Jeffery Hein
MATH 126 - Dr. Barnett
Homework 1
due January 18, 2012

Problem 1. If u and v are m-vectors, the matrix $A = I + uv^*$ is known as the *rank-one perturbation of the identity.* Show that if A is non-singular, then its inverse has the form $A^{-1} = I + \alpha uv^*$ for some scalar α, and give an expression for α. For what u and v is A singular? If it is singular, what is Null(A)?

Proof. Suppose $A = I + uv^*$ is nonsingular, we will show that $A^{-1} = I + \alpha uv^*$ where

$$\alpha = \frac{-1}{1 + \langle u, v\rangle}.$$

Since the inverse of a matrix is unique, we need only verify that $(I + uv^*)(I + \alpha uv^*) = I$. To this end, we observe that

$$AA^{-1} = (I + uv^*)\left(I - \frac{uv^*}{1 + \langle u, v\rangle}\right) = I - \frac{uv^*}{1 + \langle u, v\rangle} + uv^* - \frac{uv^*uv^*}{1 + \langle u, v\rangle}.$$

Furthermore, note that $uv^*uv^* = u(v^*u)v^* = u\langle v, u\rangle v^* = \langle u, v\rangle uv^*$. This means that we have

$$AA^{-1} = I + uv^* - (1 + \langle u, v\rangle)\left(\frac{uv^*}{1 + \langle u, v\rangle}\right) = I + uv^* - uv^* = I,$$

as desired. Of course, we implicitly assumed that $\langle u, v\rangle \neq 1$, and so we will now show that assuming $\langle u, v\rangle = -1$ leads to a contradiction. Consider the following:

$$A^2 = (I + uv^*)(I + uv^*) = I + 2uv^* + \langle u, v\rangle uv^* = I + uv^* = A.$$

Since A is nonsingular, then it follows that $I = A = I + uv^*$, hence $uv^* = 0$, which is only possible if $u = 0$ or $v = 0$. In either case, we observe that $\langle u, v\rangle = 0$, which is a contradiction. This means that if A is nonsingular, then $\langle u, v\rangle \neq -1$, and so by the contraposition of this statement, we have that if $\langle u, v\rangle = -1$, then A is singular.

Lastly, we claim that if A is singular, then Null(A) = span(u). Let $x \in$ span(u), and so there is $\alpha \in \mathbb{R}$ such that $x = \alpha u$. It then follows that

$$Ax = (I + uv^*)(\alpha u) = \alpha u + uv^*(\alpha u) = \alpha u + \alpha u\langle v, u\rangle = 0,$$

since $\langle u, v\rangle = -1$, hence span$(u) \subset$ Null(A). With $x \in$ Null(A), we have $0 = Ax = (I+uv^*)x = x+uv^*x$, which implies that $x = -u\langle v, x\rangle \in$ span(u), so our claim holds. $\square$

Problem 2. The spectral radius $\rho(A)$ of a square matrix A is the magnitude of its largest eigenvalue. Prove that $\rho(A) \leq ||A||_2$.

Proof. Suppose, for a contradiction, that $\rho(A) > ||A||_2$, and so there is an eigenvalue λ such that $|\lambda| > ||A||_2$. Additionally, there is an associated λ-eigenvector v such that $Av = \lambda v$. Let $e = v/||v||$, and so $||e|| = 1$. It then follows that

$$||Ae|| = \left|\left|\frac{Av}{||v||}\right|\right| = \left|\left|\frac{\lambda v}{||v||}\right|\right| = |\lambda| \cdot ||e|| = |\lambda|.$$

This means that

$$||A||_2 = \sup_{||u||=1} ||Au|| \geq |\lambda| > ||A||_2,$$

which is a contradiction. It therefore follows that $\rho(A) \leq ||A||_2$. $\square$

Problem 3. Use the in-class worksheet on the following $m \times m$ bidiagonal matrix to answer the below.

$$A = \begin{bmatrix} 1 & 2 & & \\ & 1 & 2 & \\ & & 1 & \ddots \\ & & & \ddots \end{bmatrix}$$

(a) find a nontrivial lower bound on the condition number $\kappa(A)$;

(b) predict the smallest m such that roughly all significant digits will be lost in the solution x to a linear system $Ax = b$ in double-precision;

(c) demonstrate your last claim in a couple of lines of code, by starting with a known x, computing b then solving via `mldivide`. [Hint: look up the `toeplitz` command to construct the matrix rather than use a loop. You need to choose a b that causes floating-point rather than exact integer arithmetic to be used!]

