GMRES and Conjugate Gradient

MTH573: Scientific Computing, Monsoon 2022

Ananya Lohani, **2019018**

Table of Contents

- Introduction
- Problem Statement
- Krylov Subspaces
- GMRES (Generalised Minimum Residual)
    - Use of Krylov Subspaces
    - Arnoldi's Method
        - Basic Algorithm
        - Practical Implementation
    - Basic GMRES Algorithm
    - Pseudocode
    - Implementation
    - Error Analysis
- Conjugate Gradient
    - Basic Algorithm
    - Pseudocode
    - Implementation
    - Error Analysis
- References

Introduction

The GMRES (Generalized Minimal Residual) algorithm is an iterative method for solving linear systems of equations. It improves an initial approximation to the solution by minimizing the residual error at each iteration. GMRES is efficient and robust, especially for large and sparse coefficient matrices, and can be used in a variety of applications.

The Conjugate Gradient (CG) algorithm is an iterative method for solving systems of linear equations. It is a variant of the gradient descent method, which is used to minimize the residual error of a linear system. CG is an efficient and reliable algorithm, especially when the coefficient matrix is symmetric and positive definite.

Both the algorithms use the method of linear least squares to solve the linear system of equations by minimizing the residual on each iteration.

Problem Statement

The problem of solving a linear system of equations is as follows:

$$\text{Given a matrix } A \epsilon \mathbb{R}^{m \times n} \text{ and a vector } b \epsilon \mathbb{R}^{m}, \text{ find a vector } x \epsilon \mathbb{R}^{n} \text{ such that:}$$
$$Ax = b$$

Both, the GMRES and Conjugate Gradient algorithms operate on square matrices, i.e. matrices that belong to $\mathbb{R}^{n \times n}$.

Alternatively, the problem may be defined as finding a vector $\boldsymbol{x}$ such that the norm of the residual vector $\boldsymbol{r}$ is exactly, or approximately in the numerical scope, zero.

$$\text{Given a matrix } A \epsilon \mathbb{R}^{n \times n} \text{ and a vector } b \epsilon \mathbb{R}^{n}, \text{ find a vector } x \epsilon \mathbb{R}^{n} \text{ such that :}$$
$$r := b - Ax$$
$$x = \arg\min_{x}(\| r \|_2) = \arg\min_{x}(\| b - Ax \|_2)$$
$$\text{i.e. the 2-norm of the residual vector } r \epsilon \mathbb{R}^{n} \text{ is minimized, or zero.}$$

Krylov Subspaces

Krylov subspaces are a particular type of vector space that is used in numerical linear algebra to solve linear systems of equations, compute matrix functions, and perform eigenvalue decomposition. They are named after the Russian mathematician Aleksandr Mikhailovich Krylov (1863-1945), who introduced the concept in the early 20th century.

Given a matrix $\boldsymbol{A}$ and a vector $\boldsymbol{v}$, the Krylov subspace of order m is defined as the linear span of the following sequence of vectors:

$$\kappa_m\left(A,\, v\right) \,:=\, \operatorname{span}\left\{\, v,\, Av,\, A^2v,\, \ldots,\, A^{m-1}v \right\}$$

where $\boldsymbol{A^i}$ denotes the i-th power of the matrix $\boldsymbol{A}$. The Krylov subspace is a subspace of the vector space in which the matrix $\boldsymbol{A}$ and the vector $\boldsymbol{v}$ are defined.

Krylov subspaces are often used in iterative methods for solving linear systems of equations, such as the Conjugate Gradient method and the GMRES algorithm. They have the advantage of requiring only matrix-vector multiplications, which can be computed efficiently for large matrices, and can be used to approximate the solution to the linear system with a specified accuracy.

GMRES (Generalised Minimum Residual)

The GMRES algorithm utilizes properties of Krylov subspaces and Arnoldi's method to minimize the residual at each iteration to find a solution to the linear system of equations.

