Math 128A, Fall 2014, Wilkening
Programming Assignment 3 (of 4)
due Wed, Nov 19

In this assignment we will study good and bad ways to compute quadrature rules and evaluate orthogonal polynomials. First some theory. The monic Legendre polynomials $p_n(x)$ are monic (i.e. the leading term is x^n) and satisfy $\langle p_n, p_m\rangle = 0$ if $n \neq m$, where $\langle f, g\rangle = \int_{-1}^{1} f(x)g(x)\,dx$. It is easy to show that

$$p_n(x) = \frac{n!}{(2n)!}\frac{d^n}{dx^n}(x^2-1)^n \tag{1}$$

has these properties: the leading term is $n!/(2n)!(d/dn)^n x^{2n} = x^n$ and integration by parts n times gives

$$\langle p_n, p_m\rangle = (-1)^n \frac{n!m!}{(2n)!(2m)!}\int_{-1}^{1}(x^2-1)^n \underbrace{\frac{d^{m+n}}{dx^{m+n}}(x^2-1)^m}_{0}\,dx = 0, \qquad (n > m).$$

We also have

$$\langle p_n, p_n\rangle = (-1)^n\frac{n!^2}{(2n)!^2}\int_{-1}^{1}\underbrace{(x-1)^n(x+1)^n}_{(x^2-1)^n}\underbrace{\frac{d^{2n}}{dx^{2n}}(x^2-1)^n}_{(2n)!}\,dx = \frac{n!^2}{(2n)!}\cdot\frac{n!}{(n+1)\cdots(2n)}\int_{-1}^{1}(1+x)^{2n}\,dx,$$

which simplifies to $c_n = \langle p_n, p_n\rangle = \dfrac{n!^4}{(2n)!^2}\cdot\dfrac{2^{2n+1}}{2n+1}$. Since $p_{n+1}(x)$ and $p_n(x)$ are monic, they must be related by $p_{n+1}(x) = (x-a_n)p_n(x) + q(x)$ for some number a_n and some polynomial $q(x)$ of degree $\le n-1$. In fact $q(x) = -b_n p_{n-1}(x)$ for some number b_n since $p_{n+1}(x)$ and $(x-a_n)p_n(x)$ are orthogonal to all polynomials of degree $\le n-2$, so q has to be as well. The only polynomials of degree $n-1$ with this property are multiples of $p_{n-1}(x)$, and we denote this multiple by $-b_n$. So we have the recurrence

$$\begin{aligned} p_1(x) &= (x-a_0)p_0(x), \\ p_{n+1}(x) &= (x-a_n)p_n(x) - b_n p_{n-1}(x), \qquad (n \ge 1). \end{aligned} \tag{2}$$

The coefficients a_n and b_n are found by requiring $\langle p_{n+1}, p_n\rangle = 0$ and $\langle p_{n+1}, p_{n-1}\rangle = 0$. This gives

$$a_n = \frac{\langle xp_n, p_n\rangle}{c_n}, \qquad b_n = \frac{\langle xp_n, p_{n-1}\rangle}{c_{n-1}} = \frac{c_n}{c_{n-1}} = \frac{n!^4}{(n-1)!^4}\frac{(2n-2)!^2}{(2n)!^2}\frac{2^{2n+1}}{2^{2n-1}}\frac{2n-1}{2n+1} = \frac{n^2}{4n^2-1},$$

where $c_n = \langle p_n, p_n\rangle = \langle p_n, xp_{n-1}\rangle - a_{n-1}\underbrace{\langle p_n, p_{n-1}\rangle}_{0} - b_{n-1}\underbrace{\langle p_n, p_{n-2}\rangle}_{0}$. Since $(x^2-1)^n$ is an even function, n derivatives of it will be even or odd if n is even or odd, respectively. Thus, $a_n = c_n^{-1}\int_{-1}^{1} xp_n(x)^2\,dx = 0$ since the integrand is odd. The normalized Legendre polynomials are defined by $\varphi_n(x) = p_n(x)/\sqrt{c_n}$ so that $\langle \varphi_n, \varphi_m\rangle = \delta_{mn}$. From (2), we find that $\sqrt{c_{n+1}}\varphi_{n+1}(x) = (x-a_n)\sqrt{c_n}\varphi_n(x) - b_n\sqrt{c_{n-1}}\varphi_{n-1}(x)$. Dividing through by $\sqrt{c_n}$ and using $b_n = c_n/c_{n-1}$, we obtain

