MA 580 Assignment 1

Kyle Hansen

5 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 $\mathbf{A}\mathbf{A}^T$.

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

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

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

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

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

[Hint: You can use Fact 2.23 in Ipsen. You are essentially proving a special case of Corollary 2.24 in that book. Provide a complete proof, do not just quote that corollary.]

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

$$\mathbf{A} + \mathbf{E} = \mathbf{A} + \mathbf{A}\mathbf{A}^{-1}\mathbf{E} = \mathbf{A}\left(\mathbf{I} + \mathbf{A}^{-1}\mathbf{E}\right) \tag{1}$$

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

$$\begin{gathered} \|\mathbf{E}\| \leq \frac{1}{2\,\|\mathbf{A}^{-1}\|} \\ 2\left\|\mathbf{A}^{-1}\mathbf{E}\right\| \leq 2\left\|\mathbf{A}^{-1}\right\|\|\mathbf{E}\| \leq 1 \\ \left\|\mathbf{A}^{-1}\mathbf{E}\right\| \leq 1/2 \end{gathered} \tag{2}$$

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

$$\left\|\left(\mathbf{I} + \mathbf{A}^{-1}\mathbf{E}\right)^{-1}\right\| \leq 2. \tag{3}$$

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 3,

$$\left\|(\mathbf{A} + \mathbf{E})^{-1}\right\| = \left\|\mathbf{A}^{-1}\left(\mathbf{I} + \mathbf{A}^{-1}\mathbf{E}\right)^{-1}\right\| \leq \left\|\mathbf{A}^{-1}\right\|\left\|\left(\mathbf{I} + \mathbf{A}^{-1}\mathbf{E}\right)^{-1}\right\| \leq 2\left\|\mathbf{A}^{-1}\right\|. \tag{4}$$

Question 3

This problem provides a geometric interpretation of the condition number of a matrix. Let $\|\cdot\|$ be a vector norm. Let $\mathbf{A}$ be a nonsingular matrix. Consider the maximum and minimum magnification, $\mathbf{maxmag}(\mathbf{A})$ and $\mathbf{minmag}(\mathbf{A})$, which we defined in the class. Show that

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

Begining from the definitions

$$\begin{aligned} \mathbf{minmag}(\mathbf{A}) &= \min\frac{\|\mathbf{A}x\|}{\|x\|} \\ \mathbf{maxmag}(\mathbf{A}) &= \max\frac{\|\mathbf{A}x\|}{\|x\|} \end{aligned} \tag{5}$$

The **minmag** can be redefined using the property that that inverse of a maximum is the maximum of an inverse:

$$\mathbf{minmag}(\mathbf{A}) = \min \frac{\|\mathbf{A}x\|}{\|x\|} = \min \frac{\|b\|}{\|\mathbf{A}^{-1}b\|} = \frac{1}{\max \frac{\|\mathbf{A}^{-1}b\|}{\|b\|}} = \frac{1}{\|\mathbf{A}^{-1}\|} \tag{6}$$

Then, since $\mathbf{maxmag}(\mathbf{A}) = \|\mathbf{A}\|$ (by the interpretation discussed in class), $\mathbf{minmag}(\mathbf{A}) = \|\mathbf{A^{-1}}\|^{-1}$ (by Equation 6), and $\kappa(\mathbf{A}) = \|\mathbf{A}\| \|\mathbf{A}^{-1}\|$, the condition can be rewritten:

$$\kappa(\mathbf{A}) = \|\mathbf{A}\| \left\|\mathbf{A}^{-1}\right\| = \|\mathbf{A}\| \left(\frac{1}{\|\mathbf{A}^{-1}\|}\right)^{-1} = \frac{\mathbf{maxmag}(\mathbf{A})}{\mathbf{minmag}(\mathbf{A})}. \tag{7}$$

Question 4

Computer problem, Python

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

$$\begin{aligned} -\Delta u(x,y) &= f(x,y) \qquad \text{in } \Omega = (0,1) \times (0,1), \\ u &= 0, \qquad\qquad \text{on } \partial\Omega. \end{aligned} \tag{8}$$

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

$$\Delta u \approx 4u_{i,j} - u_{i+1,j} - u_{i-1,j} - u_{i,j+1} - u_{i,j-1} \tag{9}$$

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

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 flatting algorithm described in the assignment's appendix.