Use of Krylov Subspaces

One of the key steps in the GMRES method is the construction of the Krylov subspace, which is used to approximate the solution to the linear system. By restricting the search space to the Krylov subspace, the GMRES method can compute an approximate solution more efficiently, as it avoids the need to compute matrix-matrix multiplications.

In addition, the Krylov subspace has the property that the residual vector (the difference between the exact solution and the approximate solution) belongs to the Krylov subspace. This allows the GMRES method to construct an iterative solution that is guaranteed to converge to the exact solution as the dimension of the Krylov subspace increases.

Overall, the use of Krylov subspaces in the GMRES method allows for an efficient and robust algorithm for solving linear systems of equations.

Arnoldi's Method

The basic idea behind Arnoldi's method is to iteratively construct a sequence of orthonormal vectors, called the Arnoldi basis, that span a Krylov subspace that approximates the eigenspace

of the matrix. The method starts with an initial vector $\boldsymbol{v}$ and applies the matrix $\boldsymbol{A}$ to it repeatedly to generate a sequence of vectors:

$$v,\ Av,\ A^2v,\ A^3v,\ \ldots.$$

The vectors in this sequence are then orthonormalized using the Gram-Schmidt process, and the resulting orthonormal vectors form the Arnoldi basis of the Krylov subspace.

Arnoldi's method can be used to compute a few of the dominant eigenvalues and eigenvectors of the matrix $\boldsymbol{A}$, or to compute all the eigenvalues and eigenvectors in a specified region of the complex plane. It is particularly useful for large, sparse matrices, as it avoids the need to compute matrix-matrix multiplications and can be implemented efficiently using sparse matrix techniques.

Basic Algorithm

ALGORITHM 1. Arnoldi's Method

1. *Choose a vector* v_1, *such that* $\|v_1\|_2 = 1$
2. *For* $j = 1, 2, \ldots, m$ *Do:*
3. *Compute* $h_{ij} = (Av_j, v_i)$ *for* $i = 1, 2, \ldots, j$
4. *Compute* $w_j := Av_j - \sum_{i=1}^{j} h_{ij} v_i$
5. $h_{j+1,j} = \|w_j\|_2$
6. *If* $h_{j+1,j} = 0$ *then Stop*
7. $v_{j+1} = w_j / h_{j+1,j}$
8. *EndDo*

At each step, the algorithm multiplies the previous Arnoldi vector $\boldsymbol{v_j}$ by $\boldsymbol{A}$ and then orthonormalizes the resulting vector $\boldsymbol{w_j}$ against all previous $\boldsymbol{v_i}$'s by a standard Gram-Schmidt procedure. It will stop if the vector $\boldsymbol{w_j}$ computed in line 4 vanishes.

Practical Implementation

In the previous description of the Arnoldi process, exact arithmetic was assumed, mainly for simplicity. In practical implementations, using the Modified Gram-Schmidt method is more accurate in case of round-off.

ALGORITHM 2. Arnoldi-Modified Gram-Schmidt

1. *Choose a vector* v_1 *of norm 1*
2. *For* $j = 1, 2, \ldots, m$ *Do:*
3. *Compute* $w_j := Av_j$
4. *For* $i = 1, \ldots, j$ *Do:*
5. $h_{ij} = (w_j, v_i)$
6. $w_j := w_j - h_{ij}v_i$
7. *EndDo*
8. $h_{j+1,j} = \|w_j\|_2$. *If* $h_{j+1,j} = 0$ *Stop*
9. $v_{j+1} = w_j / h_{j+1,j}$
10. *EndDo*

Basic GMRES Algorithm

Let K_m be the order-m Krylov subspace generated by A and b. Instead of solving the system $Ax = b$ directly, GMRES uses least squares to find $x_m \in K_m$ that minimizes the residual $r_m = || b - Ax_m ||_2$. The algorithm terminates when this residual is smaller than some tolerance or threshold.