Proof of (a). To determine a lower bound to $\kappa(A)$, we need only compute lower bounds for $||A||$ and $||A^{-1}||$. We showed in class that

$$A^{-1} = \begin{bmatrix} 1 & -2 & 4 & -8 & \cdots \\ & 1 & -2 & 4 & \ddots \\ & & 1 & -2 & \ddots \\ & & & 1 & \ddots \\ & & & & \ddots \end{bmatrix},$$

and so we can determine a lower bound by considering A^{-1} acting on $x = (0, \ldots, 0, 1)$. This yields a vector of the form

$$A^{-1}x = \begin{bmatrix} (-2)^{m-1} & (-2)^{m-2} & \cdots & -2 & 1 \end{bmatrix}^*,$$

and it is easily seen that $||A^{-1}x|| = \sqrt{\frac{4^m-1}{3}}$, and so this is a lower bound for $||A^{-1}||$.

Now, consider A acting on the unit vector

$$v = \begin{bmatrix} \frac{1}{\sqrt{m}} & \cdots & \frac{1}{\sqrt{m}} \end{bmatrix}^*,$$

which yields

$$Av = \begin{bmatrix} \frac{3}{\sqrt{m}} & \cdots & \frac{3}{\sqrt{m}} & \frac{1}{\sqrt{m}} \end{bmatrix}^*.$$

It then follows that

$$||Av|| = \sqrt{\frac{9}{m} + \cdots + \frac{9}{m} + \frac{1}{m}} = \sqrt{\frac{9(m-1)+1}{m}} = \sqrt{9 - \frac{8}{m}},$$

and so we conclude that

$$\sqrt{\frac{4^m - 1}{3}} \cdot \sqrt{9 - \frac{8}{m}} \leq \kappa(A)$$

is our nontrivial lower bound. □

Solution to (b). Now that we've provided a nontrivial lower bound to $\kappa(A)$, we want to determine the smallest m such that we lose most of our significant digits when solving $Ax = b$ for x. We've showed in class that we expect to recover x in such a system to no better than $16 - \log_{10}(\kappa(A))$ digits. This means that we need only find the smallest m such that $\log_{10}(\kappa(A)) \geq 16$. We determine this value of m by determining the smallest m such that $\log_{10}$ of the lower bound found in part (a) is greater than 16, thereby forcing $\log_{10}(\kappa(A)) \geq 16$. This occurs when $m = 53$. □

Solution to (c). To prove this claim, we consider the following MATLAB script:

```
format long g;
m = 53;
R = zeros(m,1); C = zeros(m,1);
R(1) = 1; R(2) = 2; C(1) = 1;
A = toeplitz(C,R);
x = 0.1 + (1:m);
b = A*x';
(A\b)'
```

In executing this script, we would normally expect to recover the original x; however, what we find is that the first 8-10 entries of the solution are now significantly different than the values present in x. In fact, as we increase the value of m, the first few entries become increasingly larger. One key observation to note is that for $m = 52$, this effect is likewise present. This is to be expected though, since we only found a lower bound to $\kappa(A)$, not an exact value. Notwithstanding, this exercise demonstrates our ability to predict such an m to a reasonable range. □

Problem 4. How many nested loops are implied by each of the following MATLAB commands? (*i.e.* how many loops would you need to write to code the equivalent in C or fortran?) `A = rand(100,100); x = 1:100; b = A*x'; B = A*A;`

Proof. To show how many implied loops there are for each MATLAB command, I will write (naïve) equivalent code in C.

```
int A[100][100], B[100][100];
int x[100], b[100];
int i,j;

// MATLAB command: A = rand(100,100);
for (i=0; i<100; i++)
  for (j=0; j< 100; j++)
    A[i][j] = rand();

// MATLAB command: x = 1:100;
for (i=0; i<100; i++)
  x[i] = i+1;

// MATLAB command: b = A*x';
for (i=0; i<100; i++) {
  b[i] = 0;
  for (j=0; j<100; j++)
    b[i] += A[i][j] * x[j];
}

// MATLAB command: B = A*A;
for (i=0; i<100; i++) {
  for (j=0; j<100; j++) {
    B[i][j] = 0;
    for(k=0; k<100; k++)
      B[i][j] += A[i][k] * A[k][j]
  }
}
```

And so we would be naively led to believe that the commands would require 2, 1, 2, and 3 nested loops, respectively. However, we can execute the `A = rand(100,100);` command in a single loop from `i=1:10000`. We can likewise reduce the number of loops in the `B = A*A;` command to two by the same method. In this way, we could execute these commands with 1, 1, 2, and 2 nested loops. □

Problem 5. Give an exact formula, in terms of β and t for the smallest positive integer n that does not belong to the floating-point system $\mathbb{F}$, and compute n for IEEE single- and double-precision. Give one line of code, and its output, which demonstrates this is indeed the case for double-precision.