$$\begin{aligned} \sqrt{b_1}\varphi_1(x) &= (x-a_0)\varphi_0(x), \\ \sqrt{b_{n+1}}\varphi_{n+1}(x) &= (x-a_n)\varphi_n(x) - \sqrt{b_n}\varphi_{n-1}(x), \qquad (n \ge 1), \end{aligned} \tag{3}$$

where $\varphi_0(x) = c_0^{-1/2} = \sqrt{1/2}$ is needed to get started. It is conventional to define $b_0 = c_0$ so that $c_n = \prod_{j=0}^{n} b_j$. The recurrence relation (3) can be organized as matrix-vector multiplication,

$$\begin{pmatrix} a_0 & \sqrt{b_1} & 0 & & & \\ \sqrt{b_1} & a_1 & \sqrt{b_2} & 0 & & \\ & \ddots & \ddots & \ddots & \ddots & \\ & & \sqrt{b_{n-2}} & a_{n-2} & \sqrt{b_{n-1}} & 0 \\ & & & \sqrt{b_{n-1}} & a_{n-1} & \sqrt{b_n} \end{pmatrix}\begin{pmatrix} \varphi_0(x) \\ \varphi_1(x) \\ \vdots \\ \varphi_n(x) \end{pmatrix} = x\begin{pmatrix} \varphi_0(x) \\ \varphi_1(x) \\ \vdots \\ \varphi_{n-1}(x) \end{pmatrix}.$$

We saw in class that the zeros of $p_n(x)$ are real, distinct, and lie in the interval $(-1,1)$. We will denote them by $-1 < x_1 < \cdots < x_n \le 1$. At each x_j, the last entry of the vector on the left becomes zero, so we can drop it, as well as the last column of the matrix. That means x_j is an eigenvalue and $[\varphi_0(x_j);\cdots;\varphi_{n-1}(x_j)]$ is an eigenvector of the matrix that remains when you drop the last column. From these eigenvectors, we can form a matrix $\Phi_{jk} = \varphi_j(x_k)$, $0 \le j < n$, $1 \le k \le n$, which satisfies

$$A\Phi = \Phi X, \qquad A = \begin{pmatrix} a_0 & \sqrt{b_1} & & \\ \sqrt{b_1} & a_1 & \ddots & \\ & \ddots & \ddots & \sqrt{b_{n-1}} \\ & & \sqrt{b_{n-1}} & a_{n-1} \end{pmatrix}, \qquad X = \begin{pmatrix} x_1 & & \\ & \ddots & \\ & & x_n \end{pmatrix}. \tag{4}$$

The symmetric, tridiagonal matrix A is known as the Jacobi matrix of this orthogonal polynomial system. In matlab, we can compute the eigenvalues and eigenvectors of a matrix with the `eig` command. `[Q,X]=eig(A)` will give a diagonal matrix X and orthogonal matrix Q such that $AQ = QX$. Since the eigenvalues are distinct, the eigenvectors are uniquely determined up to constant multiples. That means the columns of Φ are multiples of the columns of Q. The square of these multiples give the quadrature weights! To see this, note that

$$\delta_{ij} = \int_{-1}^{1} \varphi_i(x)\varphi_j(x)\,dx = \sum_{k=1}^{n} \varphi_i(x_k)\varphi_j(x_k)\,w_k, \qquad (0 \le i,j < n), \tag{5}$$

where $w_k = \int_{-1}^{1} L_k(x)\,dx$ are the quadrature weights. Indeed, we showed in class that choosing the nodes to be the zeros of $p_n(x)$ (or equivalently of $\varphi_n(x)$) and choosing the weights this way will integrate polynomials of degree $2n-1$ exactly, and these products of $\varphi_i(x)\varphi_j(x)$ have degree at most $2n-2$. Equation (5) can also be written in matrix form as follows:

$$I_{n\times n} = \Phi\,\mathrm{diag}(w)\Phi^T.$$

As in matlab, $\mathrm{diag}(w)$ is the diagonal matrix $W_{ij} = w_i\delta_{ij}$ while $\mathrm{diag}(X) = [x_{11}; x_{22}; \ldots; x_{nn}]$ extracts the diagonal of X into a vector. Since Q is orthogonal, it satisfies $QQ^T = I$. We also know $Q = \Phi\,\mathrm{diag}(\alpha)$ for some vector α (since the columns of Q and Φ are constant multiples of each other). Thus, it must be that $w_k = \alpha_k^2$. We know $\varphi_0(x) = \sqrt{1/b_0}$, so all the entries in the first row of Φ have this value. Thus, $\alpha_k = Q_{0k}/\Phi_{0k}$ and $w_k = b_0 Q_{0k}^2$. This is know as the Golub-Welsch algorithm for computing quadrature weights and abscissas.