The GMRES algorithm uses the Arnoldi method for numerical stability. The Arnoldi iteration produces H_m, an $(m + 1) \times m$ upper Hessenberg matrix, and Q_m, a matrix whose columns make up an orthonormal basis of $K_m(A, b)$, such that $AQ_m = Q_{m+1}H_m$. The GMRES algorithm finds the vector x_m which minimizes the norm $|| b - Ax_m ||_2$, where $x_m = Q_m y_m + x_0$ for some $y_m \in \mathbb{R}^n$. Since the columns of Q_m are orthonormal, the residual can be equivalently computed as:

$$\|b - Ax_m\|_2 = \|Q_{m+1}(\beta e_1 - H_m y_m)\|_2 = \|H_m y_m - \beta e_1\|_2 \qquad \text{eq(1.1)}$$

Here e_1 is the vector $[1, 0, \ldots, 0]^T$ of length $(m + 1)$ and $\beta = || b - Ax_0 ||_2$, where x_0 is an initial guess of the solution. Thus, to minimize $|| b - Ax_m ||_2$, the right side of eq(1.1) can be minimized, and x_m can be computed as $x_m = Q_m y_m + x_0$.

Pseudocode

ALGORITHM 3: The GMRES Algorithm

procedure GMRES(A, $\mathbf{b}$, $\mathbf{x}_0$, k, `tol`)
  $Q \leftarrow \text{empty}(\text{size}(\mathbf{b}), k+1)$
  $H \leftarrow \text{zeros}(k+1, k)$
  $r_0 \leftarrow \mathbf{b} - A(\mathbf{x}_0)$
  $Q_{:,0} = r_0 / \|r_0\|_2$
  for $j = 0 \ldots k-1$ **do**
    $Q_{:,j+1} \leftarrow A(Q_{:,j})$
    for $i = 0 \ldots j$ **do**
      $H_{i,j} \leftarrow Q_{:,i}^{\mathsf{T}} Q_{:,j+1}$
      $Q_{:,j+1} \leftarrow Q_{:,j+1} - H_{i,j} Q_{:,i}$
    $H_{j+1,j} \leftarrow \|Q_{:,j+1}\|_2$
    if $|H_{j+1,j}| >$ `tol` **then**
      $Q_{:,j+1} \leftarrow Q_{:,j+1} / H_{j+1,j}$
    $\mathbf{y} \leftarrow$ least squares solution to $\|H_{:j+2,:j+1}\mathbf{x} - \beta e_1\|_2$
    `res` $\leftarrow \|H_{:j+2,:j+1}\mathbf{y} - \beta e_1\|_2$
    if `res` $<$ `tol` **then**
      return $Q_{:,:j+1}\mathbf{y} + \mathbf{x}_0$, `res`
  return $Q_{:,:j+1}\mathbf{y} + \mathbf{x}_0$, `res`

Implementation

```python
def gmres(A, b, x0, max_iter=1000, tol=1e-10):
    k = max_iter

    # Initialize matrices Q and H
    Q = np.empty((len(b), k+1))
    H = np.zeros((k+1, k))

    # Calculate the initial residual vector r0 = b - A(x0)
    r0 = b - A.dot(x0)
    beta = np.linalg.norm(r0)
    Q[:, 0] = r0 / beta

    # Initialize the first standard basis vector e1 of length k+1
    e1 = np.zeros(k+1)
    e1[0] = 1

    # Perform the Arnoldi iteration
    for j in range(k):
        Q[:, j+1] = A.dot(Q[:, j])

        # Update H
```

```
        for i in range(j+1):
            H[i, j] = Q[:, i].T.dot(Q[:, j+1])
            Q[:, j+1] -= H[i, j] * Q[:, i]

        H[j+1, j] = np.linalg.norm(Q[:, j+1])

        # Avoid dividing by zero
        if H[j+1, j] > tol:
            Q[:, j+1] /= (H[j+1, j] + 1e-13)

    # Solve the least squares problem to find y
    y = np.linalg.lstsq(H[:j+2, :j+1], beta*e1, rcond=None)[0]
    x = Q[:, :j+1].dot(y) + x0
    res = np.linalg.norm(A @ x - b)

    if res < tol:
        return x, res

    return x, res
```

