17 GMRES

Lab Objective: *The Generalized Minimal Residuals (GMRES) algorithm is an iterative Krylov subspace method for efficiently solving large linear systems. In this lab we implement the basic GMRES algorithm, then make an improvement by using restarts. We then discuss the convergence of the algorithm and its relationship with the eigenvalues of a linear system. Finally, we introduce SciPy's version of GMRES.*

The GMRES Algorithm

GMRES is an iterative method that uses Krylov subspaces to reduce a high-dimensional problem to a sequence of smaller dimensional problems. Let A be an invertible $m \times m$ matrix and let $\mathbf{b}$ be a vector of length m. Let $\mathcal{K}_n(A, \mathbf{b})$ be the order-n Krylov subspace generated by A and $\mathbf{b}$. Instead of solving the system $A\mathbf{x} = \mathbf{b}$ directly, GMRES uses least squares to find $\mathbf{x}_n \in \mathcal{K}_n$ that minimizes the residual $r_n = \|\mathbf{b} - A\mathbf{x}_n\|_2$. The algorithm terminates when this residual is smaller than some predetermined value. In many situations, this happens when n is much smaller than m.

The GMRES algorithm uses the Arnoldi iteration for numerical stability. The Arnoldi iteration produces H_n, an $(n+1) \times n$ upper Hessenberg matrix, and Q_n, a matrix whose columns make up an orthonormal basis of $\mathcal{K}_n(A, \mathbf{b})$, such that $AQ_n = Q_{n+1}H_n$. The GMRES algorithm finds the vector $\mathbf{x}_n$ which minimizes the norm $\|\mathbf{b} - A\mathbf{x}_n\|_2$, where $\mathbf{x}_n = Q_n\mathbf{y}_n + \mathbf{x}_0$ for some $\mathbf{y}_n \in \mathbb{R}^n$. Since the columns of Q_n are orthonormal, the residual can be equivalently computed as

$$\|\mathbf{b} - A\mathbf{x}_n\|_2 = \|Q_{n+1}(\beta\mathbf{e}_1 - H_n\mathbf{y}_n)\|_2 = \|H_n\mathbf{y}_n - \beta\mathbf{e}_1\|_2. \tag{17.1}$$

Here $\mathbf{e}_1$ is the vector $[1, 0, \ldots, 0]^\mathsf{T}$ of length $n+1$ and $\beta = \|\mathbf{b} - A\mathbf{x}_0\|_2$, where $\mathbf{x}_0$ is an initial guess of the solution. Thus, to minimize $\|\mathbf{b} - A\mathbf{x}_n\|_2$, the right side of (17.1) can be minimized, and $\mathbf{x}_n$ can be computed as $\mathbf{x}_n = Q_n\mathbf{y}_n + \mathbf{x}_0$.

Algorithm 17.1 The GMRES algorithm. This algorithm operates on a vector $\mathbf{b}$ and a linear operator 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($A$, $\mathbf{b}$, $\mathbf{x}_0$, $k$, tol)
2:     $Q \leftarrow \text{empty}(\text{size}(\mathbf{b}), k+1)$                ▷ Initialization.
3:     $H \leftarrow \text{zeros}(k+1, k)$
4:     $r_0 \leftarrow \mathbf{b} - A(\mathbf{x}_0)$
5:     $Q_{:,0} = r_0 / \|r_0\|_2$
6:     for $j = 0 \ldots k-1$ do                         ▷ Perform the Arnoldi iteration.
7:         $Q_{:,j+1} \leftarrow A(Q_{:,j})$
8:         for $i = 0 \ldots j$ do
9:             $H_{i,j} \leftarrow Q_{:,i}^{\mathsf{T}} Q_{:,j+1}$
10:            $Q_{:,j+1} \leftarrow Q_{:,j+1} - H_{i,j} Q_{:,i}$
11:        $H_{j+1,j} \leftarrow \|Q_{:,j+1}\|_2$
12:        if $|H_{j+1,j}| >$ tol then                          ▷ Avoid dividing by zero.
13:            $Q_{:,j+1} \leftarrow Q_{:,j+1} / H_{j+1,j}$
14:        $\mathbf{y} \leftarrow$ least squares solution to $\|H_{:j+2,:j+1}\mathbf{x} - \beta e_1\|_2$      ▷ $\beta$ and $\mathbf{e}_1$ as in (17.1).
15:        res $\leftarrow \|H_{:j+2,:j+1}\mathbf{y} - \beta e_1\|_2$
16:        if res $<$ tol then
17:            return $Q_{:,:j+1}\mathbf{y} + \mathbf{x}_0$, res
18:    return $Q_{:,:j+1}\mathbf{y} + \mathbf{x}_0$, res
```

Problem 1. Write a function that accepts a matrix A, a vector $\mathbf{b}$, and an initial guess $\mathbf{x}_0$, a maximum number of iterations k defaulting to 100, and a stopping tolerance `tol` that defaults to 10^{-8}. Use Algorithm 17.1 to approximate the solution to $A\mathbf{x} = \mathbf{b}$ using the GMRES algorithm. Return the approximate solution and the residual at the approximate solution.

You may assume that A and $\mathbf{b}$ only have real entries. Use `scipy.linalg.lstsq()` to solve the least squares problem. Be sure to read the documentation so that you understand what the function returns.

Compare your function to the following code.