In summary, what you do is form the tridiagonal matrix A in (4), compute `[Q,X]=eig(A)`, set `x=diag(X)`, and `w=b0*Q(1,:)'.^2`. A variant that works better on infinite domains (e.g. for the Laguerre polynomials in problem 7 below) is to compute `x=eig(A)` (without computing eigenvectors), compute Φ at these gridpoints using the recurrence (3), and then define `w=1.0./diag(Phi'*Phi)`. This formula follows from $\Phi = Q\,\mathrm{diag}(\alpha)^{-1}$ since $\Phi^T\Phi = \mathrm{diag}(\alpha)^{-1}\underbrace{Q^TQ}_{I}\mathrm{diag}(\alpha)^{-1} = \mathrm{diag}(w)^{-1}$.

---

1. (good method) Write a program to take as input vectors $a = [a_0,\ldots,a_{n-1}]$, $b = [b_0,\ldots,b_n]$, and $x = [x_1,\ldots,x_m]$, and returns as output the matrix $\Phi_{jk} = \varphi_j(x_k)$, where the recurrence relation (3) is used to compute $\varphi_j(x_k)$. Make a plot of $\varphi_n(x)$ with $n = 10, 50$ by calling your program with `a=zeros(1,n)`, `b=[2,[1:n].^2./(4*[1:n].^2-1)]`, `x=cos(linspace(-pi,0,10*n))`. The last row of Φ will be $\varphi_n(x)$. At this point the x's are not quadrature abscissas, just points where you want to evaluate $\varphi_n(x)$.

2. (bad method) Write a program to take vectors $a = [a_0,\ldots,a_{n-1}]$, $b = [b_0,\ldots,b_n]$ and return the coefficients of the polynomials $\varphi_n(x) = s_{n0}x^n + s_{n1}x^{n-1} + \cdots + s_{n,n-1}x + s_{nn}$, constructed via the recurrence (3). This can be done by building a matrix S with the first row storing the coefficients of $\varphi_0(x)$, the second row storing those of $\varphi_1(x)$, etc. Without `for` loops, the code would look something like this:

```
n = length(a);
S = zeros(n+1,n+1);
S(1,1) = sqrt(1/b(1));
S(2,:) = (S(1,:)-[0,a(1)*S(1,1:end-1)])/sqrt(b(2));
S(3,:) = (S(2,:)-[0,a(2)*S(2,1:end-1)]-[0,0,sqrt(b(2))*S(1,1:end-2)])/sqrt(b(3));
S(4,:) = (S(3,:)-[0,a(3)*S(3,1:end-1)]-[0,0,sqrt(b(3))*S(2,1:end-2)])/sqrt(b(4));
etc.
```

The formula for $S(j+1,:)$ copies $S(j,:)$, which corresponds to multiplying by x, then adds to this $-a(j)S(j,:)$ and $-\sqrt{b(j)}S(j-1,:)$, each shifted by one or two slots to account for the fact that the first column of the jth row corresponds to x^{j-1}. Your job is to turn the last two lines into a loop that computes `S(3,:)` through `S(n+1,:)`. As a check, the 5th row of S will contain

```
[9.280776503073438, 0, -7.954951288348661, 0,  0.795495128834866]
```

which represents $\varphi_4(x) = 9.280776503073438x^4 - 7.954951288348661x^2 + 0.795495128834866$. The code should return the matrix S. As a specific task, set $n = 50$ and compute the 51×51 matrix S. Then make a plot of $\varphi_{10}(x)$ (which will look fine) and $\varphi_{50}(x)$, which will be completely wrong due to roundoff errors. Note that the coefficients of $\varphi_k(x)$ are in row $k+1$ of S, and you can use matlab's builtin function polyval to evaluate $\varphi_k(x)$ at the grid points `x=cos(linspace(-pi,0,500))`. The command `y=polyval(S(11,[1:11]),x)` will evaluate $\varphi_{10}(x)$ at the points in the vector x.