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

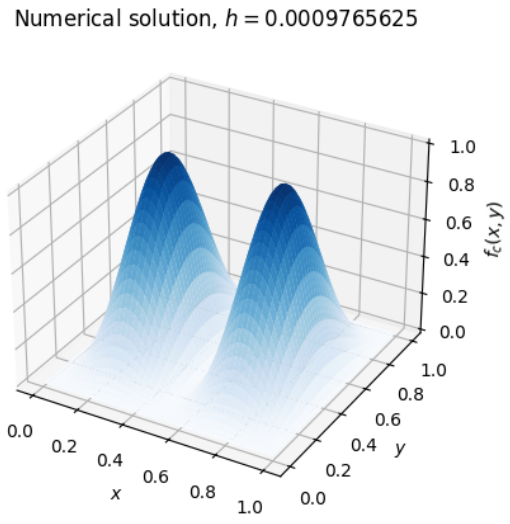

Figure 1: Numerical solution, $n = 2^{10}$

4(b) Solve the problem with

$$f(x, y) = -2\pi^2(\cos^2(\pi x)\sin^2(2\pi y) - 5\sin^2(\pi x)\sin^2(2\pi y) + 4\sin^2(\pi x)\cos^2(2\pi y)), \quad (10)$$

and with $n = 2^k, k = 3, \ldots, 10$ 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.

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 $f(\boldsymbol{x}) = \|\mathbf{A}\boldsymbol{x}\|_1$, 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.

