Lab 1

GMRES

Lab Objective: *In this lab we will learn how to use the GMRES algorithm.*

The GMRES ("Generalized Minimal Residuals") algorithm is an effecient way to solve large linear systems. It is an iterative method that uses Krylov subspaces to reduce a high-dimensional problem to a sequence of smaller dimensional problems.

The GMRES Algorithm

Let A be an invertible $m \times m$ matrix and let $\mathbf{b}$ be an m-vector. Let $\mathcal{K}_n(A, \mathbf{b})$ be the order-n Krylov subspace generated by A and $\mathbf{b}$. The idea of the GMRES algorithm is that instead of solving $A\mathbf{x} = \mathbf{b}$ directly, we solve the least squares problem

$$\underset{\mathbf{x} \in \mathcal{K}_n}{\text{minimize}} \qquad \|\mathbf{b} - A\mathbf{x}\|_2 \tag{1.1}$$

for increasing values of n. At each iteration, we compute the *residual*, or error between the least squares solution and a true solution of $A\mathbf{x} = \mathbf{b}$. The algorithm returns when this residual is sufficiently small. In good circumstances, this will happen when n is still much less than m.

The GMRES algorithm is integrated with the Arnoldi iteration for numerical stabiility, so that instead of solving (1.1), at the n^{th} iteration we solve

$$\underset{\mathbf{y} \in \mathbb{F}^n}{\text{minimize}} \qquad \|H_n\mathbf{y} - \|\mathbf{b}\|_2\mathbf{e}_1\|_2. \tag{1.2}$$

Here, H_n is the $(n+1) \times n$ upper Hessenberg matrix generated by the Arnoldi iteration and $\mathbf{e}_1$ is the vector $(1, 0, \ldots, 0)$ of length $n+1$. If $\mathbf{y}$ is the minimizer for the n^{th} iteration of (1.2), then the residual is

$$\frac{\|H_n\mathbf{y} - \|\mathbf{b}\|_2\mathbf{e}_1\|_2}{\|\mathbf{b}\|_2}, \tag{1.3}$$

and the corresponding minimizer for (1.1) is $Q_n\mathbf{y}$, where Q_n is the matrix whose columns are $\mathbf{q}_1, \ldots, \mathbf{q}_n$ as defined by the Arnoldi iteration. This algorithm is outlined in Algorithm 1.1. For a complete derivation see [TODO: ref textbook].

Algorithm 1.1 The GMRES algorithm. This algorithm operates on a vector $\mathbf{b}$ and matrix A. It iterates k times or until the residual is less than tol, returning an approximate solution to $A\mathbf{x} = \mathbf{b}$ and the error in this approximation.

```
1: procedure GMRES($A, \mathbf{b}, k, tol$)
2:     $Q \leftarrow$ size($\mathbf{b}$)$k + 1$                                  ▷ Initialize
3:     $H \leftarrow$ zeros($k + 1, k$)
4:     $Q[:, 0] = \mathbf{b}/\|\mathbf{b}\|_2$
5:     for $n = 1 \dots k$ do
6:         Set entries of $Q$ and $H$ as in Arnoldi iteration
7:         Calculate least squares solution $\mathbf{y}$ of 1.2.
8:         Calculate the residual $res$ given by Equation 1.3.
9:         if $res < tol$ then
10:            return $Q[:, n + 1]\mathbf{y},\ res$
11:    return $Q[:, n + 1]\mathbf{y},\ res$
```

Problem 1. Use Algorithm 1.1 to complete the following Python function implementing the GMRES algorithm.

```python
def gmres(b, Amul, k=100, tol=1E-8):
    '''Use the GMRES algorithm to approximate the solution to Ax=b, where ↩
        A is the linear operator defined by `Amul'.

    INPUTS:
    b    - A NumPy array.
    Amul - A function handle. Should describe a linear operator.
    k    - Maximum number of iterations of the GMRES algorithm. Defaults
           to 100.
    tol  - Stop iterating if the residual is less than `tol'. Defaults to
           1E-8.

    RETURN:
    Return (y, res) where `y' is an approximate solution to Ax=b and `res'
    is the residual.

    Examples:
    >>> A = np.array([[1,0,0],[0,2,0],[0,0,3]])
    >>> Amul = lambda x: A.dot(x)
    >>> b = np.array([1, 4, 6])
    >>> gmres(Amul, b)
    (array([ 1.,  2.,  2.]), 1.5083413465299765e-17)
    '''
```

You may make the following assumptions:

1. The input `b` is a real array and the function `Amul()` always outputs real arrays.
2. The vectors found by the Arnoldi iteration will never be zero. In other words, $\mathbf{b}$ will never be in a proper invariant subspace of A.