3. (good method) Write a program to compute an n-point quadrature rule by the Golub-Welsch algorithm described above. The input should be the recursion coefficient vectors $a = [a_0, \dots, a_{n-1}]$, $b = [b_0, \dots, b_n]$. (The last entry of b can be ignored). The code should return two vectors, x and w, each of length n, containing the quadrature weights and abscissas. Apply your scheme to the functions $f(x) = \cos(k \operatorname{acos}(x))$ for $k = 0, \dots, 2n$ and compute the error. (These are actually polynomials in x.) Assuming x and w are column vectors, this can be done via:

```
for k=0:2*n, E(k+1)=abs(w'*cos(k*acos(x)) - (1+(-1)^k)/(1-k^2 + 1.0e-18)); end.
```

Try this for $n = 10$ and report the abscissas x_k, weights w_k, and errors E_k. Then repeat with n=40 and report `norm(E(1:80))`, i.e. omit the last entry when computing the norm, which is the only one the scheme doesn't integrate exactly (aside from roundoff error).

4. (bad method) Use your code from part 2 to generate the coefficients of $\varphi_n(x)$ and use matlab's builtin function 'roots' to find the abscissas: `x=sort(roots(S(n+1,1:n+1)))`. Evaluate the matrix Φ from the theory above: `for j=1:n, Phi(j,:)=polyval(S(j,1:j),x); end`. Then compute the weights as `w=1.0./diag(Phi'*Phi)`. Check the quality of the quadrature weights by computing x_k, w_k and E_k for $n = 10$ and `norm(E(1:80))` for $n = 40$, just as in part 3. For still larger values of n, this method can fail completely, returning complex zeros.

Moral of the story: try to avoid representing polynomials by expanding them out. Recurrence relations like (2) or (3) are much more stable.

The remaining problems are "extra credit" for students who lost points on (or didn't submit one of) the first two programming assignments. If you have perfect scores already, they are for "glory", not points. You can't bank points for the last programming assignment. You don't have to do all of them to get some extra credit.

5. Make a "Riemann sum" picture of a 20 point Gaussian quadrature scheme integrating the function $f(x) = e^{x-x^2}$ from -1 to 1. In more detail, if $x_1, \dots, x_n$ are the abscissas and $w_1, \dots, w_n$ are the weights, define $\xi_0 = -1$ and $\xi_k = \xi_{k-1} + w_k$. The quadrature rule then looks like $\int_{-1}^{1} f(x)\,dx = \sum_{k=1}^{n} f(x_i)[\xi_k - \xi_{k-1}]$, so the kth rectangle extends from ξ_{k-1} to ξ_k and has height $f(x_k)$. Plot the function $f(x)$ on top of these rectangles. For example, if $n = 3$, after computing the x_k and ξ_k, which are both indexed from $k = 1$ in matlab, we could do this:

```
xx = [ xi(1)  xi(1)   xi(2)  xi(2)  xi(2)   xi(3)  xi(3)  xi(3)   xi(4)   xi(4) ];
zz = [   0   f(x(1)) f(x(1))  0    f(x(2)) f(x(2))  0    f(x(3)) f(x(3))    0   ];
plot(xx,zz);  hold on;
plot(x,f(x),'.');  x2=linspace(-1,1,200);
plot(x2,f(x2),'-'); hold off; set(gca,'xlim',[-1,1]);
```

Note the way we trace out the Riemann rectangles by dropping down to the x-axis and up to the next rectangle height. An example with $n = 12$ is given below. (Generating `xx` and `zz` should be done with a loop rather than by hand like this.)

6. The Laguerre polynomials are orthogonal with respect to $\langle f, g\rangle = \int_0^\infty f(x)g(x)e^{-x}\,dx$. They also satisfy a recurrence relation of the form (2) for the monic polynomials and (3) for the normalized polynomials. In this case, $a_n = 2n+1$, $b_0 = 1$, $b_n = n^2$ for $n \geq 1$. After computing $\varphi_n(x)$ using the recurrence (3), we can re-scale them using the square root of the weight function: $\tilde{\varphi}_n(x) = \varphi_n(x)e^{-x/2}$. These functions are orthogonal with respect to the unweighted inner product $\langle f, g\rangle = \int_0^\infty f(x)g(x)\,dx$, and therefore do not grow rapidly like the polynomials $\varphi_n(x)$. Make a plot of $\tilde{\varphi}_{80}(x)$ on the interval $[0, 400]$ using the grid `x=800*(sin(linspace(0,pi/4,1000)).^2)`. A plot of $\tilde{\varphi}_{50}(x)$ is given below.