Proof. Fix $t \in \mathbb{Z}^+$ and $\beta \geq 2$. The set $\mathbb{F}$ is given by the construction

$$\mathbb{F} = \{ \pm m\beta^{e-t} : e \in \mathbb{Z}, m \in [\beta^{t-1}, \beta^t - 1] \} \cup \{ 0, \pm\infty, \text{NaN} \}$$

as discussed in class. Restricting our attention to the positive reals in $\mathbb{F}$, it is easily seen that

$$\mathbb{F} \cap \mathbb{R}^+ = \bigcup_{e\in\mathbb{Z}} [\beta^e, \beta^{e+1}] = \bigcup_{e\in\mathbb{Z}} \{ \beta^e + m \cdot \beta^{e-t} : m \in [0, \beta^{t+1}] \}$$

To determine the smallest integer not contained in $\mathbb{F} \cap \mathbb{R}^+$, we consider the relative difference between elements in $[\beta^e, \beta^{e+1}]$ and only consider the smallest positive e where β^{e-t} is strictly greater than 1. We choose this value of e, because adjacent ordered values in the corresponding set will be separated by exactly β, and since $\beta \geq 2$, this is the first set where we are guaranteed to skip integers. This phenomena starts occurring when $e = t + 1$, and so we claim that the smallest positive integer not contained in $\mathbb{F} \cap \mathbb{R}^+$ is given by the formula

$$n = \beta^{t+1} + 1.$$

To see that this is the smallest such integer, choose $k < \beta^{t+1}$ and let $d \in \mathbb{Z}$ be such that $\beta^d \leq k < \beta^{d+1}$; it's clear that $d \leq t$. Our claim is that $k \in [\beta^d, \beta^{d+1}]$. To this end, we must show that $k = \beta^d + m \cdot \beta^{d-t}$ for some $m \in [0, \beta^{t+1}]$. To prove this, it suffices to solve for $m = k \cdot \beta^{t-d} - \beta^t$, which is clearly an integer since $t - d \geq 0$, and show that $0 \leq m \leq \beta^{t+1}$. Since $\beta^d \leq k$, then $\beta^t \leq k \cdot \beta^{t-d}$, hence $m \geq 0$. Moreover, since $k \leq \beta^{d+1}$, then $k \cdot \beta^{t-d} \leq \beta^{t+1}$, hence $m \leq \beta^{t+1}$ as well. This proves that $k \in \mathbb{F} \cap \mathbb{R}^+$, and we therefore conclude that $n = \beta^{t+1} + 1$ is the smallest positive integer not contained in $\mathbb{F}$.

For IEEE double-precision, we take $\beta = 2$ and $t = 53$, and so can see that $2^{53} + 1$ is not represented by executing the following line of code:

```
sprintf('%f %f %f', 2^53-1, 2^53, 2^53+1)
```

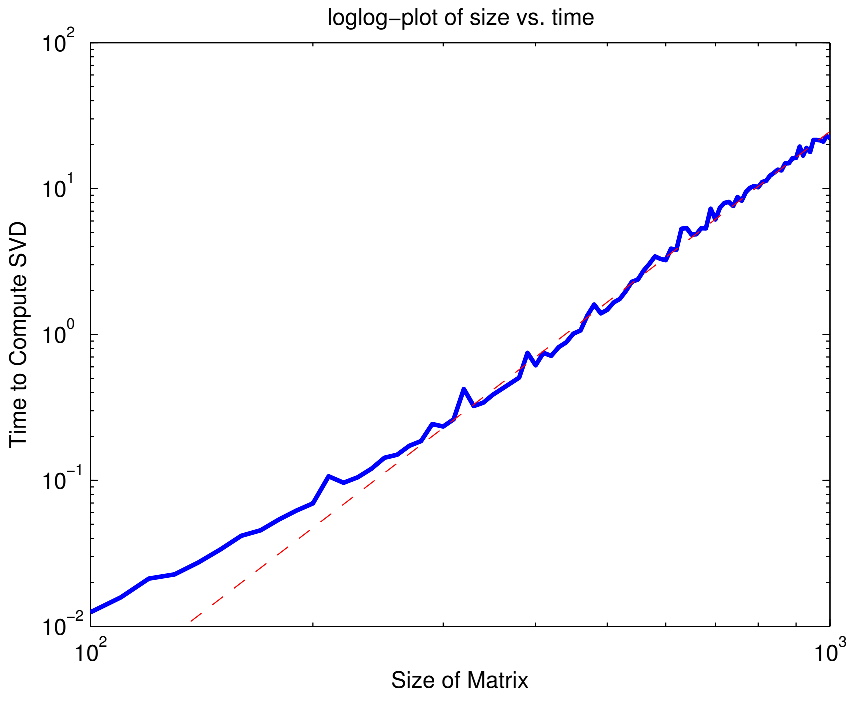
which produces the following integers:

```
9007199254740991    9007199254740992    9007199254740992
```

and so we can see that MATLAB was unable to represent $2^{53} + 1 = 9007199254740993$ in double-precision. □

Problem 6. Measure how the time to compute the singular values of a random real dense $m \times m$ matrix scales with m, focusing on the range $10^2 \leq m \leq 10^3$. Produce a log-log graph of time vs m, and the simple power law to which it is asymptotic. BONUS: for what large m would you expect this to break down and why?