Hint: Use `scipy.linalg.lstsq()` to solve the least squares problem.

Convergence of GMRES

At the n-th iteration, GMRES computes the best approximate solution $\mathbf{x} \in \mathcal{K}_n$ to $A\mathbf{x} = \mathbf{b}$. If A is full rank, then $\mathcal{K}_m = \mathbb{F}^m$, so the m^{th} iteration will always return an exact answer. However, we say the algorithm converges after n steps if the n^{th} residual is sufficiently small.

The rate of convergence of GMRES depends on the eigenvalues of A.

Problem 2. Finish the following Python function by modifying your solution to Problem 1.

```python
def plot_gmres(b, A, tol=1E-8):
    '''Use the GMRES algorithm to approximate the solution to Ax=b.

    INPUTS:
    b   - A 1-D NumPy array of length m.
    A   - A 2-D NumPy array of shape mxm.
    tol - Stop iterating and create the desired plots if the residual is
          less than `tol'. Defaults to 1E-8.

    OUTPUT:
    Follow the GMRES algorithm until the residual is less than tol, for a
    maximum of m iterations. Then create the two following plots (subplots
    of a single figure):

    1. Plot the eigenvalues of A in the complex plane

    2. Plot the convergence of the GMRES algorithm by plotting the
    iteration number on the x-axis and the residual on the y-axis.
    Use a log scale on the y-axis.
    '''
```

Use this function to investigate the convergence of GMRES as follows. Define an $m \times m$ matrix

$$A_n = nI + P,$$

where I is the $m \times m$ identity matrix and P is a $m \times m$ matrix of numbers from a random normal distribution with mean 0 and standard deviation $1/(2\sqrt{m})$. Call `plot_gmres` on A_n for $n = -4, -2, 0, 2, 4$. Use $m = 200$ and let `b` be an array of ones. What do you conjecture about the convergence of the GMRES algorithm?

Hints:

1. Create a plot with a log scale on the y-axis with the function `plt.semilogy()`.

2. Create a matrix with entries from a random normal distribution with `np.random.rand()`.
3. Output for $n = 2$, $m = 200$ is in Figure 1.1 below.

Ideas for this problem were taken from Example 35.1 on p.271 of [TB97].

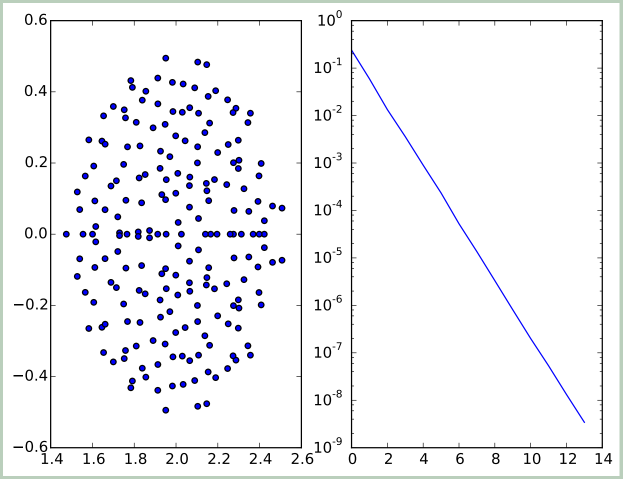

Figure 1.1: The left plot is the eigenvalues of the matrix A_2, which is defined in Problem 2. The right plot is the convergence of the GMRES algorithm on A_2 with starting vector $\mathbf{b} = (1, 1, \ldots, 1)$. This figure is one possible output of the function `plot_gmres()`.

Improving GMRES

There are many ways to make the GMRES algorithm more robust and efficient.

Breakdowns in GMRES

One of the selling points of GMRES is that it can't break down unless it has reached an exact solution.

The only way GMRES could break down is if a vector found by the Arnoldi iteration is 0. That is, suppose, we have already computed

$$\mathcal{K}_n(A, \mathbf{b}) = \operatorname{span}\{\mathbf{b}, A\mathbf{b}, \ldots, A^{n-1}\mathbf{b}\} = \operatorname{span}\{\mathbf{q}_1, \ldots, \mathbf{q}_n\}.$$

We next compute $A^n\mathbf{b}$ and orthogonalize it against $\mathcal{K}_n(A, \mathbf{b})$, yielding $\mathbf{q}_{n+1}$. But if $A^n\mathbf{b} \in \mathcal{K}_n(A, \mathbf{b})$ then $\mathbf{q}_{n+1}$ will be 0, and our algorithm will break when we try to normalize $\mathbf{q}_{n+1}$.

