

MA 580 Assignment 2

Kyle Hansen

26 September 2025

AI Use Statement: Here is where I put the AI use statemet

Question 1

Let $\mathbf{A} \in \mathbb{R}^{m \times n}$. Prove the following. In your solution use only the materials discussed in the lectures so far and results from the previous homework.

1(a) Show that $\lambda \neq 0$ is an eigenvalue of $\mathbf{A}^T \mathbf{A}$ if and only if it is an eigenvalue of $\mathbf{A} \mathbf{A}^T$. The nonzero eigenvalues of $\mathbf{A}^T \mathbf{A}$ are defined such that, for some $x \neq 0$,

$$(\mathbf{A}^T \mathbf{A})x = \lambda x \tag{1}$$

then it can be shown that this value λ is also an eigenvalue of $\mathbf{A} \mathbf{A}^T$:

$$\begin{aligned} \mathbf{A}^T \mathbf{A} x &= \lambda x \\ \mathbf{A} \mathbf{A}^T \mathbf{A} x &= \lambda \mathbf{A} x \end{aligned} \tag{2}$$

Then, for some $b = \mathbf{A} x$,

$$(\mathbf{A} \mathbf{A}^T) b = \lambda b \tag{3}$$

That is, all nonzero eigenvalues of $\mathbf{A}^T \mathbf{A}$ are also eigenvalues of $\mathbf{A} \mathbf{A}^T$.

The inverse can be shown similarly:

$$\begin{aligned} \mathbf{A} \mathbf{A}^T x &= \lambda x \\ \mathbf{A}^T \mathbf{A} \mathbf{A}^T x &= \lambda \mathbf{A}^T x \end{aligned} \tag{4}$$

Then for some $z = \mathbf{A}^T x$, $\mathbf{A}^T \mathbf{A} z = \lambda z$. Then the matrices $\mathbf{A} \mathbf{A}^T$ and $\mathbf{A}^T \mathbf{A}$ share all non-zero eigenvalues.

1(b) Show $\|\mathbf{A}^T\|_2 = \|\mathbf{A}\|_2$.

This can be shown using the spectral radius, since $\|\mathbf{A}\|_2 = \rho(\mathbf{A}^T \mathbf{A})^{1/2}$, which implies $\|\mathbf{A}^T\|_2 = \rho(\mathbf{A} \mathbf{A}^T)^{1/2}$.

Since $\mathbf{A} \mathbf{A}^T$ and $\mathbf{A}^T \mathbf{A}$ have the same nonzero eigenvalues, then they share the same spectral radius $\rho(\mathbf{A}) = \max|\lambda|$. Then:

$$\begin{aligned}\rho(\mathbf{A}^T \mathbf{A})^{1/2} &= \rho(\mathbf{A} \mathbf{A}^T)^{1/2} \\ \|\mathbf{A}\|_2 &= \|\mathbf{A}^T\|_2.\end{aligned}\tag{5}$$

1(c) Show $\|\mathbf{A} \mathbf{A}^T\|_2 = \|\mathbf{A}^T \mathbf{A}\|_2 = \|\mathbf{A}\|_2^2$.

Again using the spectral radius, these norms can be rewritten in terms of the largest eigenvalue of the two products:

$$\begin{aligned}\|\mathbf{A}^T \mathbf{A}\|_2 &= \rho\left((\mathbf{A}^T \mathbf{A})^2\right)^{1/2} \\ \|\mathbf{A} \mathbf{A}^T\|_2 &= \rho\left((\mathbf{A} \mathbf{A}^T)^2\right)^{1/2},\end{aligned}\tag{6}$$

where $(\mathbf{A}^T \mathbf{A})^T = \mathbf{A}^T \mathbf{A}$ and $(\mathbf{A} \mathbf{A}^T)^T = \mathbf{A} \mathbf{A}^T$.

