Coursework report presented for
M4N9

Computational Linear Algebra
Project 3

Yadu *Bhageria*

Mathematics Department
Imperial College London
United Kingdom
January 13, 2017

1. **GMRES without Preconditioning**
   Once again the PageRank problem is solved in this project but by viewing it as a sparse linear system ($\boldsymbol{Ax} = \boldsymbol{b}$)
$$(\boldsymbol{I} - \alpha \boldsymbol{G}^T)\boldsymbol{x} = \mathbf{1}$$
   The main idea behind the generalized minimal residuals (GMRES) method is to minimise the residual $||\boldsymbol{r}_n|| = ||\boldsymbol{b} - \boldsymbol{Ax}_n||$ where $\boldsymbol{x}_n$ is an element from the Krylov subspace spanned by $(\boldsymbol{A}, \boldsymbol{b})$. This is iteratively done until a pre-prescribed tolerance limit for the relative residual, $||\boldsymbol{b} - \boldsymbol{Ax}_n||/||\boldsymbol{b}||$, is achieved. The implementation of this method can be found in the GMRES.m file.

   The number of iterations needed to converge and the top50 sites with $\alpha = 0.85$ and a relative residual tolerance of 10^{-8} can be found using the code in `q1.m` file.

Number of Iterations	29

Table 1: Number of Iterations taken by the GMRES method

Position	Index	Position	Index
1	1489	26	10
2	4392	27	103
3	67	28	7
4	6428	29	2218
5	4824	30	1662
6	2079	31	719
7	1	32	148
8	1490	33	7896
9	1618	34	137
10	2409	35	788
11	18	36	6131
12	1807	37	142
13	998	38	4
14	42	39	15
15	212	40	8717
16	1863	41	9
17	1864	42	94
18	1084	43	35
19	1080	44	75
20	127	45	2217
21	8052	46	83
22	7756	47	5754
23	33	48	11
24	1661	49	55
25	2476	50	7801

Table 2: Index of Top 50 sites found by using the GMRES method

The standard GMRES algorithm[1] is written below. Note that this algorithm labels each iteration as i rather than n to avoid confusion with the size of the input matrix $\boldsymbol{A} \in \mathbb{R}^{n\times n}$.

```
$q_1 = \frac{b}{||b||}$
for $i = 1, 2, \dots$ do
    Perform step $i$ of the Arnoldi Algorithm to find a new vector that spans the Krylov subspace and consequently the next column of $\boldsymbol{H}$.
    $v = \boldsymbol{A}q_n$                                        $\triangleright\ \boldsymbol{A} = (\boldsymbol{I} - \alpha \boldsymbol{G}^T)$
    for $j = 1$ to $i$ do
        $h_{ji} = q_j^\star v$
        $v = v - h_{ji}q_j$
    end for
    $h_{i+1,i} = ||v||$
    $q_{n+1} = \frac{v}{||v||}$

    Now solve the least squares problem $\boldsymbol{y} = \arg\min ||\boldsymbol{e}_1||b|| - \boldsymbol{H}_i\boldsymbol{y}||$
    $\boldsymbol{H}_i = \widetilde{Q}\widetilde{R}$
    Solve $\boldsymbol{H}_i\boldsymbol{y} = \widetilde{Q}\widetilde{R}\boldsymbol{y} = \boldsymbol{e}_1||b||$

    Compute the solution vector for this iteration
    $\boldsymbol{x}_i = \widetilde{Q}\boldsymbol{y}_i$
    Break loop if relative tolerance is achieved
    if $||\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_n||/||\boldsymbol{b}|| <$ Tolerance then
        break
    end if
end for
```

There are four major parts to this algorithm

(a) Doing the i^{th} iteration of the Arnoldi algorithm. This requires a matrix-vector product that is $\mathcal{O}(n^2)$ FLOPs and further FLOPs that are $\mathcal{O}(in)$.

(b) Performing the QR factorization of $\boldsymbol{H}_i$ that requires $\mathcal{O}(i^3)$ FLOPs.

