ECE 7650 (Advance Matrix Algorithm)

REPORT

On

"FINAL EXAMINATION FOR FALL 2016"

By

Jamiu Babatunde Mojolagbe

(Student ID: #7804719)

The Department of Electrical and Computer Engineering

University of Manitoba

Submitted to

Ian Jeffrey, PhD

(Course Instructor)

PROBLEM 1

Question 1(i)

This question was implemented in file named "**Q1i.m**". Random matrices of chosen dimension were used and the following results were obtained:

Case 1: For randomly chosen '**m**'

Matrix Dimension	Size of Krylov Subspace (m)	Rank of V
10x10	5	6
20x20	12	13
50x50	35	12
100x100	98	8
200x200	131	7
300x300	211	7
400x400	313	1

Case 2: For fixed '**m**' for different dimensions of input matrix **A**

Matrix Dimension	Size of Krylov Subspace (m)	
	Rank(V) for m=5	**Rank(V) for m=10**
10x10	6	10
20x20	6	11
50x50	6	11
100x100	6	11
200x200	6	11
300x300	6	11
400x400	6	11

Observation:

As it can be observed from the tables, it is obvious that choosing $\mathbf{V} = [\underline{r}\ A\underline{r}\ A^2\underline{r}\ \ldots\ A^{m-1}\underline{r}]$ resulted in linearly dependent bases **V** such that, **V** is rank deficient. While **Case 1** presented above tends to be a generic case and not totally clear, however observing **Case 2** actually led to the conclusion that for a chosen size of Krylov subspace '**m**', provided that **m < dim(A)**, **rank(V) = m+1** and for **m = dim(A)**, **rank(V)** = **m** for linearly independent bases.

Note: *The ranks obtained are more than '**m**' since dimension of **V** is actually **nx(m+1)** but in a specific case* V_m *of dimension **nxm** can be obtained from it by just removing the last column.*

Question 1(ii)

Arnoldi method was implemented based on Modified Gram-Schmidt process as a function called '**arnoldi.m**'. The driver program for this question is named '**Q1ii.m**'. For the sake of clarity, '**Raw V**' used to refer to the bases **V** obtained from $\mathbf{V} = [\underline{r}\ \ A\underline{r}\ \ A^2\underline{r}\ \ .\ .\ .\ \ A^{m-1}\underline{r}]$ while the ones obtained from Arnoldi iteration is termed '**Arnoldi V**'.

Case 1: For randomly chosen '**m**'

Matrix Dimension	Size of Krylov Subspace (m)	Rank of	
		Raw V	Arnoldi V
10x10	9	10	10
20x20	11	12	12
50x50	41	13	42
100x100	78	11	79
200x200	153	8	154
300x300	240	4	241
400x400	297	2	298

Case 2: For fixed '**m**' for different dimensions of input matrix **A**

Matrix Dimension	Rank			
	V for m=5		V for m=10	
	Raw V	Arnoldi V	Raw V	Arnoldi V
10x10	6	6	10	10
20x20	6	6	11	11
50x50	6	6	11	11
100x100	6	6	11	11
200x200	6	6	11	11
300x300	6	6	11	11
400x400	6	6	11	11

Observation:

From the table shown above for **Case 1**, it can be observed that using Arnoldi method, the bases **V** obtained are linearly independent such the rank of **V** are now **m+1** (for the reasons previously presented). If $\mathbf{V_m}$ is considered, that is **V** of dimension **nxm**, then **rank(V) = m**.

Hence forth, the table in **Case 2** perfectly confirm the assertion made in the "**Observation**" section of the previous question, that is **Question 1(i)** above for **Case 2**.

Note: *The ranks obtained are more than '**m**' since dimension of **V** is actually **nx(m+1)** but in a specific case $\boldsymbol{V_m}$ of dimension **nxm** can be obtained from it by just removing the last column.*

PROBLEM 2

Question 2(i)

While it is necessary to say that Arnoldi process in itself does not directly access the entries of a given matrix but instead makes the matrix map vectors and as such reaches its conclusions from their images. However, it is important to show rigorously or partially rigorously that for $\mathbf{V_m^T A V_m = H_m}$, when **m=n** that the eigenvalues of $\mathbf{H_m = H_n}$ are equal to the eigenvalues of **A**.

Recall that square matrices say, A and B, are related by:

$$\mathbf{B = T^{-1}AT = T^{T}AT}$$

where $\mathbf{T^{-1}}$ => Non-singular matrix (that is, invertible)

The transformation represented by $\mathbf{T^{-1}AT}$ above is known as similarity transformation or conjugation by matrix **T**.

