




## Spring 2022 Math 3607: Exam 2

Due: 6:00PM, Friday, March 11, 2022

Please read the statements below and sign your name.

### Disclaimers and Instructions



- You are **not** allowed to use MATLAB commands and functions **other than** the ones discussed in lectures, accompanying live scripts, textbooks, and homework/practice problem solutions.
- You may be requested to explain your code to me, in which case a proper and satisfactory explanation must be provided to receive any credits on relevant parts.
- You are **not** allowed to search online forums or even MathWorks website for this exam.
- You are **not** allowed to collaborate with classmates, unlike for homework.
- If any code is found to be plagiarized from the internet or another person, you will receive a zero on the *entire* exam and will be reported to the COAM.
- Do not carry out computations using *Symbolic Math Toolbox*. Any work done using `sym`, `syms`, `vpa`, and such will receive NO credit.
- **Notation.** Problems marked with  are to be done by hand; those marked with  are to be solved using a computer.
- Answers to analytical questions (ones marked with ) without supporting work or justification will not receive any credit.

---

### Academic Integrity Statements

- All of the work shown on this exam is my own.
- I will not consult with any resources (MathWorks website, online searches, etc.) other than the textbooks, lecture notes, supplementary resources provided on the course Carmen pages, or MATLAB's built-in help documentation.
- I will not discuss any part of this exam with anyone, online or offline.
- I understand that academic misconduct during an exam at The Ohio State University is very serious and can result in my failing this class or worse.
- I understand that any suspicious activity on my part will be automatically reported to the OSU Committee on Academic Misconduct (COAM) for their review.

Signature \_\_\_\_\_

**Notation.** Problems marked with  are to be done by hand; those marked with  are to be solved using a computer.


# 1 Catastrophic Cancellation

[25 points]

Let

$$f(x) = \begin{cases} \frac{1 - \cos x}{x^2} & \text{if } x \neq 0, \\ \frac{1}{2} & \text{if } x = 0. \end{cases}$$

We are interested in a stable numerical evaluation of  $f(x)$  for small  $x$ .

- (a)  Find the condition number  $\kappa_f(x)$ ; simplify it as much as you can. Then compute


$$\lim_{x \rightarrow 0} \kappa_f(x).$$

Based on the limit computation, is the evaluation of  $f(x)$  for small  $x$  well-conditioned or ill-conditioned?


- (b)  For small  $x$ , the “obvious” evaluation algorithm

$$f1 = @ (x) (1 - \cos (x)) ./ (x.^2);$$

suffers from catastrophic cancellation. Explain why.

- (c)  Using the first three nonzero terms of the Taylor series expansion of  $f(x)$ , establish an alternate algorithm `f2` to compute  $f(x)$  stably for small  $x$ . Fully justify the derivation of your algorithm.

**Note.** Pay attention to the underlined requirement. You may need to begin with more than three three nonzero terms of the Taylor series expansion of  $\cos x$ .

- (d)  Evaluate  $f(x)$  for  $x = 10^{-k}$  for  $k \in \mathbb{N}[1, 10]$  using the two algorithms `f1` and `f2`. Tabulate the results neatly. Comment on the results.

**Note.** The table should have three columns with the first being `x`, the second being `f1`, and the third being `f2`. Use either `format long g` or an appropriate `fprintf` statement to display full accuracy. Do it as efficiently as you can, avoiding the use of a loop whenever possible.

## 2 PLU Factorization

[25 points]

- (a) Complete the following MATLAB function `myplu.m` by filling in the main loop. Inside the loop, you are allowed use an if-statement, but a for-loop is NOT allowed. This program carries out PLU factorization of a square matrix and counts the number of row swaps. Include the function at the end of your live script.

```
function [L,U,P,s] = myplu(A)
% MYPLU    PLU factorization
% Input:
%   A      square matrix
% Output:
%   P,L,U  permutation, unit lower triangular, and upper triangular
%           matrices such that LU=PA
%   s      number of row swaps

%% Initialization/Preallocation
n = length(A);
P = eye(n); % preallocate P
L = eye(n); % preallocate L
s = 0;      % initialize s

%% Main Loop
for j = 1:n-1
    %% [FILL IN] Pivoting
    % (An if-statement is allowed.)

    %% [FILL IN] Introducing Zeros Below Diagonal
    % (A for-loop is NOT allowed.)

end

%% Clean-Up
U = triu(A);
end
```