(c) Computing the current estimate solution $\boldsymbol{x}_i$ that is again a matrix vector multiplication and thus requires $\mathcal{O}(i^2)$ FLOPs.

(d) Checking to see if the relative residual tolerance has been achieved. This is simply taking the norm of vectors of size i and thus requires $\mathcal{O}(i) FLOPs$

It is clear that this basic algorithm requires most iterations at the i^{th} iteration for the QR factorization of $\boldsymbol{H}_i$, i.e. $\mathcal{O}(i^3)$. Next the computation of $\boldsymbol{x}_n$ requires $\mathcal{O}(i^2)$ computations. Thus algorithm implemented by me tries to minimize these iterations to $\mathcal{O}(i)$ at the i^{th} iteration.

[1]from lecture notes

The GMRES algorithm implemented by me is below. Refer to the code in GMRES.m for even more detail. Note that this algorithm also labels each iteration as i rather than n to avoid confusion with the size of the input matrix $\boldsymbol{A} \in \mathbb{R}^{n\times n}$, that $= (\boldsymbol{I} - \alpha \boldsymbol{G}^T)$.

```
$q_1 = \frac{b}{||b||}$                                          ▷ In this case $\boldsymbol{b} = \boldsymbol{1}$
for $i = 1, 2, \dots$ do
    Step $i$ of the Arnoldi Algorithm
    $v = Aq_n$                                                   ▷ $\boldsymbol{A} = (\boldsymbol{I} - \alpha \boldsymbol{G}^T)$
    for $j = 1$ to $i$ do
        $h_{ji} = q_j^{\star} v$
        $v = v - h_{ji} q_j$
    end for
    $h_{i+1,i} = ||v||$
    $q_{n+1} = \frac{v}{||v||}$
    Now solve the least squares problem $\boldsymbol{y} = \arg\min ||\boldsymbol{e}_1||b|| - \boldsymbol{H}_i \boldsymbol{y}||$
    Perform Given's Rotations on new values
    for $j = 1$ to $i - 1$ do
        $T_1 = h_{j,i}$'                                         ▷ Store temporary variable
        $T_2 = h_{j+1,i}$                                        ▷ Store temporary variable
        $h_{j,i} = c_j * T_1 - s_j * T_2$
        $h_{j+1,i} = s_j * T_1 + c_j * T_2$
    end for
    Compute coefficients for new Givens Rotation that sets H(i+1,i) to 0
    $r = \sqrt{h_{i,i}^2 + h_{i+1,i}^2}$
    $c_i = h_{i,i}/r$
    $s_i = -h_{i+1,i}/r$
    Compute values using the rotation
    $h_{i,i} = r$                                                ▷ $r = c_i * H_{i,i} - s_i * H_{i+1,i}$
    $H_{i+1,i} = 0$                                              ▷ By definition of the rotation
    $y_{i+1} = s_i * y_i$                                        ▷ Using the fact that $y_{i+1} = 0$
    $y_i = c_i \cdot b_i$                                        ▷ Using the fact that $b_{i+1} = 0$
    Break loop if relative tolerance is achieved
    if then $\frac{|y_{i+1}}{||b||} <$ Tolerance
        break
    end if
    Find y to minimize $||\tilde{H}_i y - ||b||e_1||$
    $x_i = \hat{Q}_i y$
end for
Backwards substitution
for $j = i : -1 : 1$ do
    $y_j = y_j - H_{j,j+1:i} \cdot y_{j+1:i}$
    $y_j = y_j / h_{j,j}$
end for
Compute solution
$x = Q_{:,1:i} \cdot y_{1:i}$
```

In the next section I go through the algorithm explaining it and the optimizations made as well as calculating the number of FLOPs it takes, and .