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

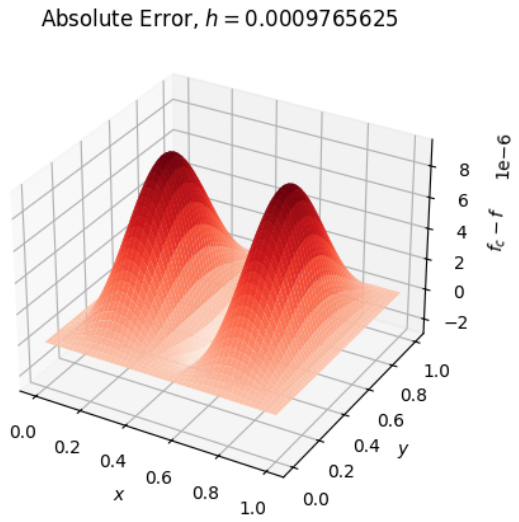

Figure 2: Absolute error, $n = 2^{10}$

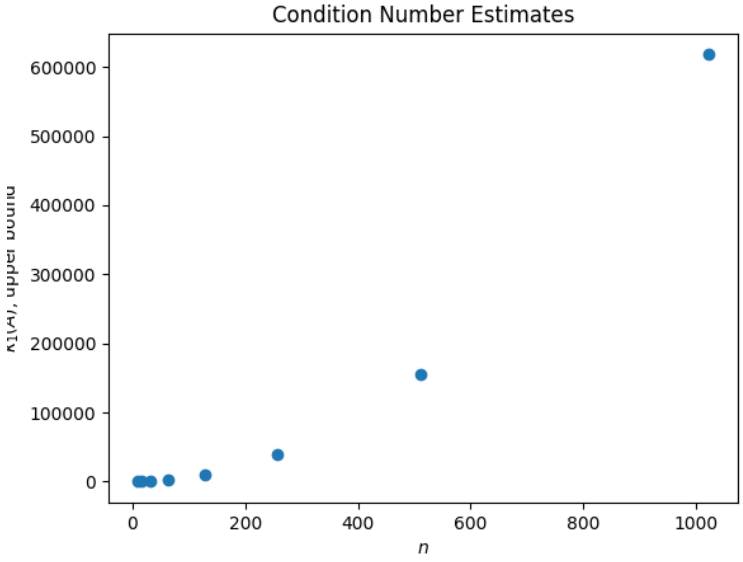

Figure 3: Condition number estimates, $\kappa_1(\mathbf{A})$

```
import numpy
import scipy
import scipy.sparse as sparse
import matplotlib
import matplotlib.pyplot as plt

pi = numpy.pi
# part a -- matrix assignment
def source_function(x, y):
    z = -2*(pi**2)*((numpy.cos(pi*x)**2 * numpy.sin(2*pi*y)**2)
                    -5*( numpy.sin(pi*x)**2 * numpy.sin(2*pi*y)**2)
                    +4*(numpy.sin(pi*x)**2 * numpy.cos(2*pi*y)**2))
    return z

def analytic_solution(x, y):
    return numpy.sin(pi*x)**2 * numpy.sin(2*pi*y)**2

class poisson_system:
    def __init__(self, x, y, A, f):
        self.x_mesh = x
        self.y_mesh = y
        self.coefficients = A
        self.source = f

def unpack_1d_array(a):
    m = int(numpy.sqrt(a.size))
    n = m+1
    A = numpy.zeros((m,m))
    for i in range(0, m):
        for j in range(0, m):
            A[i][j] = a[i+ (j)*(m)]
    return A

def system_def(f, n):
    m = n-1
    I = sparse.eye(m,m)
    e = numpy.ones(m)
    T = sparse.diags([-e, 4*e, -e], [-1, 0, 1], (m,m))
    S = sparse.diags([-e, -e], [-1, 1], (m,m))
    A = sparse.kron(I, T) + sparse.kron(S, I)
    h = 1/n
    x = numpy.arange(h, 1, h)
    y = numpy.arange(h, 1, h)
    q_vec = numpy.zeros((n-1)**2)
    for i in range(0, (n-1)):
```

```
46          for j in range(0, (n-1)):
47              k = i+ (j)*(n-1)
48              q_vec[k] = (h**2)*source_function(x[i], y[j])
49      return poisson_system(x, y, A, q_vec)
50
51  def hager_invnorm(A):
52      # Hager 1984, Higham 1988
53      n = A.shape[0]
54      x = (1/n)*numpy.ones((n))
55      while (True):
56          y = sparse.linalg.spsolve(A, x)
57          # heaviside instead of sign() so that sign(0) = 1
58          xi = 2*numpy.heaviside(y, 1) - 1
59          z = sparse.linalg.spsolve(A,xi)
60          if (numpy.max(abs(z)) <= z@x):
61              return numpy.sum(abs(y))
62          x = numpy.zeros(n)
63          x[numpy.argmax(z)] = 1
64
65
66  k = numpy.arange(3, 9, 1)
67  n = 2**k
68  h = 1/n
69  e = numpy.zeros(n.shape)
70  r = e.copy()
71  cond = e.copy()
72
73  for ni in range(0, n.size):
74      print("n= "+str(n[ni]))
75      system = system_def(source_function, n[ni])
76      A = system.coefficients
77      x = system.x_mesh
78      y = system.y_mesh
79      q = system.source
80
81      u = sparse.linalg.spsolve(A, q)
82      U = unpack_1d_array(u)
83
84      K = numpy.zeros(U.shape)
85      for i in range(0, K.shape[0]):
86          for j in range(0, K.shape[1]):
87              K[i][j] = analytic_solution(x[i], y[j])
88
89      e[ni] = numpy.max(numpy.abs(K-U))
90      I = sparse.eye(A.shape[0], A.shape[1])
```

```
    A_inv = sparse.linalg.spsolve(A, I)
    cond[ni] = sparse.linalg.onenormest(A)*hager_invnorm(A)
    # cond[ni] = numpy.linalg.cond(A.toarray(), p=1)
    print(cond[ni])
    if (ni>0):
        r[ni] = e[ni]/e[ni-1]
    if n[ni] == 2**10:
        x, y = numpy.meshgrid(x, y)

        fig = plt.figure()
        ax = plt.axes(projection="3d")
        ax.plot_surface(x, y,(U-K), cmap=matplotlib.cm.Reds)
        plt.title("Absolute Error, h="+str(h[ni]))
        plt.show()

        fig = plt.figure()
        ax = plt.axes(projection="3d")
        ax.plot_surface(x, y, U, cmap=matplotlib.cm.Blues)
        plt.title("Numerical solution, h="+str(h[ni]))
        plt.show()

print("h")
print(h)
print("n")
print(n)
print("e")
print(e)
print("r")
print(r)
print("condition number")
print(cond)
```

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