```
>>> A = np.array([[1,0,0],[0,2,0],[0,0,3]])
>>> b = np.array([1, 4, 6])
>>> x0 = np.zeros(b.size)
>>> gmres(A, b, x0, k=100, tol=1e-8)
(array([ 1.,  2.,  2.]), 7.174555448775421e-16)
```

Convergence of GMRES

One of the most important characteristics of GMRES is that it will always arrive at an exact solution (if one exists). At the n-th iteration, GMRES computes the best approximate solution to $A\mathbf{x} = \mathbf{b}$ for $\mathbf{x}_n \in \mathcal{K}_n$. If A is full rank, then $\mathcal{K}_m = \mathbb{F}^m$, so the mth iteration will always return an exact answer. Sometimes, the exact solution $\mathbf{x} \in \mathcal{K}_n$ for some $n < m$, in this case x_n is an exact solution. In either case, the algorithm is convergent after n steps if the nth residual is sufficiently small.

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

Problem 2. Add a keyword argument `plot` defaulting to `False` to your function from Problem 1. If `plot=True`, keep track of the residuals at each step of the algorithm. At the end of the iteration, before returning the approximate solution and its residual error, create a figure with two subplots.

1. Make a scatter plot of the eigenvalues of A on the complex plane.
2. Plot the residuals versus the iteration counts using a log scale on the y-axis (use `ax.semilogy()`).

Problem 3. Use your function from Problem 2 to investigate how the convergence of GMRES relates to the eigenvalues of a matrix as follows. Define an $m \times m$ matrix

$$A_n = nI + P,$$

where I is the identity matrix and P is an $m \times m$ matrix with entries taken from a random normal distribution with mean 0 and standard deviation $1/(2\sqrt{m})$. Call your function from Problem 2 on A_n for $n = -4, -2, 0, 2, 4$. Use $m = 200$, let $\mathbf{b}$ be an array of all ones, and let $\mathbf{x}_0 = \mathbf{0}$.

Use `np.random.normal()` to create the matrix P. When analyzing your results, pay special attention to the clustering of the eigenvalues in relation to the origin. Compare your results with $n = 2$, $m = 200$ to Figure 17.1.

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

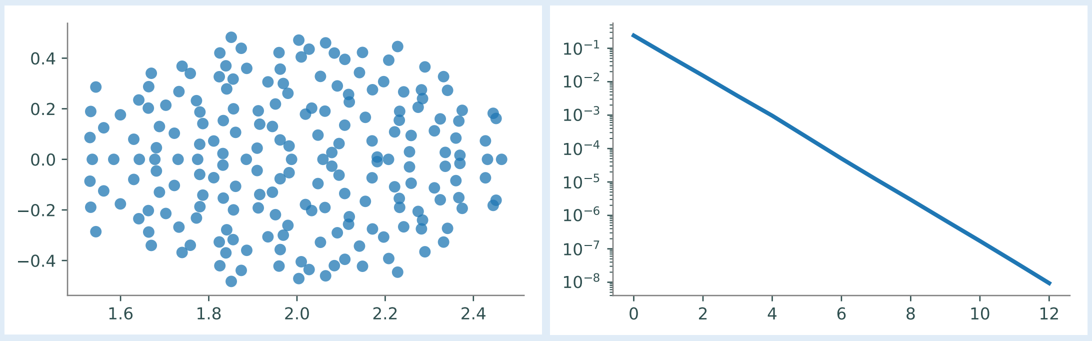

Figure 17.1: On the left, the eigenvalues of the matrix A_2 defined in Problem 3. On the right, the rapid convergence of the GMRES algorithm on A_2 with starting vector $\mathbf{b} = (1, 1, \ldots, 1)$.

GMRES with Restarts

The first few iterations of GMRES have low spatial and temporal complexity. However, as k increases, the kth iteration of GMRES becomes more expensive temporally and spatially. In fact, computing the kth 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 with an improved initial guess. The new initial guess is taken to be the vector that was found upon termination of the last GMRES iteration run. 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. Write a function that implements GMRES with restarts as follows.

1. Perform the GMRES algorithm for a maximum of k iterations.
2. If the desired tolerance was reached, terminate the algorithm. If not, repeat step 1 using x_k from the previous GMRES algorithm as a new initial guess x_0.
3. Repeat step 2 until the desired tolerance has been obtained or until a given maximum number of restarts has been reached.

Your function should accept all of the same inputs as the function you wrote in Problem 1 with the exception of k, which will now denote the number of iterations before restart (defaults to 5), and an additional parameter `restarts` which denotes the maximum number of restarts before termination (defaults to 50).

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
>>> from scipy.sparse import linalg as spla

>>> A = np.random.rand(300, 300)
>>> b = np.random(300)
>>> x, info = spla.gmres(A, b)
>>> print(info)
3000
```

The function outputs two objects: the approximate solution `x` and an integer `info` which gives information about the convergence of the algorithm. If `info=0` then convergence occured; if `info` is positive then it equals the number of iterations performed. In the previous case, the function performed 3000 iterations of GMRES before returning the approximate solution `x`. The following code verifies how close the computed value was to the exact solution.

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

A better approximation can be obtained using GMRES with restarts.

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

This time, the returned approximation **x** is about as close to a true solution as can be expected.

Problem 5. Plot the runtimes of your implementations of GMRES from Problems 1 and 4 and `scipy.sparse.linalg.gmres()` use the default tolerance and `restart=1000` with different matrices. Use the $m \times m$ matrix P with $m = 25, 50, \ldots 200$ and with entries taken from a random normal distribution with mean 0 and standard deviation $1/(2\sqrt{m})$. Use a vector of ones for $\mathbf{b}$ and a vector of zeros for $\mathbf{x}_0$. Use a single figure for all plots, plot the runtime on the y-axis and m on the x-axis.

Bibliography

[TB97] Lloyd N. Trefethen and David Bau, III. *Numerical linear algebra*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1997. [3]