(a) First for each iteration in the algorithm, the Arnoldi algorithm is performed to find the new vector spanning the Krylov subspace. By utilising the sparsity of $\boldsymbol{A}$, that is $= (\boldsymbol{I} - \alpha \boldsymbol{G}^T)$, it is possible to make the computation of $v = \boldsymbol{A} q_n$ fast rather than $\mathcal{O}(n^2)$ as in the basic GMRES algorithm.
Now the for loop in the Arnoldi algorithm involves two calculations, $h_{ji} = q_j^\star v$ and $v = v - h_{ji} q_j$, both of which are $\mathcal{O}(n)$. So the loop itself is of order $\mathcal{O}(in)$
Next the calculations of $h_{i+1,i} = ||v||$ and $q_{n+1} = \frac{v}{||v||}$ are both $\mathcal{O}(n)$.
Thus the Arnoldi part of the algorithm is of order $\mathcal{O}(in)$

(b) Now to solve the least squares problem of $\boldsymbol{y} = \arg\min ||\boldsymbol{e}_1||b|| - \boldsymbol{H}_i \boldsymbol{y}||$ normally requires $\mathcal{O}(i^3)$ iterations as QR factorization of $\boldsymbol{H}_i$ has to be performed. But as the structure of $\boldsymbol{H}_i$ is known to be Hessenberg

$$\boldsymbol{H}_i = \begin{pmatrix} h_{1,1} & h_{1,2} & \cdots & & h_{1,i} \\ h_{2,1} & h_{2,2} & \cdots & & h_{2,i} \\ & \ddots & & & \vdots \\ & & \ddots & & \vdots \\ & & & h_{i,i-1} & h_{i,i} \\ & & & & h_{i+1,i} \end{pmatrix}$$

it is possible to do this calculation in $\mathcal{O}(i)$ time by using i Givens Rotations to turn $\boldsymbol{H}_i$ into an upper triangular matrix by eliminating the values below the diagonal. Thus allowing a solution to be found using backwards substitution later. This process also actually makes it very straightforward to calculate the residual at iteration i eliminating the need to calculate $\boldsymbol{x}_i$ at each iteration. More on that later. The following explanation is taken heavily from the given text: `SaadGMRES.pdf`.
Consider the i^{th} iteration and the rotation matrix, Ω_j, that turns the value of $h_{j+1,j}$ to 0. So

$$\Omega_j = \begin{pmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & c_j & s_j & & \\ & & -s_j & c_j & & \\ & & & & \ddots & \\ & & & & & 1 \end{pmatrix}$$

And then

$$\boldsymbol{H}_i \boldsymbol{y} = ||b|| \boldsymbol{e}_1$$
$$\Omega_1 \boldsymbol{H}_i \boldsymbol{y} = \Omega_1 ||b|| \boldsymbol{e}_1$$

That is

$$\begin{pmatrix} c_1 & s_1 & & \\ -s_1 & c_1 & & \\ & & \ddots & \\ & & & 1 \end{pmatrix} \begin{pmatrix} h_{1,1} & h_{1,2} & \dots & & h_{1,i} \\ h_{2,1} & h_{2,2} & \dots & & h_{2,i} \\ & \ddots & & & \vdots \\ & & & h_{i,i-1} & h_{i,i} \\ & & & & h_{i+1,i} \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} c_1 & s_1 & & \\ -s_1 & c_1 & & \\ & & \ddots & \\ & & & 1 \end{pmatrix} \begin{pmatrix} ||b|| \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

Which gives

$$\begin{pmatrix} h_{1,1}^{(1)} & h_{1,2}^{(1)} & \dots & & h_{1,i}^{(1)} \\ 0 & h_{2,2}^{(1)} & \dots & & h_{2,i}^{(1)} \\ & h_{3,2} & & & \vdots \\ & & \ddots & & \vdots \\ & & & h_{i,i-1} & h_{i,i} \\ & & & & h_{i+1,i} \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} \beta_1^{(1)} \\ \beta_2 \\ \vdots \\ 0 \end{pmatrix}$$