Since eigenvalues of a squared matrix are the squared eigenvalues of the original matrix,

$$\begin{aligned}\rho\left((\mathbf{A}^T \mathbf{A})^2\right)^{1/2} &= \rho(\mathbf{A}^T \mathbf{A}) \\ \rho\left((\mathbf{A} \mathbf{A}^T)^2\right)^{1/2} &= \rho(\mathbf{A} \mathbf{A}^T).\end{aligned}\tag{7}$$

These terms are equal to the squares of Equation 5 from the previous subsection— since $\rho(\mathbf{A} \mathbf{A}^T)^{1/2} = \rho(\mathbf{A}^T \mathbf{A})^{1/2} = \|\mathbf{A}\|_2$, the same is true of their squares, and

$$\|\mathbf{A} \mathbf{A}^T\|_2 = \|\mathbf{A}^T \mathbf{A}\|_2 = \|\mathbf{A}\|_2^2\tag{8}$$

1(d) Suppose $\mathbf{A} \in \mathbb{R}^{n \times n}$ is nonsingular. Show $\kappa_2(\mathbf{A}^T \mathbf{A}) = \kappa_2(\mathbf{A})^2$.

The condition number of the product is defined as

$$\kappa_2(\mathbf{A}^T \mathbf{A}) = \|\mathbf{A}^T \mathbf{A}\|_2 \left\| (\mathbf{A}^T \mathbf{A})^{-1} \right\|_2.\tag{9}$$

From the previous section, $\|\mathbf{A}\mathbf{A}^T\|_2 = \|\mathbf{A}^T\mathbf{A}\|_2 = \|\mathbf{A}\|_2^2$, so

$$\kappa_2(\mathbf{A}^T\mathbf{A}) = \|\mathbf{A}\|_2^2 \left\| (\mathbf{A}^T\mathbf{A})^{-1} \right\|_2. \quad (10)$$

Where the inverse term can be expanded as $(\mathbf{A}^T\mathbf{A})^{-1} = \mathbf{A}^{-1}(\mathbf{A}^T)^{-1}$. Since the inverse of a transpose is the transpose of an inverse, this can be rewritten $\mathbf{A}^{-1}(\mathbf{A}^{-1})^T$. The same property as above can be applied again, since $\|\mathbf{A}^{-1}(\mathbf{A}^{-1})^T\|_2 = \|(\mathbf{A}^{-1})^T\mathbf{A}^{-1}\|_2 = \|\mathbf{A}^{-1}\|_2^2$, and

$$\kappa_2(\mathbf{A}^T\mathbf{A}) = \|\mathbf{A}\|_2^2 \|\mathbf{A}^{-1}\|_2^2 = (\|\mathbf{A}\|_2 \|\mathbf{A}^{-1}\|_2) = \kappa_2(\mathbf{A}^2) \quad (11)$$

Question 2

Let $\mathbf{A} \in \mathbb{R}^{n \times n}$ be nonsingular, and suppose $\mathbf{E} \in \mathbb{R}^{n \times n}$ satisfies $\|\mathbf{E}\| \leq \frac{1}{2\|\mathbf{A}^{-1}\|}$, where $\|\cdot\|$ is an induced matrix norm. Then, $\mathbf{A} + \mathbf{E}$ is nonsingular, and

$$\|(\mathbf{A} + \mathbf{E})^{-1}\| \leq 2\|\mathbf{A}^{-1}\|.$$

Since \mathbf{A} is nonsingular, the addition can be rewritten

$$\mathbf{A} + \mathbf{E} = \mathbf{A} + \mathbf{A}\mathbf{A}^{-1}\mathbf{E} = \mathbf{A}(\mathbf{I} + \mathbf{A}^{-1}\mathbf{E}) \quad (12)$$

The condition $\|\mathbf{E}\| \leq 1/2\|\mathbf{A}^{-1}\|$ can be rewritten (using the Cauchy-Schwarz inequality)

$$\begin{aligned} \|\mathbf{E}\| &\leq \frac{1}{2\|\mathbf{A}^{-1}\|} \\ 2\|\mathbf{A}^{-1}\mathbf{E}\| &\leq 2\|\mathbf{A}^{-1}\|\|\mathbf{E}\| \leq 1 \\ \|\mathbf{A}^{-1}\mathbf{E}\| &\leq 1/2 \end{aligned} \quad (13)$$

