MATH 5330: Computational Methods of Linear Algebra

Lecture Note 11: GMRES Revisited

Xianyi Zeng
Department of Mathematical Sciences, UTEP

1 Introduction

In the GMRES algorithm we described before, there are two parts at loose. Let us recap a little bit over here. In the process of searching for the solution of $A\boldsymbol{x}=\boldsymbol{b}$ where $A\in\mathbb{R}^{n\times n}$ is non-singular, we start with a guess $\boldsymbol{x}_0$ and compute its residual $\boldsymbol{r}_0=\boldsymbol{b}-A\boldsymbol{x}_0$. Let $\boldsymbol{r}_0\neq 0$, then the Krylov subspace method searches for the m-th iteration a solution in the affine space:

$$\boldsymbol{x}_m\in\boldsymbol{x}_0+\mathcal{K}_m\,,\quad \mathcal{K}_m=\mathrm{span}(\boldsymbol{r}_0,A\boldsymbol{r}_0,\cdots,A^{m-1}\boldsymbol{r}_0)\,,$$

so that the residual $\boldsymbol{r}_m=\boldsymbol{b}-A\boldsymbol{x}_m$ has the minimal L^2-norm among all vectors in this affine space.

In the first step, we find an orthonormal basis of $\mathcal{K}_m$ so long as the algorithm proceeds (i.e., $\dim\mathcal{K}_m=m$):

$$\mathcal{K}_m=\mathrm{span}(\boldsymbol{v}_1,\boldsymbol{v}_2,\cdots,\boldsymbol{v}_m)\,,$$

and see that there exists an $m\times(m+1)$ Hessenberg $\overline{H}_m$ such that:

$$AV_m=V_{m+1}\overline{H}_m=V_m\overline{H}_m+h_{m+1,m}\boldsymbol{v}_{m+1}\boldsymbol{e}_m^t\,,$$

where H_m is the $m\times m$ Hessenberg matrix that is composed of the first m rows of $\overline{H}_m$.

The first loose part comes from the Arnoldi's process that we used to find this orthonormal basis; this algorithm is repeated here in Algorithm 1.1. Since Algorithm 1.1 is based on the Gram-Schmidt process, it is numerically unstable – round-off errors may lead to non-orthogonal basis. The first task of this lecture is to fix this issue by using the Householder transformation instead to find the orthogonal basis.

Algorithm 1.1 The Arnoldi's Algorithm

Compute $\boldsymbol{r}_0=\boldsymbol{b}-A\boldsymbol{x}_0$ and $\boldsymbol{v}_1=\boldsymbol{r}_0/||\boldsymbol{r}_0||$
for $j=1,\cdots,m$ **do**
  $\boldsymbol{w}_j=A\boldsymbol{v}_j$
  for $i=1,\cdots,j$ **do**
    $h_{ij}=\boldsymbol{w}_j\cdot\boldsymbol{v}_i$
    $\boldsymbol{w}_j\leftarrow\boldsymbol{w}_j-h_{ij}\boldsymbol{v}_i$
  end for
  $h_{j+1,j}=||\boldsymbol{w}_j||$
  if $h_{j+1,j}=0$ **then**
    Stop;
  end if
  $\boldsymbol{v}_{j+1}=\boldsymbol{w}_j/h_{j+1,j}$
end for

The second loose part come from the least-squares solver that is used a black box algorithm before in finding $\boldsymbol{x}_m$ in $\boldsymbol{x}_0 + \mathcal{K}_m$. That is, suppose the basis V_{m+1} and the Hessenberg matrix $\overline{H}_m$ are constructed, we solve for the least-squares problem:

$$\boldsymbol{y}_m = \arg\min_{\boldsymbol{y}\in\mathbb{R}^m} \left|\left|\beta \boldsymbol{e}_1 - \overline{H}_m \boldsymbol{y}\right|\right|, \tag{1.1}$$

and compute the next iterate:

$$\boldsymbol{x}_m = \boldsymbol{x}_0 + V_m \boldsymbol{y}_m .$$

Here $\beta = ||\boldsymbol{r}_0||$ and our second target is to use Givens rotations to derive a solver for (1.1).

