Scientific Computing, Fall 2020
Assignment V: Function Approximation

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

Posted October 29th 2020
Due **Sunday Oct 22nd 2020**

For grading purposes the maximum is considered to be 110 points (10 additional extra points possible).

1 [55 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,

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

1.1 [20pts] Least Squares fitting

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 m were used. It is similar to interpolation ($m = n$), 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 can call this "least-squares polynomial interpolation", and it is implemented in MATLAB's *polyfit* function.

[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 and the right hand side of the linear system.
Hint: You can do this by relying on overdetermined linear systems, as we did in an earlier homework. But you can also use what you learned about optimization: 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 $\partial e_m/\partial \boldsymbol{a} = 0$, which will directly give you a system of equations for the polynomial coefficients. It is good if you explore and understand the relation between the two.

[5pts] Now solve the equations you derived above 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.

[10pts] 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 (error in the) polynomial approximation significantly. Does it appear that there is some limit as $m \to \infty$?
Hint: Make a plot of the error $\epsilon_m(x) = f(x) - p_m(x)$ for several increasing m. Plot the error as a function on a fine grid with many points, say 1000, and study it; there is no need to compute norms but rather to examine the results and see what you find. Note that 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 $n = m$ grows also.

1.2 [35pts] 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,

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

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 $f(x)$. *Note: You may write this by hand and scan into the PDF.*
Hint: You may encounter the ill-conditioned Hilbert matrix from the second homework.
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 $e = \exp(1)$.

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

[10pts] 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 [60+5 points] Convergence of Interpolating Polynomials

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

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

where a is a given parameter that determines the smoothness of the function; for this assignment fix $a = 3$. Note that this function is periodic, but we don't have to use or know that if we only look at the interval $[-\pi, \pi]$, but we could if we wanted to since it tells us something about the function outside of the interval (i.e., it helps us not only interpolate but also to extrapolate the function).

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 $p_{\text{equi}}(x)$ of degree n with $n + 1$ equi-spaced nodes, as obtained using MATLAB's *polyfit*. Note that due to periodicity the values of the function at the first and last nodes will be equal, but that the polynomial itself does not recognize the periodicity.
2. Piecewise linear $p_1(x)$ interpolant on $n + 1$ equi-spaced nodes, as obtained using MATLAB's *interp*1 function.
3. Piecewise cubic (periodic) spline $p_3(x)$ on $n + 1$ equi-spaced nodes, as obtained using MATLAB's spline toolbox function *csape*:

   ```
   p=csape(x,y,'periodic'); % Find the periodic spline
   y_tilde=fnval(p,x_tilde); % Evaluate on fine grid
   ```

   If you do not have access to the spline or curve-fitting toolbox, you can use MATLAB's built-in function *spline* or, equivalently, *interp*1, but you will not get a periodic interpolant, which will probably increase the error somewhat near the endpoints.
4. An orthogonal polynomial $p_{\text{ortho}}(x)$ interpolant: Choose one from the two options given below and report which one you chose.

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 to the actual function $f(x)$. We can also compute an estimate for the Euclidean norm of the interpolation error by summing the error over the fine grid,

$$E_2\left[\phi(x)\right] = \left[h \sum_{i=0}^{N} \left|f(\tilde{x}_i) - \phi(\tilde{x}_i)\right|^2\right]^{1/2} \approx \left[\int_{-\pi}^{\pi} \left|f(x) - \phi(x)\right|^2 dx\right]^{1/2},$$

where $h = 2\pi/N$.

Choices of orthogonal polynomials

Chebyshev Instead of uniformly spaced nodes, use the Chebyshev nodes

$$x_i = \pi \cos\left(\frac{2i-1}{2\left(n+1\right)}\pi\right), \quad i = 1, \ldots, n+1,$$

and then compute a global polynomial interpolant. Here it is OK to use MATLAB's *polyfit* function but you should know that this will run into ill-conditioning for large n and there are much better and faster methods to do Chebyshev interpolation using the FFT. (In fact, Chebyshev nodes are uniform nodes in the variable θ where $x = cos\theta \in [-1, 1]$ so there is a close-link between Fourier series and Chebyshev polynomials.) Note that this does not use or see the fact the function is periodic.

Fourier The Fourier (trigonomentric) *periodic* interpolant $p_{\mathrm{FFT}}(x)$ on n equi-spaced nodes. [*Hint: Note that for Fourier transforms periodicity is already assumed so you should include only one of the points $x = -\pi$ and $x = \pi$, not both.*] Note that this only works for periodic functions.

2.1 [30+5pts] Comparing different interpolants

[7.5pts for p_{equi}, p_1 and p_3, and p_{ortho}, extra 5 points if you do Fourier]
For a given small n, say $n = 8$, plot the different interpolants together with the function and see how good they are. Plot the error $\varepsilon(x) = |f(x) - \phi(x)|$ of the different interpolants for a larger n, say $n = 32$, and visually compare the accuracy of the different interpolants in different regions of the interval. Discuss the results and relate to the theory covered in class. [*Hint: For the Fourier polynomial, MATLAB's function interpft may be useful to interpolate the Fourier series on the fine grid of nodes.*]

2.2 [30pts] Interpolation Error

[10pts] For different numbers of nodes, $n = 2^k$, $k = 2, 3, \ldots$, compute the estimated interpolation error E_2 for $p_1(x)$ and $p_3(x)$ and then plot the error versus n using an appropriate scaling of the axes.

[10pts] For $p_1(x)$ and $p_3(x)$, the theoretical estimates from class suggest that

$$E_2\left[p_1(x)\right] \approx C_1 n^{-2} \text{ and } E_2\left[p_3(x)\right] \approx C_3 n^{-4}.$$

Verify this scaling from the plot of E_2 versus n.
Hint: Assume that the error scales as $E_2 \approx C\, n^p$, where p is some unknown power exponent. Then $\log E_2 = \ln C + p \log n$ is a linear relation between the logs, with slope p. Therefore, plotting on a log-log scale will show the desired scaling.

[10pts] For $p_{\mathrm{ortho}}(x)$, theory suggests that for this sort of smooth function (specifically, exponentially-decaying Fourier coefficients) convergence is spectral, i.e., faster than any power law, something close to exponential,

$$E_2\left[p_{\mathrm{ortho}}(x)\right] \sim \exp(-n).$$

Verify that your numerical results are consistent with this prediction [*Hint: The spectral convergence is so fast that numerical roundoff will not permit really seeing the exponential decay well, but simply plotting an exponentially-decaying curve on the same plot or plotting the error on a log-linear scale will do*].