Error Analysis

I implemented the above-mentioned pseudocode in Python 3.9 and computed the relative error of the solution x with respect to Scipy's implementation of GMRES for randomly generated square matrices with sizes `n = 10, 50, 100, 250, 500, 1000`. I chose x_0 = `[0 0 0 ... 0]`^{T} for fairly consistent results. The results are as follows:

n	relative error in x
10	1.58246e-12
50	2.74687e-13
100	5.21597e-13
250	1.44335e-12
500	1.37188e-12
1000	7.80949e-13

The errors are of the order 10^{-13} to 10^{-12}, which is fairly close to Scipy's implementation.

Conjugate Gradient

The conjugate gradient (CG) algorithm is an iterative method for solving a system of linear equations, specifically a system of the form $Ax = b$, where A is a symmetric, positive definite matrix. It is an efficient algorithm for solving large, sparse systems of linear equations that arise in many scientific and engineering applications.

Basic Algorithm

The CG algorithm works by iteratively improving an approximation to the solution x of the system $Ax = b$. At each iteration, it calculates the residual r, which is the difference between the right-hand side of the equation b and the current approximation to the solution Ax. It then computes the search direction p, which is a direction that reduces the residual in the least-squares sense. The algorithm then takes a step in the search direction and updates the current approximation to the solution. This process is repeated until the residual is small enough, at which point the algorithm terminates.

The key property that makes the CG algorithm efficient is that the search direction p at each iteration is chosen to be conjugate to the search directions at all previous iterations. This ensures that the algorithm converges quickly and requires fewer iterations to reach a solution.

Pseudocode

ALGORITHM 4: The Conjugate Gradient Algorithm

1. *Compute* $r_0 := b - Ax_0, p_0 := r_0$.
2. *For* $j = 0, 1, \ldots,$ *until convergence Do:*
3. $\quad \alpha_j := (r_j, r_j)/(Ap_j, p_j)$
4. $\quad x_{j+1} := x_j + \alpha_j p_j$
5. $\quad r_{j+1} := r_j - \alpha_j A p_j$
6. $\quad \beta_j := (r_{j+1}, r_{j+1})/(r_j, r_j)$
7. $\quad p_{j+1} := r_{j+1} + \beta_j p_j$
8. *EndDo*

Implementation

```
def conjugate_gradient(A, b, x, tol=1e-10, max_iter=1000):
    r = b - A @ x
    p = r
    prev_norm = r.T @ r
```

```
    for _ in range(max_iter):
        Ap = A @ p
        alpha = prev_norm / (p.T @ Ap)
        x += alpha * p
        r -= alpha * Ap
        norm = r.T @ r
        if np.sqrt(norm) < tol:
            break
        p = r + (norm / prev_norm) * p
        prev_norm = norm

    return x
```

Error Analysis

I implemented the above-mentioned pseudocode in Python 3.9 and computed the relative error of the solution x with respect to Scipy's implementation of GMRES for randomly generated square matrices with sizes `n = 10, 50, 100, 250, 500, 1000`. I chose x_0 = `[1 1 1 ... 1]`^{T} for fairly consistent results. The results are as follows:

n	relative error in x
10	4.62623e-11
50	4.84575e-11
100	6.23181e-11
250	2.07756e-11
500	3.69937e-11
1000	2.17015e-10

The errors are of the order 10^{-11} to 10^{-10}, which is fairly close to Scipy's implementation.

References

1. Iterative Methods for Sparse Linear Systems [2nd Edition] by Yousef Saad.
2. GMRES by ACME [link]
3. GMRES on Wikipedia [link]
4. Conjugate Gradient on Wikipedia [link]
5. Several StackOverflow answers