2 Householder Arnoldi Algorithm

The result of the Arnoldi's algorithm is:

$$AV_m = V_{m+1}\overline{H}_m ,$$

and we notice the first column of V_{m+1} is $\boldsymbol{v}_1 = \beta^{-1}\boldsymbol{r}_0$, hence we have:

$$[\boldsymbol{r}_0 \ AV_m] = V_{m+1}R_m , \tag{2.1}$$

where:

$$R_m = \begin{bmatrix} \beta & \\ 0 & \\ \vdots & \overline{H}_m \\ 0 & \end{bmatrix} = \begin{bmatrix} \beta & h_{11} & \cdots & h_{1m} \\ 0 & h_{21} & \cdots & h_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & h_{m+1,m} \end{bmatrix} .$$

Note that the column space of $[\boldsymbol{r}_0 \ AV_m]$ is exactly $\mathcal{K}_{m+1}$ and (2.1) is its (reduced) QR decomposition. If we were going to carry out the process until the end by using a sequence of Householder matrices P_1, $\cdots$, P_n[1] to reduce $[\boldsymbol{r}_0 \ AV_{n-1}]$ to upper-triangular form:

$$P_n P_{n-1} \cdots P_1 [\boldsymbol{r}_0 \ AV_{n-1}] = R_{n-1} ,$$

we immediately see that:

$$V_n = P_1 P_2 \cdots P_n ,$$

which provides us a way to compute the vectors $\boldsymbol{v}_i$ gradually. For example, we'll see that P_j will keep the first $j-1$ rows unchanged, hence $\boldsymbol{v}_1$ is exactly $P_1\boldsymbol{e}_1$ and $\boldsymbol{v}_2 = P_1P_2\boldsymbol{e}_2$, etc.

The Householder version of the Arnoldi's algorithm builds on this process, and it is nothing but applying the Householder transformations to compute the QR decomposition of $[\boldsymbol{r}_0 \ AV_m]$. This is provided in Algorithm 2.1.

[1] We use P instead to avoid confusion with the Hessenberg matrices H.

Algorithm 2.1 Householder Arnoldi

1: Compute $\boldsymbol{z}_1 = \boldsymbol{r}_0$
2: **for** $j = 1,\cdots,m+1$ **do**
3: Compute the Householder unit vector $\boldsymbol{w}_j$ such that
4: The first $j-1$ components of $\boldsymbol{w}_j$ are zero
5: $(I - 2\boldsymbol{w}_j\boldsymbol{w}_j^t)\boldsymbol{z}_j$ is a multiple of $\boldsymbol{e}_j$
6: $P_j = I - 2\boldsymbol{w}_j\boldsymbol{w}_j^t$
7: $\boldsymbol{h}_{j-1} = P_j\boldsymbol{z}_j$
8: $\boldsymbol{v}_j = P_1P_2\cdots P_j\boldsymbol{e}_j$
9: **if** $j \le m$ **then**
10: $\boldsymbol{z}_{j+1} = P_jP_{j-1}\cdots P_1A\boldsymbol{v}_j$
11: **end if**
12: **end for**

It is not difficult to see that lines 3–5 is exactly the same as the Householder orthogonalization we had in the previous lecture. To understand this algorithm, let us look at the first loop in which the Householder matrix P_1 is computed to transform $\boldsymbol{z}_1 = \boldsymbol{r}_0$ into $\boldsymbol{h}_0 = \beta\boldsymbol{e}_1 \in \mathbb{R}^n$:

$$P_1\boldsymbol{r}_0 = \beta\boldsymbol{e}_1 = \boldsymbol{h}_0\,,$$

and following the *a priori* knowledge of the relations between $\boldsymbol{v}$ vectors and P matrices, we can already compute:

$$\boldsymbol{v}_1 = P_1\boldsymbol{e}_1\,,$$

as stated in line 8.