Similarly now $h_{3,2}$ can be eliminated by multiplying both sides of the equation from the left by Ω_2. This process can be repeated for each of $h_{j+1,j}$ for $j = 1, \dots, n$ by multiplying in order by $\Omega_1, \dots, \Omega_n$. The final system then looks like

$$\begin{pmatrix} h_{1,1}^{(5)} & h_{1,2}^{(5)} & \dots & h_{1,i}^{(5)} \\ 0 & h_{2,2}^{(5)} & \dots & h_{2,i}^{(5)} \\ & & \ddots & \vdots \\ & & & h_{i,i}^{(5)} \\ & & & 0 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} \beta_1^{(5)} \\ \beta_2^{(5)} \\ \vdots \\ \beta_5^{(5)} \\ \beta_6 \end{pmatrix}$$

That is

$$\Omega_1 \dots \Omega_i \boldsymbol{H}_i \boldsymbol{y} = \boldsymbol{U}_i \boldsymbol{y} = \Omega_1 \dots \Omega_i ||b|| \boldsymbol{e}_1 = \boldsymbol{\beta}$$

Now since each Ω_j is unitary then so is their product defined by $\boldsymbol{Q}_i = \Omega_1 \dots \Omega_i$ and thus it can be said that

$$\min || ||b|| \boldsymbol{e}_1 - \boldsymbol{H}_i \boldsymbol{y} || = \min || \boldsymbol{\beta} - \boldsymbol{U}_i \boldsymbol{y} ||$$

The minimum value of the right hand side is simply found by solving the reduced $i \times i$ system which removes the $(i+1)^{\text{th}}$. We can find a solution vector $\boldsymbol{y}^\star$ on the $n \times n$ part of the system $\boldsymbol{U}_i \boldsymbol{y} = \boldsymbol{\beta}$ using backwards substitution. Thus the residual is simply the $(i+1)^{\text{th}}$ term in the $\boldsymbol{\beta}$ vector. Since we know the residual now, there is no need to compute $\boldsymbol{x}_i$ explicitly at this step of the algorithm and it can simply be computed at the end the final iteration of the GMRES algorithm.

Now this method requires only 1 Givens rotation at each step. This can be proven easily by induction.

For $i = 1$,

$$\boldsymbol{H}_1 = \begin{pmatrix} h_{1,1} \\ h_{2,1} \end{pmatrix}$$

This can be reduced to an upper triangular form with 1 Givens rotation, $\Omega_1 = \begin{pmatrix} c_1 & s_1 \\ -s_1 & c_1 \end{pmatrix}$, s.t.

$$\Omega_1 \boldsymbol{H}_1 = \begin{pmatrix} h_{1,1}^{(1)} \\ 0 \end{pmatrix} = \boldsymbol{U}_1$$

Here $r = \sqrt{h_{1,1}^2 + h_{2,1}^2}$, $c_1 = h_{1,1}/r$, $s_1 = -h_{2,1}/r$.
Now if $\Omega_1 \dots \Omega_i \boldsymbol{H}_i$ is upper-triangular by performing a single Givens rotation at each iteration then $\boldsymbol{H}_{i+1}$ looks like

$$\boldsymbol{H}_{i+1} = \begin{pmatrix} \boldsymbol{U}_i & \boldsymbol{h}_{i+1} \\ 0 & h_{i+2,i+1} \end{pmatrix}$$

where the previous i Givens rotations have been performed on the vector $\boldsymbol{h}_{i+1} \in \mathbb{R}^{i+1}$. And since since $\boldsymbol{U}_i \in \mathbb{R}^{i+1 \times i}$, then $\boldsymbol{H}_{i+1}$ only has one value, the $h_{i+2,i+1}$ entry, that is non-zero in the diagonal below the main diagonal. This can be zeroed using a single Givens rotation where Ω_{i+1} is of the form

$$\Omega_{i+1} = \begin{pmatrix} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & & c_{i+1} & s_{i+1} \\ & & & -s_{i+1} & c_{i+1} \end{pmatrix}$$

where $r = \sqrt{h_{i+1,i+1}^2 + h_{i+2,i}^2}$, $c_{i+1} = h_{i+1,i+1}/r$, $s_{i+1} = -h_{i+2,i+1}/r$. This gives