Then since $\|\mathbf{A}^{-1}\mathbf{E}\| \leq 1/2$, Fact 2.23 in [1] can be used to show that $(\mathbf{I} + \mathbf{A}^{-1}\mathbf{E})$ is nonsingular and

$$\left\| (\mathbf{I} + \mathbf{A}^{-1}\mathbf{E})^{-1} \right\| \leq 2. \quad (14)$$

Since both \mathbf{A} and $(\mathbf{I} + \mathbf{A}^{-1}\mathbf{E})$ are nonsingular and the product of two nonsingular matrices is also nonsingular, then the product $(\mathbf{A} + \mathbf{E}) = \mathbf{A}(\mathbf{I} + \mathbf{A}^{-1}\mathbf{E})$ is nonsingular.

Again using the Cauchy-Schwarz inequality and the result from Equation 14,

$$\|(\mathbf{A} + \mathbf{E})^{-1}\| = \left\| \mathbf{A}^{-1}(\mathbf{I} + \mathbf{A}^{-1}\mathbf{E})^{-1} \right\| \leq \|\mathbf{A}^{-1}\| \left\| (\mathbf{I} + \mathbf{A}^{-1}\mathbf{E})^{-1} \right\| \leq 2\|\mathbf{A}^{-1}\|. \quad (15)$$

Question 3

Let $\|\cdot\|$ be a vector norm. Let \mathbf{A} be a nonsingular matrix. Consider the maximum and minimum magnification, $\mathbf{maxmag}(\mathbf{A})$ and $\mathbf{minmag}(\mathbf{A})$.

Show that

$$\kappa(\mathbf{A}) = \frac{\mathbf{maxmag}(\mathbf{A})}{\mathbf{minmag}(\mathbf{A})}.$$

Beginning from the definitions

$$\begin{aligned}\mathbf{minmag}(\mathbf{A}) &= \min \frac{\|\mathbf{A}x\|}{\|x\|} \\ \mathbf{maxmag}(\mathbf{A}) &= \max \frac{\|\mathbf{A}x\|}{\|x\|}\end{aligned}\tag{16}$$

The \mathbf{minmag} can be redefined using the property that the inverse of a maximum is the maximum of an inverse:

$$\mathbf{minmag}(\mathbf{A}) = \min \frac{\|\mathbf{A}x\|}{\|x\|} = \min \frac{\|b\|}{\|\mathbf{A}^{-1}b\|} = \frac{1}{\max \frac{\|\mathbf{A}^{-1}b\|}{\|b\|}} = \frac{1}{\|\mathbf{A}^{-1}\|}\tag{17}$$

Then, since $\mathbf{maxmag}(\mathbf{A}) = \|\mathbf{A}\|$ (by the interpretation discussed in class), $\mathbf{minmag}(\mathbf{A}) = \|\mathbf{A}^{-1}\|^{-1}$ (by Equation 17), and $\kappa(\mathbf{A}) = \|\mathbf{A}\| \|\mathbf{A}^{-1}\|$, the condition can be rewritten:

$$\kappa(\mathbf{A}) = \|\mathbf{A}\| \|\mathbf{A}^{-1}\| = \|\mathbf{A}\| \left(\frac{1}{\|\mathbf{A}^{-1}\|} \right)^{-1} = \frac{\mathbf{maxmag}(\mathbf{A})}{\mathbf{minmag}(\mathbf{A})}.\tag{18}$$

Question 4

Computer problem, Python

Computer problem. Consider the following elliptic partial differential equation (PDE).

$$\begin{aligned}-\Delta u(x, y) &= f(x, y) && \text{in } \Omega = (0, 1) \times (0, 1), \\ u &= 0, && \text{on } \partial\Omega.\end{aligned}\tag{19}$$

4(a) Develop a routine that given the right-hand-side function f and the number n , returns the coefficient matrix and right-hand side vector for the linear system resulting from the finite-difference discretization of the problem, using sparse matrix storage.

I used the same sample code provided in the appendix, adapted to function in Python. Since the 2D finite difference scheme is given by

$$\Delta u \approx 4u_{i,j} - u_{i+1,j} - u_{i-1,j} - u_{i,j+1} - u_{i,j-1} \quad (20)$$

Since $1/h^2$ appears in every entry of the matrix, it can be factored out for simplicity (now solving instead the system $h^2 \mathbf{A} \mathbf{x} = h^2 \mathbf{b}$).