Suppose:

$$\mathbf{A} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$$

These two matrices are similar under transformation or conjugation by **T**, such that **T** is given by

$$\mathbf{T} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

Since a linear transformation is the same as a matrix after a basis, says bi, is chosen; then it can be shown that:

$$\mathbf{L}(\Sigma\lambda_i b_i) = \Sigma a_{ji}\lambda_i b_j$$

Now, changing the basis, changes the co-efficient of the matrix such that

$$\mathbf{L}(\Sigma\gamma_i e_i) = \Sigma a_{ji}\gamma_i e_j$$

If operator **L**(V) = AV uses standard basis (Euclidean basis), then L is the matrix $\mathbf{TAT^{-1}}$ with basis

$$\mathbf{b_i = Te_i}$$

Note: *Notice how the eigenvalues are represented to depict that they are images of each other*.

Question 2(ii)

To proof that for the largest eigenvalue of **A** and $\mathbf{H_m}$ are (approximately) equal, a file named "**Q2ii.m**" was used. Random matrices were used as well as random initial vector and $\mathbf{H_m}$ was calculate from Arnoldi method previously implemented. The following results were obtained:

Case 1: For randomly chosen '**m**'

Matrix Dimension	Size of Krylov Subspace (m)	Maximum Eigenvalue of	
		A	$\mathbf{H_m}$
10x10	8	-3.9721	-3.9752
20x20	18	-3.735+1.5751i	-3.7347+1.5762i
50x50	33	8.3546	8.3544
100x100	93	-2.03653+9.94841i	-2.03653+9.94841i
200x200	171	-2.82874+15.0757i	-2.82874+15.0757i
300x300	266	-4.39007+16.9614i	-4.39007+16.9614i
400x400	333	19.5548+6.06034i	19.5548+6.06034i

Case 2: For fixed '**m**' for different dimensions of input matrix **A**

Matrix Dimension	Maximum Eigenvalue			
	m=5		**m=10**	
	A	$\mathbf{H_m}$	**A**	$\mathbf{H_m}$
10x10	2.1836+2.2106i	-0.99642+2.7697i	-2.6912	-2.6912
20x20	5.1404	4.9816	4.3142+1.3456i	-4.4952
50x50	-8.6843	-8.8097	-2.5343+7.2262i	-5.2299+5.1851i
100x100	-8.62056+5.87658i	-6.8939	9.28855+4.45615i	8.4777+4.5603i
200x200	15.1315+2.55873i	-7.7372	-8.25263+12.1035i	8.94859+10.3865i
300x300	-9.26965+15.9366i	-10.9218+7.77496i	18.424	-8.13645+11.1553i
400x400	-7.43289+18.9942i	-9.83433+6.59414i	20.7206	-10.4521+12.1804i

Observation:

Though **Case 1** uses randomly chosen '**m**' but it really does so such that it is very useful for arriving at a very good conclusion here. Why? Because the randomly chosen '**m**'s in **Case 1** above tend to be less that the dimension of a but **are not much less**, which is good for our purpose here. Therefore, from table for **Case 1**, it can be concluded that when **m < n** but not much less, that is when the value of **m** approaches the dimension **n**, then the eigenvalues of $\mathbf{H_m}$ tend to mimic eigenvalues of **A** - and in fact some cases in **Case 1** above the eigenvalue of **A** is equal to eigenvalue of $\mathbf{H_m}$ for that given precision.

Observing **Case 2** above, however, clear things up and it can be concluded that when **m<<n** then the eigenvalues (maximums are used here) of $\mathbf{H_m}$ are not in any way approximately equals to eigenvalues of **A**.

Question 2(iii)

Give a matrix $\mathbf{A^{nxn}}$, show that when $\mathbf{m < n}$, $\mathbf{A = V_m H_m V_m^T}$ is true or false.

Recall from Arnoldi that:

$$\mathbf{H_m = V_m^T A_m V_m} \quad \text{----------} \quad \text{(i)}$$

where $\mathbf{V_m}$ is **nxm**
$\mathbf{H_m}$ is **mxm**
A is **nxn**

As given in the question:

$$\mathbf{A = V_m H_m V_m^T} \quad \text{----------} \quad \text{(ii)}$$

Put (i) into (ii)

$$\mathbf{A = (V_m V_m^T) A (V_m V_m^T)}$$

When **n=m**,

then $\mathbf{V_m V_m^T = V_m^T V_m}$= Identity matrix (**I**) => unitary matrix
therefore, $\mathbf{A = IAI = A}$

When **n < m**

$\mathbf{A = (V_m V_m^T) A (V_m V_m^T)}$
$\mathbf{V_m V_m^T} \neq I$
thus; $\mathbf{A \neq V_m V_m^T A_m V_m V_m^T}$

Finally, it can be concluded that for the inequality n < m, statement $\mathbf{A = V_m H_m V_m^T}$ is not true (that is false) for given real matrix $\mathbf{A^{nxn}}$.