7. Compute the eigenvalues x_k of the Jacobi matrix associated with a_k, b_k from problem 6 and compute the matrix $\tilde{\Phi}_{jk} = \tilde{\varphi}_j(x_k)$. Compute the weights $\tilde{w}_k$ from `wtil=1.0./diag(Phi1'*Phi1)`, where `Phi1` stands for $\tilde{\Phi}$. This gives the Laguerre quadrature scheme

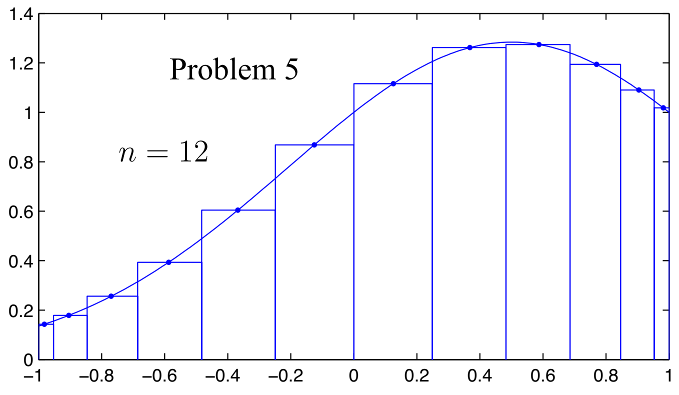
$$\int_0^\infty f(x)e^{-x}\,dx \approx \sum_{k=1}^{n} f(x_k)\,w_k, \qquad w_k = e^{-x_k}\tilde{w}_k.$$

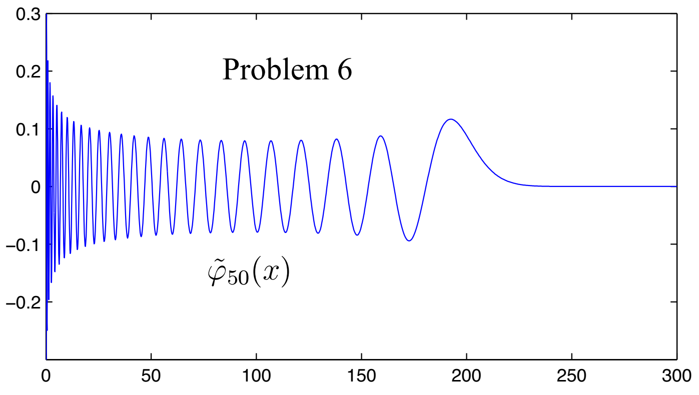
Check your results x_k, w_k for $n = 2$ and $n = 3$ against the answer in the back of the book for problem 4.9.7. Use your new scheme to approximate the integral $\int_0^\infty x^{10}\cos(x)e^{-x}\,dx$ using $n = 20, 30, 40$.

8. Use 10-point Gaussian quadrature on $[-1, 1]$ to compute the 20-point open Newton-Cotes formula on $[-1, 1]$. In other words, let x_j, w_j be the Gaussian quadrature abscissas and weights and $\tilde{x}_j$, $\tilde{w}_j$ be the Newton-Cotes abscissas and weights. Then $\tilde{x}_j = -1 + 2j/21$, $j = 1, \dots, 20$ and

$$\tilde{w}_j = \int_{-1}^{1} \tilde{L}_j(x)\,dx, \qquad \tilde{L}_j(x) = \prod_{k\neq j} \frac{x - \tilde{x}_k}{\tilde{x}_j - \tilde{x}_k}.$$

Compute these integrals for the $\tilde{w}_j$ using 10 point Gaussian quadrature. Evaluate the Lagrange polynomials $\tilde{L}_j$ as you did in the second programming assignment. Plot $\tilde{w}_j$ versus $\tilde{x}_j$. Clearly, a "Riemann sum" interpretation is not possible when some of the weights are negative and many of the weights are huge — much larger than the width of the domain.


Problem 5
$n = 12$


Problem 6
$\tilde{\varphi}_{50}(x)$