The matrix \mathbf{A} should then have 4 in every diagonal entry, and using the scheme $k = i + (j - 1)(n - 1)$ to convert the 2D coordinate to an index, -1 should appear one entry to the left and right of the diagonal (corresponding with $i \pm 1$) and, as well as $(n - 1)$ entries to the left and right (corresponding with $j \pm 1$). The boundary conditions mean that some rows appear without all 5 coefficients—the matrix is only $(n - 1) \times (n - 1)$.

The vector b is simply set to $b_k = h^2 u_k$, where $u_k = u_{i,j}$ using the flattening algorithm described in the assignment’s appendix.

The matrix and vector assignment appear in the `system_def` function in the Python code.

4(b) Solve the problem with

$$f(x, y) = -2\pi^2(\cos^2(\pi x) \sin^2(2\pi y) - 5 \sin^2(\pi x) \sin^2(2\pi y) + 4 \sin^2(\pi x) \cos^2(2\pi y)), \quad (21)$$

and with $n = 2^k, k = 3, \dots, 10$ using Gaussian elimination. Note that for this choice of f the analytic solution is given by,

$$u(x, y) = \sin^2(\pi x) \sin^2(2\pi y).$$

The code is presented in Appendix A. As an example, a numerical solution is shown in Figure 1 and its absolute error is shown in Figure 2.

The results for all run cases are presented in Table 1. The results clearly demonstrate a logarithmic convergence rate, where each doubling of n (or quadrupling of n^2) results in a reduction in error by one quarter. Because of this convergence rate, a linear increase in the number of meshpoints (which results in significantly increased computational cost) has a diminishing effect on error reduction.

4(c) Scipy’s `linalg.onenormest` could be used to estimate the one norm $\|\mathbf{A}\|_1$, but does not directly compute the condition number, which requires $\|\mathbf{A}^{-1}\|_1$, or an appropriate estimate.

Matlab provides the function `condest`, which uses an iterative method [2] that computes the gradient of $f(\mathbf{x}) = \|\mathbf{A}\mathbf{x}\|_1$, in order to converge on an upper bound for $\|\mathbf{A}^{-1}\|_1$. When I tested my own implementation of this algorithm I found its results to be equivalent to directly computing the matrix inverse and using `onenormest`. This iterative estimate appears as `hager_invnorm` in the Python code.

Table 1: Convergence results for two-dimensional Poisson problem

h	no. of unknowns	$\ e_h\ _\infty$	$\ e_{2h}\ _\infty / \ e_h\ _\infty$
0.125	8^2	$1.72 \cdot 10^{-1}$	0.2298
0.0625	16^2	$3.96 \cdot 10^{-2}$	0.2449
0.03125	32^2	$9.70 \cdot 10^{-3}$	0.2487
0.01563	64^2	$2.41 \cdot 10^{-3}$	0.2499
0.00781	128^2	$6.03 \cdot 10^{-3}$	0.2500
0.00391	256^2	$1.51 \cdot 10^{-4}$	0.2500
0.00195	512^2	$3.77 \cdot 10^{-4}$	0.2500
0.00098	1024^2	$9.42 \cdot 10^{-5}$	—

Numerical solution, $h = 0.0009765625$

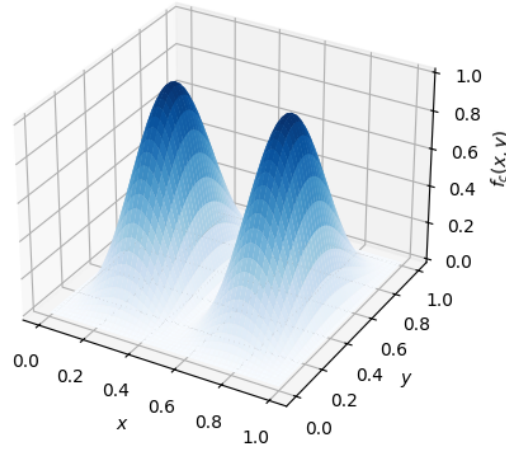


Figure 1: Numerical solution, $n = 2^{10}$

The condition numbers for n up to 2^{10} are shown in Figure 3. κ shows a clear $\kappa \propto n^{\approx 2}$ increase, clearly justifying the need for a more stable algorithm, potentially different than the GE scheme used in this solve.