$$\Omega_{i+1} \boldsymbol{H}_{i+1} = \begin{pmatrix} \boldsymbol{U}_i & \begin{pmatrix} h_{i+1,1} \\ \vdots \\ h_{i+1,i+1}^{(i+1)} \end{pmatrix} \\ 0 & 0 \end{pmatrix}$$

i.e. the $h_{i+1,i+1}$ has changed and obviously $h_{i+2,i+1}$ has been zeroed but none of the other values have changed.
Thus if $\Omega_1 \dots \Omega_i \boldsymbol{H}_i = \boldsymbol{U}_i$ is upper-triangular by performing a single Givens rotation at each iteration then a single givens rotation can make the matrix $\boldsymbol{H}_{i+1}$ upper-triangular as well. So only a single Givens rotation needs to be performed at each step of the iteration.
Implementing the i^{th} Givens rotation requires 5 FLOPs plus 1 square root computation to compute the coefficients c_i and s_i. Since $y_{i+1} = 0$ only 2 FLOPs are needed to compute y_{i+1} and y_i. No computations are required to compute $h_{i,i}$ due to optimisations and $h_{i+1,i}$ is set to 0 by definition of the rotation. Finally at the start of the each new iteration i 6 FLOPs are needed to perform previously done rotations on the new column of values of $\boldsymbol{H}_i$ giving a total of $6(i-1)$ FLOPs.

Thus the entire process of Givens rotations takes $6(i-1)+5$ FLOPs plus 1 square root computation at the i^{th} iteration. Thus this part of the algorithm has been reduced to make i^{th} iteration require only $\mathcal{O}(i)$ FLOPs rather than $\mathcal{O}(i^3)$. Furthermore as discussed before there is no need to explicitly calculate $\boldsymbol{x}_i$ at each iteration - only the last one - so the FLOPs needed for backwards substitution have also been saved.

(c) Next in my GMRES algorithm after the end of the last iterations, backwards substitution is used on the computed $i \times i$ vector $\boldsymbol{y}$. This takes $\mathcal{O}(i^2)$ FLOPs.

(d) Finally the solution vector $\boldsymbol{x}$ is computed by $x = Q_{:,1:i} \cdot y_{1:i}$, which requires $\mathcal{O}(in)$ FLOPs. Once again this is computed only once at the end of the last iteration rather than at every iteration thus saving FLOPs.

Thus the entire implementation is computed in an $\mathcal{O}(in)$ FLOPs. Given an assumption of $i << n$ it can be said that the algorithm finds the solution is $\mathcal{O}(n)$ time. In any case it has been shown how only $\mathcal{O}(i)$ FLOPs are needed at the i^{th} iteration rather than $\mathcal{O}(i^3)$ as asked.

One point I'd like to raise in the implementation of the algorithm for problems where the size of $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ is very large is that storing matrices of size $n \times n$ could be a problem. $\boldsymbol{A}$ might be given in a sparse matrix so would not be a problem itself. In such a scenario the GMRES algorithm could be implemented with a maximum limit on the number of iterations. This would limit the size of the $\boldsymbol{Q}$ and $\boldsymbol{H}$ matrices and only requiring memory allocation on $\mathcal{O}(n)$ thus conversing a significant amount of memory.

2. **Comparison with the Power Method**

$\alpha \backslash$ Tol	10^{-8}	10^{-10}
0.5	16	19
0.7	22	27
0.9	32	37
0.99	41	47
0.9999	48	53

Table 3: Number of GMRES iterations needed without preconditioning

It is clear that as α increases so does the number of iterations needed for the GMRES method to converge. Also as expected a smaller tolerance level for the relative residual requires more iterations as well.

