CS4220 Assignment 2  Due: 2/13/14 (Thur) at 11pm

You must work either on your own or with one partner. You may discuss background issues and general solution strategies with others, but the solutions you submit must be the work of just you (and your partner). If you work with a partner, you and your partner must first register as a group in CMS and then submit your work as a group. Each submitted function is worth 5 points. Points may be deducted for poor style.

Topics: SVD, LU, Fast Transforms, connections between geometry and linear algebra.

1  Matrix-Times-Vector and the 1-Norm

Download and run the script ShowRange2. Observe that it displays the set

\[ S_2 = \{ x \in \mathbb{R}^2 \mid \| x \|_2 \leq 1 \} \]

and the set

\[ \text{Im}(S_2) = \{ y \in \mathbb{R}^2 \mid y = Ax \text{ and } x \in S_2 \} \]

for random 2-by-2 matrices \( A \). Develop a script ShowRange1 that does the same thing only where the 1-norm is used instead of the 2-norm. In particular, it should display

\[ S_1 = \{ x \in \mathbb{R}^2 \mid \| x \|_1 \leq 1 \} \]

and the set

\[ \text{Im}(S_1) = \{ y \in \mathbb{R}^2 \mid y = Ax \text{ and } x \in S_1 \} \]

for random 2-by-2 matrices \( A \). Submit ShowRange1 to CMS.

2  Hadamard Systems

The Hadamard matrices are defined as follows:

\[
H_1 = \begin{bmatrix} 1 \end{bmatrix}
\]

\[
H_2 = \begin{bmatrix} H_1 & H_1 \\ H_1 & -H_1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}
\]

\[
H_4 = \begin{bmatrix} H_2 & H_2 \\ H_2 & -H_2 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}
\]

\[
H_{n} = \begin{bmatrix} H_m & H_m \\ H_m & -H_m \end{bmatrix} \quad (n = 2m)
\]

Suppose \( b \in \mathbb{R}^n \) and that \( n \) is a power of two and that we want solve \( H_n x = b \), i.e.,

\[
\begin{bmatrix} H_m & H_m \\ H_m & -H_m \end{bmatrix} \begin{bmatrix} x(1:m) \\ x(m+1:n) \end{bmatrix} = \begin{bmatrix} b(1:m) \\ b(m+1:n) \end{bmatrix}
\]

By manipulating this equation, show that we can solve the order-\( n \) Hadamard system \( H_n x = b \) by solving a pair of half-sized Hadamard systems. Use this idea to develop a recursive implementation for
function x = HadSolve(b)
    \% b is a column-vector and n is a power of 2.
    \% Solves the nxn Hadamard system Hn*x = b

In addition, write a recursive function HadSolveFlops that computes the number of flops required to solve an order-n Hadamard system, i.e.,

function N = HadSolveFlops(n)
    \% n is a power of 2.
    \% N is the number of flops required to solve an order-n Hadamard system.

Submit HadSolve and HadSolveFlops to CMS. A test script HadCheck can be downloaded from the course website.

3 Chebychev Interpolation

Suppose we are given the points \((x_1, y_1), \ldots, (x_n, y_n)\) with \(x_1 < x_2 < \cdots < x_n\) and “basis functions” \(B_1(x), \ldots, B_n(x)\). Our goal is to determine scalars \(\alpha_1, \ldots, \alpha_n\) so that if

\[
    f(x) = \alpha_1 B_1(x) + \alpha_2 B_2(x) + \cdots + \alpha_n B_n(x)
\]

then

\[
    f(x_i) = y_i, \quad i = 1:n.
\]

This interpolation problem is a linear equation problem. In particular, the vector of unknown coefficients \(\alpha\) satisfies \(M\alpha = y\) where \(B_i(x_j)\) is the \(ij\) entry in the matrix \(M\). In this problem you write a function that sets up and solves such a system when the basis functions \(B_i(x)\) are based on Chebyshev polynomials.

The Chebyshev polynomials \(T_0(x), T_1(x), \ldots\) are defined as follows:

\[
    T_k(x) = \begin{cases} 
        1 & \text{if } k = 0 \\
        x & \text{if } k = 1 \\
        2xT_{k-1}(x) - T_{k-2}(x) & \text{if } k \geq 2. 
    \end{cases}
\]

Thus, \(T_k(x)\) has degree \(k\). These polynomials are very nicely behaved on the interval \([-1, +1]\) in that they are bounded by 1 in absolute value. However, outside of \([-1, +1]\] the Chebyshev polynomials take on huge values. This poses a problem in the interpolation setting should any of the \(x_i\) fall outside \([-1, +1]\). A way around this is to set

\[
    B_i(x) = T_{i-1} \left( -1 + 2 \frac{x-a}{b-a} \right)
\]

where \(a = x_1\) and \(b = x_n\). We’ll call this the “Chebyshev polynomial basis with respect to the interval \([a, b]\)”.