Absolute Error, $h = 0.0009765625$

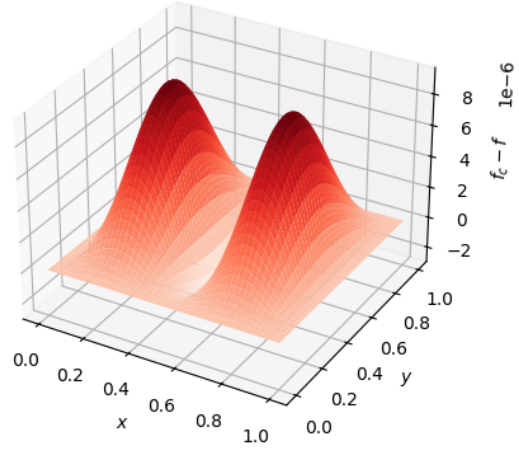


Figure 2: Absolute error, $n = 2^{10}$

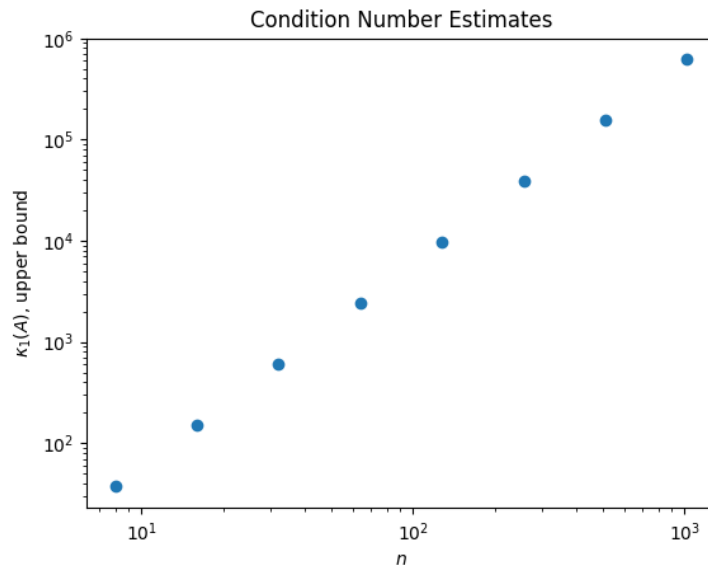


Figure 3: Condition number estimates, $\kappa_1(\mathbf{A})$

References

- [1] I. C. Ipsen, *Numerical matrix analysis: Linear systems and least squares*. SIAM, 2009.
- [2] W. W. Hager, “Condition estimates,” *SIAM Journal on scientific and statistical computing*, vol. 5, no. 2, pp. 311–316, 1984.