Now we move on to the second loop and compute P_2, such that it essentially has the form:

$$P_2 = \begin{bmatrix} 1 & 0 \\ 0 & I_{n-1} - 2\tilde{\boldsymbol{w}}_2\tilde{\boldsymbol{w}}_2^t \end{bmatrix},$$

where $\tilde{\boldsymbol{w}}_2$ is the last $n-1$ components (and also the non-zero components) of $\boldsymbol{w}_2$, so that

$$P_2P_1\boldsymbol{r}_0 = P_2(\beta\boldsymbol{e}_1) = \beta\boldsymbol{e}_1\,,$$

and

$$P_2P_1A\boldsymbol{v}_1 = P_2\boldsymbol{z}_2 = \boldsymbol{h}_1 = \begin{bmatrix} h_{11} \\ h_{21} \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

Similarly, the next orthonormal basis is constructed as $\boldsymbol{v}_2 = P_1P_2\boldsymbol{e}_2$. As a safety check, we have:

$$\boldsymbol{v}_1 \cdot \boldsymbol{v}_2 = (\boldsymbol{e}_1^tP_1)(P_1P_2\boldsymbol{e}_2) = (P_2\boldsymbol{e}_1P_2)^t\boldsymbol{e}_2 = \boldsymbol{e}_1^t\boldsymbol{e}_2 = 0\,.$$

This process can be continued until we find the orthonormal basis of $\mathcal{K}_{m+1}$. For example, we can check that $\boldsymbol{v}_m$ is orthogonal to $\boldsymbol{v}_j$ for $j < m$:

$$\boldsymbol{v}_j \cdot \boldsymbol{v}_m = (P_1P_2\cdots P_j\boldsymbol{e}_j)^t(P_1P_2\cdots P_m\boldsymbol{e}_m) = \boldsymbol{e}_j^t(P_{j+1}\cdots P_m\boldsymbol{e}_m) = (P_m\cdots P_{j+1}\boldsymbol{e}_j)^t\boldsymbol{e}_m = \boldsymbol{e}_j^t\boldsymbol{e}_m = 0\,.$$

3 Least-Squares with Hessenberg Matrices

Next suppose $\overline{H}_m$ is already computed and we want to solve the least-squares problem:

$$\boldsymbol{y}_m = \arg\min_{\boldsymbol{y}\in\mathbb{R}^m} \left|\left|\beta \boldsymbol{e}_1 - \overline{H}_m \boldsymbol{y}\right|\right|, \tag{3.1}$$

here $\boldsymbol{e}_1$ is the first standard unit vector of $\mathbb{R}^{m+1}$. We already learned that this is equivalent to solving the normal equation:

$$\overline{H}_m^t \overline{H}_m \boldsymbol{y}_m = \beta \overline{H}_m^t \boldsymbol{e}_1 , \tag{3.2}$$

and the idea is to reduce $\overline{H}_m$ to the row echelon form.

The two robust methods we've learned to achieve this are the Givens rotations and the Householder transforms. For general system the latter requires only two-thirds of the computational cost of the former; but when we're dealing with an Hessenberg matrix, only one rotation is needed to reduce each column! Particularly, we first find the Givens rotation $G_1 \in \mathbb{R}^{(m+1)\times(m+1)}$ such that:

$$\begin{aligned} G_1 \overline{H}_m &= \begin{bmatrix} c_1 & -s_1 & & & \\ s_1 & c_1 & & & \\ & & 1 & & \\ & & & \ddots & \\ & & & & 1 \end{bmatrix} \begin{bmatrix} h_{11} & h_{12} & h_{13} & \cdots & h_{1m} \\ h_{21} & h_{22} & h_{23} & \cdots & h_{2m} \\ 0 & h_{32} & h_{33} & \cdots & h_{3m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & h_{m+1,m} \end{bmatrix} \\ &= \begin{bmatrix} h_{11}^{(1)} & h_{12}^{(1)} & h_{13}^{(1)} & \cdots & h_{1m}^{(1)} \\ 0 & h_{22}^{(1)} & h_{23}^{(1)} & \cdots & h_{2m}^{(1)} \\ 0 & h_{32} & h_{33} & \cdots & h_{3m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & h_{m+1,m} \end{bmatrix}, \end{aligned}$$

where

$$c_1 = \frac{h_{11}}{\sqrt{h_{11}^2 + h_{21}^2}}, \quad s_1 = -\frac{h_{21}}{\sqrt{h_{11}^2 + h_{21}^2}}.$$

From this point on, we define G_2 such that $G_2 G_1 \overline{H}_m$ will eliminate h_{32}, G_3 such that $G_3 G_2 G_1 \overline{H}_m$ eliminates h_{43}, etc., until G_m such that $G_m \cdots G_1 \overline{H}_m$ is an upper-tridiagonal matrix:

$$G_m \cdots G_1 \overline{H}_m = \begin{bmatrix} h_{11}^{(m)} & h_{12}^{(m)} & h_{13}^{(m)} & \cdots & h_{1m}^{(m)} \\ 0 & h_{22}^{(m)} & h_{23}^{(m)} & \cdots & h_{2m}^{(m)} \\ 0 & 0 & h_{33}^{(m)} & \cdots & h_{3m}^{(m)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & h_{mm}^{(m)} \\ 0 & 0 & 0 & \cdots & 0 \end{bmatrix} \overset{\text{def}}{=\!=} \begin{bmatrix} \tilde{H}_m \\ \boldsymbol{0}^t \end{bmatrix}, \tag{3.3}$$

where $\tilde{H}_m$ is an $m \times m$ upper-tridiagonal matrix. If there were no round-off errors, the matrix $\overline{H}_m$ generated by Algorithm 2.1 is exactly the same as that generated by Algorithm 1.1. Hence from previous lectures we know that as long as the algorithm does not breakdown (i.e., $h_{m+1,m} \neq 0$), $\tilde{H}_m$ will be non-singular, and we can find $\boldsymbol{y}_m$ by solving:

$$\tilde{H}_m^t \tilde{H}_m \boldsymbol{y}_m = \beta \begin{bmatrix} \tilde{H}_m^t & \boldsymbol{0} \end{bmatrix} G_m \cdots G_1 \boldsymbol{e}_1$$

or equivalently

$$\tilde{H}_m \boldsymbol{y} = \beta \big[\, I_m \;\; \mathbf{0} \,\big] G_m \cdots G_1 \boldsymbol{e}_1 \,. \tag{3.4}$$

This indicates that as we build up the Givens rotations, an additional vector $\boldsymbol{g}$ can be used to track the right hand side of (3.4) by:

$$\boldsymbol{g} = \beta \boldsymbol{e}_1 \,; \quad \boldsymbol{g} \leftarrow G_i \boldsymbol{g} \,,\; i = 1, \cdots, m \,,$$

and finally $\boldsymbol{y}_m$ is $\tilde{H}_m^{-1}$ times the first m components of $\boldsymbol{g}$.

4 Householder GMRES

Now we have all the components that are needed to construct a robust GMRES algorithm. The last technical pieces include keeping the memory usage small and incorporating the stopping criterion in the process of constructing the orthonormal basis.

Combining the Householder Arnoldi algorithm and the least squares solves, there are several variables whose quantity increases as the outer loop continues: the normal basis $\boldsymbol{v}_j$, the vectors $\boldsymbol{z}_j$, the columns of the Hessenberg matrices $\boldsymbol{h}_j$, and the Householder matrices P_j or equivalently the vectors $\boldsymbol{w}_j$. Among these, the $\boldsymbol{z}$ vectors are actually not carried from one iteration to the next, and its memory usage can be limited to a single vector. Variables that are local to each iteration include the Givens matrices G_j (or equivalently the two numbers s_j and c_j), and the upper triangular matrix $\tilde{H}_m$ that is reduced from $\overline{H}_m$.