Notice that

\[
    a \leq x \leq b \quad \Rightarrow \quad -1 \leq -1 + 2 \frac{x-a}{b-a} \leq b
\]

so when we set up the \(M\) matrix, the underlying Chebyshev evaluations are “nice”. Complete the following function so that it performs as specified:

function alfa = ChebyInterp(x,y)
    \% x and y are column n-vectors and x(1)<x(2)<\ldots<x(n).
    \% alfa is a column n-vector with the property that if
    \% \% f(x) = alfa(1)B_{1}(x) + \cdots + alfa(n)B_{n}(x)
    \% where the B_{1}(x), \ldots, B_{n}(x) is the Chebyshev polynomial
    \% basis with respect to \([a,b]\), then f(x(i)) = y(i), i=1:n.

Your implementation must make effective use of the Matlab \texttt{lu} function. The process by which you set up the matrix \(M\) should be vectorized. Submit \texttt{ChebyInterp} to CMS. (Note: You will have to write your own test script for this problem.)

2
4 Using LU to Estimate \( \{\sigma_n, u_n, v_n\} \)

Suppose \( U^T AV = \Sigma \) is the SVD of \( A \in \mathbb{R}^{n \times n} \) and that \( u_k = U(:,k) \), \( v_k = V(:,k) \), and \( \sigma_k = \Sigma_{kk} \). Note that

\[
A = U\Sigma V^T = \sum_{k=1}^{n} \sigma_k u_k v_k^T.
\]

In many applications the smallest “singular triple” \( \{\sigma_n, u_n, v_n\} \) is required. Typically this is because the matrix

\[
B = A - \sigma_n u_n v_n^T = \sum_{k=1}^{n-1} \sigma_k u_k v_k^T
\]

is the closest singular matrix to \( A \) as measured by the Frobenius norm. (Nice exercise: show that \( \| A - B \|_F = \sigma_n \).) In this problem we show how to estimate \( \{\sigma_n, u_n, v_n\} \) using the LU factorization, which is much cheaper than computing the SVD.

To motivate the method we assume that \( \sigma_{n-1} \gg \sigma_n \). Suppose \( z \in \mathbb{R}^n \) has unit 2-norm and that we solve \( A w = z \). Thus,

\[
w = A^{-1} z = (U\Sigma V^T)^{-1} z = (V\Sigma^{-1} U^T) z = \left(\sum_{k=1}^{n} \frac{1}{\sigma_k} u_k v_k^T\right) z = \sum_{k=1}^{n} \left(\frac{u_k^T z}{\sigma_k}\right) v_k
\]

Unless \( z \) is (nearly) orthogonal to \( u_n \), we see that \( w \) is very “rich” in the direction of \( v_n \) because of the assumption that \( \sigma_n \ll \sigma_{n-1} \). (Note. When we say that a vector \( f \) is rich in the direction of another vector \( g \), we mean that \( f \) (almost) points in the same direction as \( g \).)

Likewise, suppose \( z \in \mathbb{R}^n \) has unit 2-norm and that we solve \( A^T w = z \). Thus,

\[
w = (A^T)^{-1} z = (V\Sigma U^T)^{-1} z = (U\Sigma^{-1} V^T) z = \left(\sum_{k=1}^{n} \frac{1}{\sigma_k} u_k v_k^T\right) z = \sum_{k=1}^{n} \left(\frac{v_k^T z}{\sigma_k}\right) u_k
\]

Unless \( z \) is (nearly) orthogonal to \( v_n \), we see that \( w \) is very rich in the direction of \( u_n \) because of the assumption that \( \sigma_n \ll \sigma_{n-1} \).

This suggests a method for “bootstrapping” our way to good estimates for \( u_n \) and \( v_n \):

Let \( z \in \mathbb{R}^n \) be a random vector with unit 2-norm.

for \( k = 1 : n \) Repeat

\[
\text{Solve } A w = z \text{ and set } v = w/\|w\|_2.
\]

\[
\text{Set } z = v.
\]

\[
\text{Solve } A^T w = z \text{ and set } \sigma = 1/\|w\|_2 \text{ and } u = w/\|w\|_2.
\]

end

\{ \sigma, u, \text{ and } v \text{ are the approximations to } \sigma_n, u_n, \text{ and } v_n \} \]

Here is a brief heuristic explanation that explains why \( \sigma \) can be regarded as an approximation to \( \sigma_n \). If \( z = v_n \), then \( \sigma = \sigma_n \). If \( z \) is almost in the direction of \( v_n \), then \( \sigma \) will almost equal \( \sigma_n \).

Write a function \([\text{sigma, u, v}] = \text{SigUVn}(A, \text{nRepeat})\) that implements this idea. It should compute the factorization \( PA = LU \) and then use it repeatedly to solve the systems \( A w = z \) and \( A^T w = z \) as the iteration progresses. Submit \text{SigUVn} to CMS. A test function \text{ShowSigUVn} is available on the course website.