Numerical Methods I, Fall 2010
Assignment V: Polynomial Interpolation and Approximation

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

Nov 11th, 2010
Due: November 28th, 2010

1 [35pts] Least Squares polynomial approximations

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

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

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

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

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.

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 finding the least-square approximation is to minimize the functional Euclidean (L_2) norm of the error,

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

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 and $e = \exp(1)$.

1.1 [20pts] Least Squares interpolation

[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 $f(x)$.
[10pts] 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. Fix $n = 5$ and check how large m needs to be before the error $f(x) - p_m(x)$ stops changing substantially with m [Hint: Make a plot of the error for increasing m and see whether it converges as m grows].
[5pts] For several large n, choose $m = 2n$ and see how ill-conditioned the linear system you get is.

1.2 [15pts] Least Squares approximation

[10pts] Write down a linear system for finding the polynomial coefficients $\boldsymbol{a}^\star$ explicitly [*Hint: You may encounter the ill-conditioned Hilbert matrix from the second homework*].
[5pts] Solve the above system numerically for $n = 5$ and compare the solution $p(x)$ to the solution $p_m(x)$ from part 1 above. On the same plot, compare the error $f(x) - p(x)$ with $f(x) - p_m(x)$ for a large m.

2 [105 points] Convergence of Interpolating Polynomials

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

$$f(x) = \frac{\exp(a \cos x)}{2\pi I_0(a)},$$

where a is a given parameter that determines the smoothness of the function, and $I_0(a)$ is a Bessel function used for normalization purposes, available in MATLAB as $besseli(0, a)$. For the majority of this assignment fix $a = 3$, for example.

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 $p_{equi}(x)$ 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. [*Hint: Note that the calculation used by polyfit is ill-conditioned so you may not be able to get a reasonable answer from MATLAB for large n, which is fine as long as you are aware of it*].
2. The Gauss-Legendre polynomial $p_{GL}(x)$ of degree n, where the $n+1$ nodes are the Gauss points. The Gauss points and weights on the interval $[-1, 1]$ can be obtained using the function $[x, w] = GLNodeWt(n+1)$, where the MATLAB script *GLNodeWt.m* is available on the course assignment webpage. You can use MATLAB's *polyfit* here as well, even though there are numerically much more stable ways based on the orthogonality of the Legendre polynomials.
3. Piecewise linear $p_1(x)$ interpolant on $n+1$ equi-spaced nodes, as obtained using MATLAB's *interp*1 function.
4. 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 toolbox, you can use MATLAB's built-in function *spline* or, equivalently, *interp*1, but you will not get a periodic interpolant.
5. The Fourier (trigonomentric) interpolant $p_{FFT}(x)$ on n equi-spaced nodes. [*Hint: Note that for Fourier transforms periodicity is already assumed so you should include only one of the points $x = 0$ and $x = \pi$, not both.*]

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,

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

where $h = 2\pi/N$.

2.1 [35pts] Comparing different interpolants

[30pts=5pts for each of p_{equi}, p_{GL} and $p_{1/3}$, and 10 pts for p_{FFT} if you do not use *interpft* or 5 if you do] 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. [*Hint: For the Fourier polynomial, you can write your own code for evaluating it at the fine grid $\tilde{x}$ by simply writing down the explicit form of the Fourier series, or, with some ingenuity, you can do it faster by doing an inverse FFT, as done in MATLAB's function interpft. Pay close attention to the ordering of the frequencies and use an odd number of points n and also the built-in function fftshift to make it more manageable.*]

[5pts] Plot the magnitude of the Fourier coefficients (i.e., the Fourier spectrum) $\left|\hat{f}\right|$ versus frequency for a large n (large here means that the high-frequency components become smaller than roundoff so that further increasing n does not make a difference numerically).

2.2 [25pts] Interpolation Error

[10pts] For different numbers of nodes, $n = 2^k$, $k = 2, 3, \ldots$, compute the estimated interpolation error E_2 for each of the interpolants and then plot the error versus n. Comment on how fast the error converges and which interpolant is most accurate.

[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 power-law scaling from the plot by using a log-log scale. Even better, fit a linear line through the points $\log(n)$ and $\log E_2$ using the function $polyfit$ and extract the coefficients C_1 and C_4 and the power exponents from the fit (these will be used in part 4 below).

[5pts] For $p_{GL}(x)$ and $p_{FFT}(x)$, 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,

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

Verify that your numerical results are consistent with this prediction [*Hint: The spectral convergence is soo 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*].

2.3 [25pts] Approximating derivatives

An approximation to the first derivative $f'(x)$ can be obtained by simply differentiating the interpolant, $f'(x) \approx \phi'(x)$.

[15pts] For the piecewise linear and cubic interpolants as well as the Fourier interpolant, compute $\phi'(x)$ and compare to the correct derivative $f'(x)$ on the same plot, for some n. [*Hint: For the spline, if you have access to the spline toolbox, you can use $fnder(p, 1)$ to obtain the derivative. If you do not have the toolbox, you can skip the spline and get the same score, though it is possible to also work with piecewise polynomials in MATLAB including differentiating them.*]

[10pts] Compute error estimates for the derivative and plot how they depend on n, and compare to theoretical expectations. [*Hint: One can use the same way of estimating errors as in part 2 above, even though using a fine grid of points is not necessary for the derivatives as it is for the function itself.*]

2.4 [20pts] Constants in error estimates

The most important thing about how the error converges as one takes more nodes is the power-law or exponential dependence on n. It is however also useful to know how the constants C_1 and C_3 (see part 2 above) depend on the magnitude of the derivatives of the function, which for this functional form is encoded in the parameter a: The larger a the more peaked the function becomes and thus the more nodes we are likely to need.

[10pts] Calculate C_1 and C_3 for several parameters a (e.g, $a = 1, 3, 5, 7$) and estimate how they grow with a (e.g., linearly or quadratically).

[10pts extra credit] Compare the results to theoretical expectations [*Hint: This requires obtaining estimates of the norm of the second and fourth derivatives of $f(x)$, which can be done numerically if you cannot do it analytically*].