There are a few measures that we can take to reduce the memory requirement:

- The Givens matrices can be constructed along with the Arnoldi process, as for example computing G_1 only requires information of two numbers of the first column of $\overline{H}_m$, which is the same for all m.
- The size of $\tilde{H}_m$ is almost the same as that of $\overline{H}_m$, which can be significant as m becomes large. However, we can always recover $\overline{H}_m$ from $\tilde{H}_m$ and the Givens rotations; hence it is possible not to store $\overline{H}_m$ at all and only allocate spaces for $\tilde{H}_m$.
- The orthonormal basis $\boldsymbol{v}$'s and the Householder vectors $\boldsymbol{w}$'s essentially keep the same information, in the view of line 8 of Algorithm 1.1. Seeing that $\boldsymbol{v}$'s are only used later to construct the solution $\boldsymbol{x}_m = \boldsymbol{x}_0 + V_m \boldsymbol{y}_m$, their storage can be eliminated if we can find an alternative way to update $\boldsymbol{x}$'s.

The resulting GMRES algorithm is given in Algorithm 4.1.

Algorithm 4.1 Householder GMRES with stopping criterion

1: Compute $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$ and $\boldsymbol{z} = \boldsymbol{r}_0$
2: Set a maximum iteration number m and allocate $m \times m$ matrix $\tilde{H}_m = [\tilde{h}_{ij}]$; set $\tilde{H}_m = 0$
3: **for** $j = 1, \cdots, m, m+1$ **do**
4: Compute the Householder vector $\boldsymbol{w}_j$ such that:
5: The first $j-1$ components of $\boldsymbol{w}_j$ are zero
6: $(I - 2\boldsymbol{w}_j \boldsymbol{w}_j^t)\boldsymbol{z}$ is a multiple of $\boldsymbol{e}_j \in \mathbb{R}^n$
7: $P_j = I - 2\boldsymbol{w}_j \boldsymbol{w}_j^t$
8: $\boldsymbol{h} = P_j \boldsymbol{z}$
9: **if** $j = 1$ **then**
10: Set β to the first component of $\boldsymbol{h}$
11: $\boldsymbol{g} = \beta \boldsymbol{e}_1$
12: **else**
13: $\boldsymbol{h} \leftarrow G_{j-2} \cdots G_1 \boldsymbol{h}$
14: Denoting $\boldsymbol{h} = [h_i]$, compute $s_{j-1} = -\frac{h_j}{\sqrt{h_j^2 + h_{j-1}^2}}$ and $c_{j-1} = \frac{h_{j-1}}{\sqrt{h_j^2 + h_{j-1}^2}}$
15: $\tilde{\boldsymbol{h}}_{j-1} = G_{j-1} \boldsymbol{h}$
16: $\boldsymbol{g} \leftarrow G_{j-1} \boldsymbol{g}$
17: $\boldsymbol{y} = \tilde{H}_{j-1}^{-1} \boldsymbol{g}_{j-1}$ and set $\boldsymbol{u} = 0$
18: **for** $i = j-1, \cdots 1$ **do**
19: $\boldsymbol{u} \leftarrow P_i(y_i \boldsymbol{e}_i + \boldsymbol{u})$
20: **end for**
21: $\boldsymbol{x} = \boldsymbol{x}_0 + \boldsymbol{u}$
22: **if** $||\boldsymbol{b} - A\boldsymbol{x}|| \leq \varepsilon$ **then**
23: Break
24: **end if**
25: **end if**
26: $\boldsymbol{v} = P_1 P_2 \cdots P_j \boldsymbol{e}_j$
27: **if** $j \leq m$ **then**
28: $\boldsymbol{z} = P_j P_{j-1} \cdots P_1 A \boldsymbol{v}$
29: **end if**
30: **end for**
31: Return $\boldsymbol{x}$ as the solution and any diagnostic signals