- (b) Using the result from Lecture 16, write a MATLAB function `determinant2` that computes the determinant of a given matrix `A` using `myplu` function written for part (a). Include the function at the end of your live script.
- (c) Use your function and the built-in `det` on the matrices gallery('cauchy', n) for  $n = 3, 4, \dots, 8$ , and make a table using `fprintf` showing  $n$ , the determinant calculated using your function `determinant2`, and the relative error when compared to `det`.

### 3 QR Factorization<sup>1</sup>



[25 points]

The function `myqr` presented in Lecture 22 calculates the QR factorization quite inefficiently.

```

1 function [Q, R] = myqr(A)
2     [m, n] = size(A);
3     A0 = A;
4     Q = eye(m);
5     for j = 1:min(m,n)
6         Aj = A(j:m, j:n);
7         z = Aj(:, 1);
8         v = z + sign0(z(1))*norm(z)*eye(length(z), 1);
9         Hj = eye(length(v)) - 2/(v'*v) * v*v';
10        Aj = Hj*Aj;
11        H = eye(m);
12        H(j:m, j:m) = Hj;
13        Q = Q*H;
14        A(j:m, j:n) = Aj;
15    end
16    R = A;
17 end


```

- (a)  Determine the number of *flops* required in the code where the only two statements you need to consider are line 10, which updates  $A$  by  $A = HA$ , and line 13, which updates  $Q$  by  $Q = QH$ . Determine the number of *flops* for  $A \in \mathbb{R}^{m \times n}$ . Give an asymptotic answer.
- (b)  Modify the code to make it more efficient in terms of *flops*, following the suggestions below.
- $H_j$  does not need to be calculated explicit, so replace line 9 by the computation of  $\rho = 2/(\mathbf{v}^T \mathbf{v})$ . (The Greek letter  $\rho$  is to be spelled out as `rho` in your code, not as `raw` or `rou` or any other.)
  - Since  $H_j$  was not created, modify line 10 to update  $A$  by  $A = (I - \rho \mathbf{v} \mathbf{v}^T)A$ , where the right-hand side is converted to MATLAB in as efficient a form as possible.
  - Drop lines 11 and 12 and carry out the update of  $Q$ . Note that at the  $j$ -th iteration, only the  $j$ -th through the  $m$ -th columns of  $Q$  need to be modified because

$$\begin{bmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \end{bmatrix} \begin{bmatrix} I & O \\ O & \tilde{H}_j \end{bmatrix} = \begin{bmatrix} Q_{11} & Q_{12} \tilde{H}_j \\ Q_{21} & Q_{22} \tilde{H}_j \end{bmatrix}.$$

Again, the matrix multiplication  $Q_{i2} \tilde{H}_j$  is carried out by  $Q_{i2}(I - \rho \mathbf{v} \mathbf{v}^T)$  since  $H_j$  was not created. Convert it to MATLAB in as efficient a form as possible.

Design a test of your own and run it to confirm that your code works.

- (c)  Determine the number of *flops* of the modified code from part (b). The result must be much smaller than the one from part (a).

<sup>1</sup>This problem is from LM 12.6.

## 4 Visualization of Matrix Norms in 3-D

[25 points]

Recall that

$$\|A\|_p = \max_{\|\mathbf{x}\|_p=1} \|A\mathbf{x}\|_p, \quad p \in [1, \infty].$$

In this problem, we generate three-dimensional visualization of this definition. This is a direct extension of a recent homework problem.

