Scientific Computing for DPhil Students I
Assignment 3

Due at lecture at 10:00 on Tues. 19 Nov. 2019. This is the third of four assignments this term.

1 A well-conditioned problem: interpolation in Chebyshev points

Suppose f is a continuous function on $[-1,1]$, and given $n \geq 1$, let $x_j = \cos(j\pi/n)$ for $0 \leq j \leq n$ be the set of $n+1$ *Chebyshev points* in $[-1,1]$. There is a unique degree n polynomial interpolant $p(x) = \sum_{k=0}^{n} c_k x^k$ to f in these points, which we call the *Chebyshev interpolant.* Moreover, p is a well-conditioned function of f: if f is perturbed by a function Δf and Δp is the resulting perturbation in p, then it can be shown that $\|\Delta p\|/\|\Delta f\| < 1 + (2/\pi)\log(n+1)$, where $\|\cdot\|$ is the ∞-norm on $[-1,1]$. See Trefethen, *Approximation Theory and Approximation Practice,* Theorem 15.2.

What is the smallest integer n for which this bound does not ensure $\|\Delta p\|/\|\Delta f\| < 10$?

2 An unstable algorithm: polyval(polyfit)

There is an obvious method for computing $p(x)$: set up a system of linear equations $Ac = f$ for the coefficient vector $c = (c_0, \ldots, c_n)^T$, solve it, and use these coefficients to evaluate $p(x)$. This method is awful, a perfect example of an unstable algorithm for a well-conditioned problem. Unfortunately, the method is used all too often, not least because it is offered in Matlab with the commands `polyfit` and `polyval`.

(a) Look up `polyfit` *and* `polyval` *and use them to compute Chebyshev interpolants of $f(x) = |x|$ on $[-1,1]$ for $n = 40$ and 80. What is the ∞-norm of the vector c? Use* `polyval` *to compute p(*`xx`*) for* `xx = linspace(-1,1,500)'` *and plot* `xx` *against p(*`xx`*).*

(b) Comment on how the norms you just calculated relate to the plots you just plotted in light of the value of machine epsilon of around 10^{-16}.

A is called a *Vandermonde matrix:* it has dimensions $(n+1) \times (n+1)$, and its columns are the $n+1$ functions $1, x, x^2, \ldots, x^n$ sampled at the $n+1$ Chebyshev points. In Matlab you can construct A with the command `vander(x)`, where `x` is a vector of Chebyshev points. (This will put the columns in reverse order, but you can rearrange them that if you wish with `fliplr`.)

(c) Calculate the condition numbers of A for $n = 40$ and 80. Comment on their significance in the light of machine epsilon.

3 A stable algorithm: polyval(polyfit) + Arnoldi

Although this is not very well known, there is a simple modification to the algorithm above that makes it stable, going by the name of the *Arnoldi process.* The idea here is to find an $(n+1) \times (n+1)$ matrix V with orthogonal columns that span the same spaces as the columns of A. If we do this simply by constructing a QR factorization of A, that's not good enough; the instability is baked in when those high powers x^k are computed. Instead, Arnoldi computes V by orthogonalising column by column. Column v_1 is the vector of all 1's. (The columns will be orthogonal, not orthonormal.) Column v_2 is obtained by multiplying v_1 by x, then subtracting off the component in the direction of v_1. Column v_3 is obtained by multiplying v_2 by x and subtracting off the components in the directions of v_1 and v_2. And so on. Here is an Arnoldi version of `polyfit`. As output, it produces not just a vector of coefficients c, but also a Hessenberg matrix H that stores the coefficients of the orthogonalization process.

```
function [c,H] = polyfitA(x,y,n)
M = length(x); V = ones(M,1); H = zeros(n+1,n);
for k = 1:n
    v = x.*V(:,k);
    for j = 1:k
        H(j,k) = V(:,j)'*v/M;
        v = v - H(j,k)*V(:,j);
    end
    H(k+1,k) = norm(v)/sqrt(M);
    V = [V v/H(k+1,k)];
end
c = V'*y/M;
```

Once you've got c and H, you can call this Arnoldi version of `polyval` to evaluate the polynomial at a vector of points `xx`:

```
function yy = polyvalA(c,H,xx)
M = length(xx); n = size(H,2); V = ones(M,1);
for k = 1:n
    v = xx.*V(:,k);
    for j = 1:k
        v = v - H(j,k)*V(:,j);
    end
    V = [V v/H(k+1,k)];
end
yy = V*c;
```

Use these codes to compute the same interpolants for $n = 40$ and $n = 80$ as in problem 2. Plot the results as before and give the ∞-norm of c. Comment on what you see.

4 Least-squares

The method we have just explored is extremely general: it works on arbitrary real or complex domains, not just $[-1, 1]$, and it stabilizes least-squares fits as well as interpolants. To illustrate, we will find a polynomial of degree n that is a good approximation to $\text{sign}(x)$ on the domain $X = [-4, -1] \cup [1, 8]$.

(a) Discretize X by a vector `x` *with* 300 *equispaced points on the left and* 700 *on the right. Use* `polyfit` *and* `polyval` *to compute least-squares fits to sign*(`x`) *on* `x` *of degrees $n = 40$ and* 80*, and plot the error function in each case.*

(b) Likewise with `polyfitA` *and* `polyvalA`*. Comment on the difference.*