$\alpha \backslash$ Tol	10^{-8}	10^{-10}
0.5	24	30
0.7	45	58
0.9	148	192
0.99	1545	2004
0.9999	155208	201258

Table 4: Number of Power Method iterations needed

The number of iterations needed by the Power Method increases exponentially as $\alpha \to 1$. Furthermore the Power Method takes more iterations than GMRES for even the lower values of α. But this fact is offset by fewer number of FLOPs needed by the Power Iteration for each of its iterations and thus it actually slightly faster than GMRES for smaller values of α. This can be seen in the table below where the Power Method is faster for $\alpha = 0.5$, just barely faster for $\alpha = 0.7$, and slower for $\alpha \geq 0.9$.

	GMRES		Power Method	
α \ Tol	10^{-8}	10^{-10}	10^{-8}	10^{-10}
0.5	7.6235e-03	9.0224e-03	5.2145e-03	6.3042e-03
0.7	1.0756e-02	1.4373e-02	9.2291e-03	1.1510e-02
0.9	1.9047e-02	2.4409e-02	2.7946e-02	3.7016e-02
0.99	2.8994e-02	3.5956e-02	2.8627e-01	3.7244e-01
0.9999	3.7780e-02	4.6905e-02	2.9448e+01	4.0601e+01

Table 5: Time Taken over the given values of α and tolerances for the two methods

It is also worth noting, as mentioned before that the GMRES method can require a large amount of memory allocation and thus perhaps for a problem set where the given matrix of transition probabilities $\boldsymbol{G}$ is larger it might be better to use the Power Method due to memory limitations.

Memory issues aside I think it is reasonable to conclude that in our given case it is better to use the GMRES method for large values of $\alpha \sim> 0.7$ and use the Power Method for values of $\alpha \sim< 0.7$.

The results in this section are produced using the `q2.m` file along with the `sparePageRank.m` and `GMRES.m` files.

3. **GMRES with Preconditioning**

α\ Tol	10^{-8}	10^{-10}
0.5	7	9
0.7	9	11
0.9	13	15
0.99	16	18
0.9999	16	19

Table 6: Number of GMRES iterations needed with preconditioning

The number of iterations needed with preconditioning is significantly fewer than those needed without preconditioning.

A key behaviour though is lost in the number of iterations needed without preconditioning (refer to Table 3 for values). On my machine the number of iterations for GMRES (without preconditoning) to converge with $\alpha = 0.9999$ was 53. But running the same code on the MLC machines I got an iteration count of over 500. None of the other iteration counts were affected. Due to this I also tested the same code on a machine in the computer science department which gave another different value. I do not know why this discrepency occurs but I assume it has something to do with the MATLAB's internal variables and/or versions on different operating systems/machines. In order to test this further on my own machine for consistency in results reported I have considered a value of $\alpha = 0.999999$.

Preconditioning \ Tol	10^{-8}	10^{-10}
Without	53	1047
With	16	19

Table 7: Number of GMRES iterations needed for $\alpha = 0.999999$

It is clear that a jump in the number of iterations required to achieve a tolerance of 10^{-10} occurs for some α very close to 1. Also note that GMRES with preconditioning does not blow up in the number of iterations required. It might be that as $\alpha \to 1$ the problem is somehow getting ill conditioned and/or numerically unstable.

Another way to see the effect of preconditioning is to look at the difference of the solution vectors between GMRES with preconditioning and without.

Tol \ α	0.5	0.7	0.9	0.99	0.9999	0.999999
10^{-8}	5.6049e-09	3.4989e-09	3.8682e-09	1.5699e-09	2.5072e-07	2.5241e-05
10^{-10}	5.6194e-11	2.9910e-11	4.9295e-11	3.5694e-11	7.9382e-10	7.9585e-08

Table 8: Norm of the difference between the solution vectors produced using precondition and not

The difference between the solutions is of small magnitude especially for smaller values of α. Even for values of $\alpha \to 1$ it is on a small order and thus, given that in the implementation for preconditioning sparse LU matrices are used that allow for extremely quick computations of the matrix vector product $My = LUy = c$, it makes sense to always use preconditioning to solve the problem.

The results in this section are produced using the `q3.m` file along with the `GMRES.m` file that is used with the preconditioning parameter to use an incomplete LU decomposition as a preconditioner.