- (a) Complete the following program which, given  $p \in [1, \infty]$  and  $A \in \mathbb{R}^{3 \times 3}$ , approximates  $\|A\|_p$  and plots the unit sphere in the  $p$ -norm and its image under  $A$ . Avoid using loops as much as possible. Include the function at the end of your live script.

```
function norm_A = visMatrixNorms3D(A, p)
    %% Basic checks
    if size(A,1)~=3 || size(A,2)~=3
        error('A must be a 3-by-3 matrix.')
    elseif p < 1
        error('p must be >= 1.')
    end

    %% Step 1: Initialization
    nr_th = 41; nr_ph = 31;
    th = linspace(0, 2*pi, nr_th);
    ph = linspace(0, pi, nr_ph);
    [T, P] = meshgrid(th, ph);
    x1 = cos(T).*sin(P);
    x2 = sin(T).*sin(P);
    x3 = cos(P);
    X = [x1(:), x2(:), x3(:)]';

    %% Step 2: [FILL IN] Normalize columns of X into unit vectors

    %% Step 3: [FILL IN] Form Y = A*X; calculate norms of columns of Y

    %% Step 4: [FILL IN] Calculate p-norm of A (approximate)

    %% Step 5: [FILL IN] Generate surface plots

end
```

The following steps are carried out by the program.

**Step 1.** Create 3-vectors

$$\mathbf{x}_k = \begin{bmatrix} \cos \theta_i \sin \phi_j \\ \sin \theta_i \sin \phi_j \\ \cos \phi_j \end{bmatrix}, \quad \text{for } 1 \leq i \leq 41, 1 \leq j \leq 31 \quad (1)$$

using 41 evenly distributed  $\theta_i$  in  $[0, 2\pi]$  and 31 evenly distributed  $\phi_j$  in  $[0, \pi]$ . Note the use of `meshgrid`, which is useful for surface plots later.

**Step 2.** Normalize  $\mathbf{x}_k$  into a unit vector in  $p$ -norm by  $\mathbf{x}_k \rightarrow \mathbf{x}_k / \|\mathbf{x}_k\|_p$  (that is, replacing  $\mathbf{x}_k$  with  $\mathbf{x}_k / \|\mathbf{x}_k\|_p$ ).

**Step 3.** For each  $k$ , let  $\mathbf{y}_k = A\mathbf{x}_k$ . Calculate and store  $\|\mathbf{y}_k\|_p$ .

**Step 4.** Approximate  $\|A\|_p$  based on the norms  $\|\mathbf{y}_k\|_p$  calculated in the previous step.

**Step 5.** Generate surface plots of the unit sphere in the  $p$ -norm and its image under  $A$ . Use `surf` function. Use `subplot` to put two graphs side by side.

(b) Run the program with  $p = 1, \frac{3}{2}, 2, 4$ , all with the same matrix

$$A = \begin{bmatrix} 2 & 0 & 0 \\ 0 & \cos(\pi/12) & -\sin(\pi/12) \\ 0 & \sin(\pi/12) & \cos(\pi/12) \end{bmatrix}, \quad (2)$$

by executing the following code block.

```
A = [2 0 0;
     0 cos(pi/12) -sin(pi/12);
     0 sin(pi/12) cos(pi/12)];
visMatrixNorms3D(A, 1); % Depending on how you write the code,
visMatrixNorms3D(A, 3/2); % you may need to use 'clf' or 'hold off'
visMatrixNorms3D(A, 2); % in between function calls.
visMatrixNorms3D(A, 4);
```

**x: Unit sphere in 2-norm**

**Ax: Image of unit sphere under A**

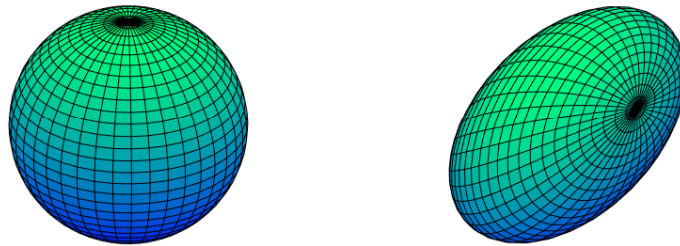


Figure 1: Example output.