PROBLEM 3

Question 3(i)

Deriving the directional derivative of cost functional given below in terms of the arbitrary direction vector **h** using the real scalar parameter **ε**:

$$\|b - Ax\|_2^2$$

Recall that the directional derivative is given by:

$$\nabla_h f(x) = \left.\frac{df(\underline{\check{x}} + \varepsilon\underline{h})}{d\varepsilon}\right|_{\varepsilon=0} = \lim_{\varepsilon\to 0} \frac{df(\underline{\check{x}} + \varepsilon\underline{h}) - f(\underline{\check{x}})}{\varepsilon}$$

Now, f(x) $= \|b - Ax\|_2^2$

$= < b - Ax,\ b - Ax >$

$= \lim_{\varepsilon\to 0} \frac{1}{\varepsilon} < b - Ax - \varepsilon Ah,\ b - Ax - \varepsilon Ah > - < b - Ax,\ b - Ax >$

since $r = b - Ax$

$= \lim_{\varepsilon\to 0} \frac{1}{\varepsilon} < r - \varepsilon Ah, r - \varepsilon Ah > - < r,\ r >$

$= \lim_{\varepsilon\to 0} \frac{1}{\varepsilon} ((r - \varepsilon Ah)^T (r - \varepsilon Ah) - r^T r)$

$= \lim_{\varepsilon\to 0} \frac{1}{\varepsilon} ((r - \varepsilon Ah)^T r + (r - \varepsilon Ah)^T (-\varepsilon Ah) - r^T r)$

$= \lim_{\varepsilon\to 0} \frac{1}{\varepsilon} (r^T r + (-\varepsilon Ah)^T r + r^T (-\varepsilon Ah) + (-\varepsilon Ah)^T (-\varepsilon Ah) - r^T r)$

$= \lim_{\varepsilon\to 0} \frac{1}{\varepsilon} (< r,\ -\varepsilon Ah > + < -\varepsilon Ah, r > + < -\varepsilon Ah,\ -\varepsilon Ah >)$

$= \lim_{\varepsilon\to 0} \frac{1}{\varepsilon} (-\varepsilon < r,\ Ah > -\varepsilon < -Ah, r > -\varepsilon^2 < Ah,\ Ah >)$

$= - < r,\ Ah > - < Ah,\ r >$

$= -2 < r,\ Ah >$

If **h** = **r**, then

$$\nabla_h f(x) = \left.\frac{df(\underline{\check{x}} + \varepsilon\underline{h})}{d\varepsilon}\right|_{\varepsilon=0} = -2 < r,\ Ar >$$

Question 3(ii)

For step length $\boldsymbol{\alpha}$, direction $\underline{\mathbf{d}}$, evaluate functional $(\underline{\tilde{x}} + \alpha \underline{d})$

$$\alpha = \underset{\alpha}{argmin} < b - Ax \ - \alpha Ad, \ b - Ax \ - \alpha Ad >$$

Recall, r = b - Ax

$$\alpha = \underset{\alpha}{argmin} < r \ - \alpha Ad, \ r - \alpha Ad >$$

Given that, $\underline{d} = \underline{r}$

$$\alpha = \underset{\alpha}{argmin} < r \ - \alpha Ar, \ r \ - \alpha Ar >$$

$$= \underset{\alpha}{argmin} ((r \ - \alpha Ar)^T (r \ - \alpha Ar))$$

$$= \underset{\alpha}{argmin} ((r \ - \alpha Ar)^T r \ + \ (r \ - \alpha Ar)^T (-\alpha Ar))$$

$$= \underset{\alpha}{argmin} (r^T r \ + \ (-\alpha Ar)^T r + \ r^T (-\alpha Ar) \ + \ (-\alpha Ar)^T (-\alpha Ar) \ )$$

$$= \underset{\alpha}{argmin} (r^T r - \ \alpha r^T A^T r - \ \alpha r^T Ar + \ (-\alpha r^T A^T)(-\alpha Ar) \ )$$

$$= \underset{\alpha}{argmin} (r^T r - \ \alpha r^T A^T r - \ \alpha r^T Ar + \ \alpha^2 r^T A^T Ar \ )$$