Here in line 17 $\tilde{H}_j$ is the upper-left $j \times j$ part of $\tilde{H}_m$ and $\boldsymbol{g}_j$ is composed of the first j components of $\boldsymbol{g}$. Also, in line 19 the number y_i is the i-th component of the vector $\boldsymbol{y}$ computed at line 17. Furthermore, whenever P_j and G_j are encountered, it is understood that the matrices are not actually formed; instead the information from $\boldsymbol{w}_j$ and (s_j, c_j) are utilized. Lastly, G_0 is taken to be the identity matrix. The changes in Algorithm 4.1 that do not appear before are not difficult to understand, for example:

- In line 8, the vector $\boldsymbol{h}$ is actually the column $\boldsymbol{h}_{j-1}$ of the matrix $\overline{H}_j$; but we do not carry it to the next iteration and instead we compute in line 15 the $(j-1)$-th column of $\tilde{H}_j$ after applying all the Givens rotations to $\boldsymbol{h}$.
- We only keep one copy of the $\boldsymbol{v}$ vector and one copy of the $\boldsymbol{z}$ vector (line 26 and line 28), the

latter is carried on to the next iteration to compute the next $\boldsymbol{h}$.

- The most significant change is in lines 18–21. To under stand this, we refer to the next expansion:

$$\begin{aligned}\boldsymbol{x}-\boldsymbol{x}_0 &= V_{j-1}\boldsymbol{y} = y_1\boldsymbol{v}_1 + y_2\boldsymbol{v}_2 + \cdots + y_{j-1}\boldsymbol{v}_{j-1} \\ &= y_1 P_1\boldsymbol{e}_1 + y_2 P_1 P_2\boldsymbol{e}_2 + \cdots + y_{j-1}P_1P_2\cdots P_{j-1}\boldsymbol{e}_{j-1} \\ &= P_1(y_1\boldsymbol{e}_1 + P_2(y_2\boldsymbol{e}_2 + \cdots + P_{j-2}(y_{j-2}\boldsymbol{e}_{j-2} + P_{j-1}(y_{j-1}\boldsymbol{e}_{j-1})))) \,.\end{aligned}$$

As a last comment, Algorithm 4.1 is usually combined with restarting or truncation techniques to keep the size of $\tilde{H}_m$ small in practice.

Numerical Exercise

This exercise concerns the GMRES algorithm 4.1, and it is a continuation of the previous computational assignment. Implement this algorithm to solve the linear system $A\boldsymbol{x} = \boldsymbol{b}$; the procedure should take the following inputs:

- The matrix A.
- The right hand side b.
- The initial guess $\boldsymbol{x}_0$.
- The tolerance to stop the algorithm ε_0.
- The maximum number of iterations m.

We will reuse the non-symmetric problem from the last time ("`mat_unsym_A.txt`", "`mat_unsym_M.txt`", and "`mat_unsym_b.txt`"); and solve the linear system by using the CGS, BICGSTAB, and GMRES and compare their performances. To fix the idea, let us choose $\varepsilon_0 = 1e-10$ and you will decide a reasonable maximum number of iteration m yourself.

1) Solve the pre-conditioned system $MA\boldsymbol{x} = M\boldsymbol{b}$ with the initial guess $\boldsymbol{x}_0$, and plot the L^2-norm of the residuals in log scale. The residual curves obtained from the three methods should be plotted in **the same** figure.

2) Solve the original system $A\boldsymbol{x} = \boldsymbol{b}$ (no preconditioning) with the initial guess $\boldsymbol{x}_0$, and plot the L^2-norm of the residuals in log scale. The residual curves (if converges to zero) obtained from the three methods should be plotted in **the same** figure.

For both parts, briefly discuss your results.