In this situation, the least squares solution to (1.2) is an *exact* solution to $A\mathbf{x} = \mathbf{b}$. In other words, $\mathbf{b}$ is in the span$\{\mathbf{q}_1, \ldots, \mathbf{q}_n\}$.

Problem 3. Update your solution to Problem 1 to deal with the scenario described above. Hint: Use something similar to lines 11-12 of the Arnoldi algorithm in Lab **??**.

GMRES with restarts

The first few iterations of GMRES have low spatial and temporal complexity. However, as k increases, the k^{th} iteration of GMRES becomes more expensive in both time and memory. In fact, computing the k^{th} iteration of GMRES for very large k can be prohibitively complex.

This issue is addressed by using GMRES(k), or GMRES with restarts. When k becomes large, this algorithm restarts GMRES but with an improved initial guess. GMRES with restarts is outlined in Algorithm 1.2.

Algorithm 1.2 The GMRES(k) algorithm. This algorithm performs GMRES on a vector $\mathbf{b}$ and matrix A. It iterates k times before restarting. It terminates after *restarts* restarts or when the residual is less than *tol*, returning an approximate solution to $A\mathbf{x} = \mathbf{b}$ and the error in this approximation.

```
1:  procedure GMRES(K)($A, \mathbf{b}, k, tol, restarts$)
2:      $r \leftarrow 0$                                         ▷ Restart counter
3:      $Q \leftarrow \text{size}(b)k + 1$                         ▷ Initialize
4:      $H \leftarrow \text{zeros}(k+1, k)$
5:      while $r \leq restarts$ do
6:          Perform the GMRES algorithm, obtaining a least squares solution $\mathbf{y}$
7:          If the desired tolerance was reached, return. Otherwise, continue.
8:          $b \leftarrow Q[:, k+1]\mathbf{y}$
9:          $r \leftarrow r + 1$
10:     return $Q[:, k+1]\mathbf{y}$, $res$       ▷ Return the approximate solution and the residual
```

The algorithm GMRES(k) will always have manageable spatial and temporal complexity, but it is less reliable than GMRES. If the true solution $\mathbf{x}$ to $A\mathbf{x} = \mathbf{b}$ is nearly orthogonal to the Krylov subspaces $\mathcal{K}_n(A, \mathbf{b})$ for $n \leq k$, then GMRES(k) could converge very slowly or not at all.

Problem 4. Implement Algorithm 1.2 with the following function.

```
def gmres_k(b, Amul, k=100, tol=1E-8, restarts=50):
    '''Use the GMRES(k) algorithm to approximate the solution to Ax=b, ↩
        where A is the linear operator defined by `Amul'.

    INPUTS:
    b        - A NumPy array.
    Amul     - A function handle. Should describe a linear operator.
    k        - Maximum number of iterations of the GMRES algorithm before
```

```
                 restarting. Defaults to 100.
    tol        - Stop iterating if the residual is less than `tol'. Defaults
                 to 1E-8.
    restarts - Maximum number of restarts. Defaults to 50.

    RETURN:
    Return (y, res) where `y' is an approximate solution to Ax=b and `res'
    is the residual.
    '''
```

Compare the speed of `gmres()` from Problem 1 and `gmres_k()` on the matrices in Problem 2.

GMRES in SciPy

The GMRES algorithm is implemented in SciPy as the function `scipy.sparse.linalg.gmres()`. Here we use this function to solve $A\mathbf{x} = \mathbf{b}$ where A is a random 300×300 matrix and $\mathbf{b}$ is a random vector.

```
>>> import numpy as np
>>> from scipy import sparse as spar
>>> from scipy import linalg as la
>>>
>>> A = np.random.rand(300, 300)
>>> b = np.random(300)
>>> x, info = spar.linalg.gmres(A, b)
>>> info
3000
```

The function outputs two objects: the approximate solution `x` and a constant `info` telling if the function converged. If `info=0` then convergence occured; if `info` is positive then it equals the number of iterations performed. In this case, the function performed 3000 iterations of GMRES before returning the approximate solution `x`. We can check how close the solution is.

```
>>> la.norm(A.dot(x)-b)
4.744196381683801
```

We can get a better approximation using GMRES with restarts.

```
>>> # Restart after 1000 iterations
>>> x, info = spar.linalg.gmres(A, b, restart=1000)
>>> info
0
>>> la.norm(A.dot(x)-b)
1.0280404494143551e-12
```

This time, the returned approximation `x` is about as close to a true solution as we could hope for.