Question A Python Code

```
1  import numpy
2  # import scipy
3  import scipy.sparse as sparse
4  import matplotlib
5  import matplotlib.pyplot as plt
6
7  pi = numpy.pi
8  # part a -- matrix assignment
9  def source_function(x, y):
10     z = -2*(pi**2)*((numpy.cos(pi*x)**2 * numpy.sin(2*pi*y)**2)
11                    -5*( numpy.sin(pi*x)**2 * numpy.sin(2*pi*y)**2)
12                    +4*(numpy.sin(pi*x)**2 * numpy.cos(2*pi*y)**2))
13     return z
14
15  def analytic_solution(x, y):
16     return numpy.sin(pi*x)**2 * numpy.sin(2*pi*y)**2
17
18  class poisson_system:
19     def __init__(self, x, y, A, f):
20         self.x_mesh = x
21         self.y_mesh = y
22         self.coefficients = A
23         self.source = f
24
25  def unpack_1d_array(a):
26     m = int(numpy.sqrt(a.size))
27     n = m+1
28     A = numpy.zeros((m,m))
29     for i in range(0, m):
30         for j in range(0, m):
31             A[i][j] = a[i+ (j)*(m)]
32     return A
33
34  def system_def(f, n):
35     m = n-1
36     I = sparse.eye(m,m)
```



```

37     e = numpy.ones(m)
38     T = sparse.diags([-e, 4*e, -e], [-1, 0, 1], (m,m))
39     S = sparse.diags([-e, -e], [-1, 1], (m,m))
40     A = sparse.kron(I, T) + sparse.kron(S, I)
41
42     h = 1/n
43     x = numpy.arange(h, 1, h)
44     y = numpy.arange(h, 1, h)
45     q_vec = numpy.zeros((n-1)**2)
46
47     for i in range(0, (n-1)):
48         for j in range(0, (n-1)):
49             k = i+ (j)*(n-1)
50             q_vec[k] = (h**2)*source_function(x[i], y[j])
51
52     return poisson_system(x, y, A, q_vec)
53
54 def hager_invnorm(A):
55     # Hager 1984, Higham 1988
56     n = A.shape[0]
57     x = (1/n)*numpy.ones((n))
58     while (True):
59         y = sparse.linalg.spsolve(A, x)
60         # heaviside instead of sign() so that sign(0) = 1
61         xi = 2*numpy.heaviside(y, 1) - 1
62         z = sparse.linalg.spsolve(A,xi)
63         if (numpy.max(abs(z)) <= z@x):
64             return numpy.sum(abs(y))
65         x = numpy.zeros(n)
66         x[numpy.argmax(z)] = 1
67
68
69     k = numpy.arange(3, 11, 1)
70     n = 2**k
71     h = 1/n
72     e = numpy.zeros(n.shape)
73     r = e.copy()
74     cond = e.copy()
75
76     for ni in range(0, n.size):
77         print("n= "+str(n[ni]))
78         system = system_def(source_function, n[ni])
79         A = system.coefficients
80         x = system.x_mesh
81         y = system.y_mesh

```

```

82     q = system.source
83
84     u = sparse.linalg.spsolve(A, q)
85     U = unpack_1d_array(u)
86
87     K = numpy.zeros(U.shape)
88     for i in range(0, K.shape[0]):
89         for j in range(0, K.shape[1]):
90             K[i][j] = analytic_solution(x[i], y[j])
91
92     e[ni] = numpy.max(numpy.abs(K-U))
93     cond[ni] = sparse.linalg.onenormest(A)*hager_invnorm(A)
94     print(cond[ni])
95     if (ni>0):
96         r[ni] = e[ni]/e[ni-1]
97     if n[ni] == 2**10:
98         x, y = numpy.meshgrid(x, y)
99
100         fig = plt.figure()
101         ax = plt.axes(projection="3d")
102         ax.plot_surface(x, y, (U-K), cmap=matplotlib.cm.Red)
103         plt.title("Absolute Error, $h="+str(h[ni])+"$")
104         ax.set_xlabel("$x$")
105         ax.set_ylabel("$y$")
106         ax.set_zlabel("$f_c - f$")
107         plt.show()
108         # plt.savefig("abs_errs")
109         plt.close()
110
111         fig = plt.figure()
112         ax = plt.axes(projection="3d")
113         ax.plot_surface(x, y, U, cmap=matplotlib.cm.Blues)
114         plt.title("Numerical solution, $h="+str(h[ni])+"$")
115         ax.set_xlabel("$x$")
116         ax.set_ylabel("$y$")
117         ax.set_zlabel("$f_c(x,y)$")
118         plt.show()
119         # plt.savefig("num_soln")
120         plt.close()
121
122     fig = plt.figure()
123     plt.scatter(n, cond)
124     plt.title("Condition Number Estimates")
125     plt.xlabel("$n$")
126     plt.ylabel("$\\kappa_1(A)$, upper bound")

```

```
127 plt.show()
128 # plt.savefig("condition_linear")
129 plt.xscale("log")
130 plt.yscale("log")
131 plt.show()
132 # plt.savefig("condition_log")
133 plt.close()
134
```