $E \approx C\, n^p$, where p is some unknown power exponent. Then $\log E = \ln C + p \log n$ is a linear relation between the logs, with slope p.

[12.5pts] If we did not know how the error changes with n, we can assume that $E = C\, n^p$ and then extract the constant C and the exponent p by fitting the numerical data with a power law. The simplest way to do this, based on the hint above, is to fit a linear line through the points $(\log n, \log E_\infty)$ by using the function *polyfit*. Do this and extract the power exponents from the fit, and compare them to the theoretical power exponents -2 and -4.

2.3 [10pts + 25pts extra credit] Estimating the L_2 approximation error

Obtaining an estimate of the L_2 error E_2 requires evaluating the integral (1), similarly for the L_1 error we need to integrate (2). In this problem, we will fix $n = 10$ and consider using numerical integration to estimate the value of $E_2\left[p_3(x)\right]$ for the spline interpolant, but if you write the code correctly (using function handles), you can easily switch to another norm or another interpolant.

2.3.1 [10pts] Numerical Quadrature

Compute and report $E_2\left[p_3(x)\right]$ estimated using numerical integration via the

1. [2.5pts] Midpoint rule.
2. [2.5pts] Composite trapezoidal rule.
3. [5pts] Composite Simpson's rule.

In each case, use a step size $h = 2\pi/100$ for the numerical integration.

2.3.2 [10pts extra credit] Accuracy of the Numerical Quadrature

For $n = 10$, compare the approximations of E_2 obtained using several different step sizes h for the numerical integration. From the data, can you tell how fast the numerical integration error decreases as the step size is reduced, e.g., is the numerical integration error $O(h)$ or $O(h^2)$?

Note that the results you obtain may be surprising and not in agreement with the theory presented in class. You can try doing the same calculation for $E_2[p_{\text{glob}}(x)]$ and $E_1[p_{\text{glob}}(x)]$ and you might obtain different results. Do not worry about this, and simply give the best answer you can get for the order of convergence of the three methods for *this problem* (rather than in general).

2.3.3 [15pts extra credit] Adaptive quadrature

[10pts] Using MATLAB's adaptive Gauss integration routine *quadl*, obtain an estimate of $E_2\left[p_3(x)\right]$ accurate to 12 digits. How many function-evaluations did MATLAB's routine need to get this accuracy? [*Hint: The help pages suggest using* $[q, fcnt] = quadl(\ldots)$].

[5pts] How many function evaluations do you need to get the same accuracy using the composite trapezoidal rule? Is the answer surprising to you in any way (e.g., is it much larger or much smaller than you expected)?