Solution. As per the problem description, the following diagram was produced:



The red line was added after the fact to illustrate the asymptotic linear nature of the log-log plot. The slope of this line was found to be approximately $\alpha = 3.86$, and so one might predict that the time to compute the SVD of a matrix obeys the power law

$$t(m) = m^4.$$

$\square$

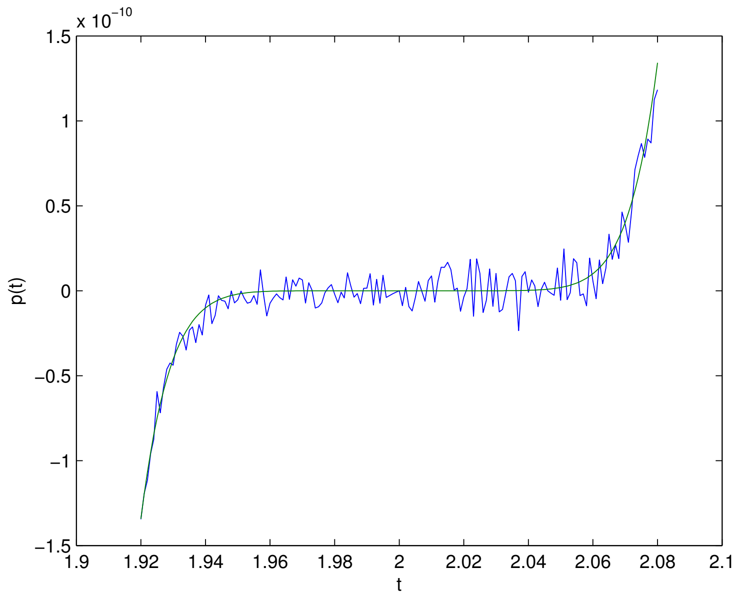
Problem 7. Consider the polynomial $p(x) = (x-2)^9 = x^9 - 18x^8 + 144x^7 - 672x^6 + 2016x^5 - 4032x^4 + 5376x^3 - 4608x^2 + 2304x - 512$.

(a) plot $p(x)$ for $x = 1.920, 1.921, 1.922, \ldots, 2.080$ evaluating p via its coefficients $1, -18, 144, \ldots$.

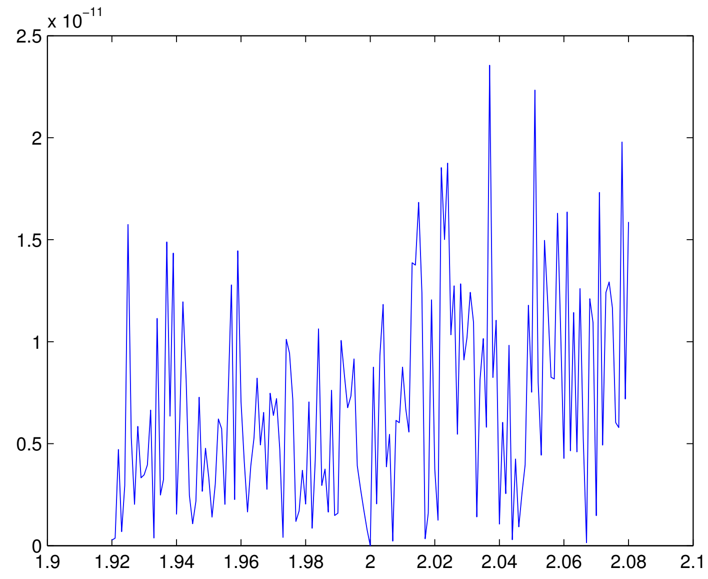
(b) overlay on your plot the same computed using $(x-2)^9$.

(c) explain, including the size of the effect!

Proof. In the following plot, we can see $p(x)$ plotted in two different ways.



The green graph illustrates the plot of $p(x) = (x-2)^9$ and the blue graph illustrates the plot of the same polynomial evaluated via its coefficient expansion. Additionally, the following graph depicts the absolute difference between the plots at each point plotted.



As you might expect, the absolute difference between the plots when they are evaluated at $x = 2$ is zero, since neither plot needs to perform floating-point operations.

These errors comes from the fact that when we calculate $p(x) = (x - 2)^9$, we only introduce 9 floating-point operations. However, when we calculate $p(x)$ in its expanded form, we perform 45 floating-point multiplications and then 9 floating-point additions. Since each floating-point operation introduces error, we naturally expect the method which uses fewer operations to be more accurate. $\square$