Now, derivative of the function with respect to α is equal to 0

$$= - r^T A^T r - \ r^T Ar + \ 2\alpha r^T A^T Ar \ = \ 0$$

$$2\alpha r^T A^T Ar \ = \ r^T A^T r + \ r^T Ar$$

$$\alpha \ = \ \frac{r^T A^T r + \ r^T Ar}{2 r^T A^T Ar}$$

$$\alpha \ = \ \frac{< r, \ Ar > \ + < Ar, \ r >}{2 < Ar, \ Ar >} \ = \ \frac{2 < r, \ Ar >}{2 < Ar, \ Ar >} \ = \frac{< r, \ Ar >}{< Ar, \ Ar >}$$

$$\alpha \ \ = \frac{< r, \ Ar >}{< Ar, \ Ar >}$$

Comparing the result obtain for α above with the alpha in **Minimum Residual 1-D**, shows that both $\alpha's$ has the same formulas

Question 3(iii)

From 3(i) above,

$$\nabla_h f(x) = -2 < r,\ Ar >$$

Such that,

$$-\nabla_h f(x) = A^T r$$

Therefore:

$$\alpha = \underset{\alpha}{argmin} < b - Ax\ - \alpha Ad, \qquad b - Ax\ - \alpha Ad >$$

$$\alpha = \underset{\alpha}{argmin} < r\ - \alpha Ad,\ r\ - \alpha Ad > \qquad \text{since } \mathbf{r = b - Ax}$$

$$= \underset{\alpha}{argmin} < r\ - \alpha AA^T r,\ r\ - \alpha AA^T r > \qquad \text{since } \mathbf{d = -\nabla_h f(x) = A^T r}$$

$$= \underset{\alpha}{argmin} ((r\ - \alpha AA^T r)^T (\ r - \alpha AA^T r))$$

$$= \underset{\alpha}{argmin} ((r\ - \alpha AA^T r)^T r\ +\ (\ r - \alpha AA^T r)^T (-\alpha AA^T r))$$

$$= \underset{\alpha}{argmin} (r^T r\ +\ (\ -\alpha AA^T r)^T r - \alpha r^T AA^T\ r\ +\ (-\alpha AA^T r)^T (-\alpha AA^T r))$$

$$= \underset{\alpha}{argmin} (r^T r\ - \alpha r^T AA^T\ r - \alpha r^T AA^T\ r\ +\ \alpha^2 r^T AA^T AA^T r\ )$$

$$= \underset{\alpha}{argmin} (r^T r\ - 2\alpha r^T AA^T\ r +\ \alpha^2 r^T AA^T AA^T r\ )$$

The derivative of the function with respect to α is equal to 0

$$=>\ (-2r^T AA^T\ r +\ 2\alpha r^T AA^T AA^T r\ ) = 0$$

$$2\alpha r^T AA^T AA^T r\ = -2r^T AA^T\ r$$

$$\alpha\ =\ \frac{r^T AA^T\ r}{r^T AA^T AA^T r}\ =\ \frac{r^T\ r}{r^T AA^T r}\ \ =\ \frac{<r,\ r>}{<Ar,\ Ar>}\ =$$

$$\alpha\ \ =\ \frac{<r,\ Ar>}{<Ar,\ Ar>}\ =\ \frac{\|r\|_2^2}{\|Ar\|_2^2}$$

Finally, the obtained result is identical to alpha in **Residual Norm Steepest Descent** algorithm.

PROBLEM 4

Question 4(i)

Recall from Arnoldi that,

$$\mathbf{H_m = V_m^T A_m V_m}$$

where $\mathbf{V_m}$ is **nxm**
$\mathbf{H_m}$ is **mxm**
A is **nxn** and symmetric

Let's consider the (**ij**)th entry of matrix $\mathbf{H_m}$

$$(H_m)_{ij} = (V_m^T)_i A (V_m)_j = h_{ij} = v_i^T A v_j$$

Recall that, orthonormal basis for

$$K_j(A,b) = \{v_1 \quad v_2 \quad \ldots \quad v_j\}$$

Thus; $$Av_j \in K_{j+1}(A,b) = span\{v_1 \quad v_2 \quad \ldots \quad v_{j+1}\}$$

Hence, $h_{ij} = v_i^T A v_j = 0$ for $\mathbf{i > j + 1}$

Since, $$v_i \perp span\{v_1 \quad v_2 \quad \ldots \quad v_{j+1}\} \quad for \quad i > j + 1$$

Such that, $$h_{ij} = h_{ji} = v_j^T A v_i$$

Since matrix **A** given is symmetric, then we have symmetric $\mathbf{H_m}$, thus $\boldsymbol{h_{ij} = 0 \quad for \quad j > i + 1}$ as a result of the reason presented/proven above.

Therefore, it can be concluded that for a symmetric matrix **A**, $\mathbf{H_m}$ produce from Arnoldi process is symmetric tridiagonal.

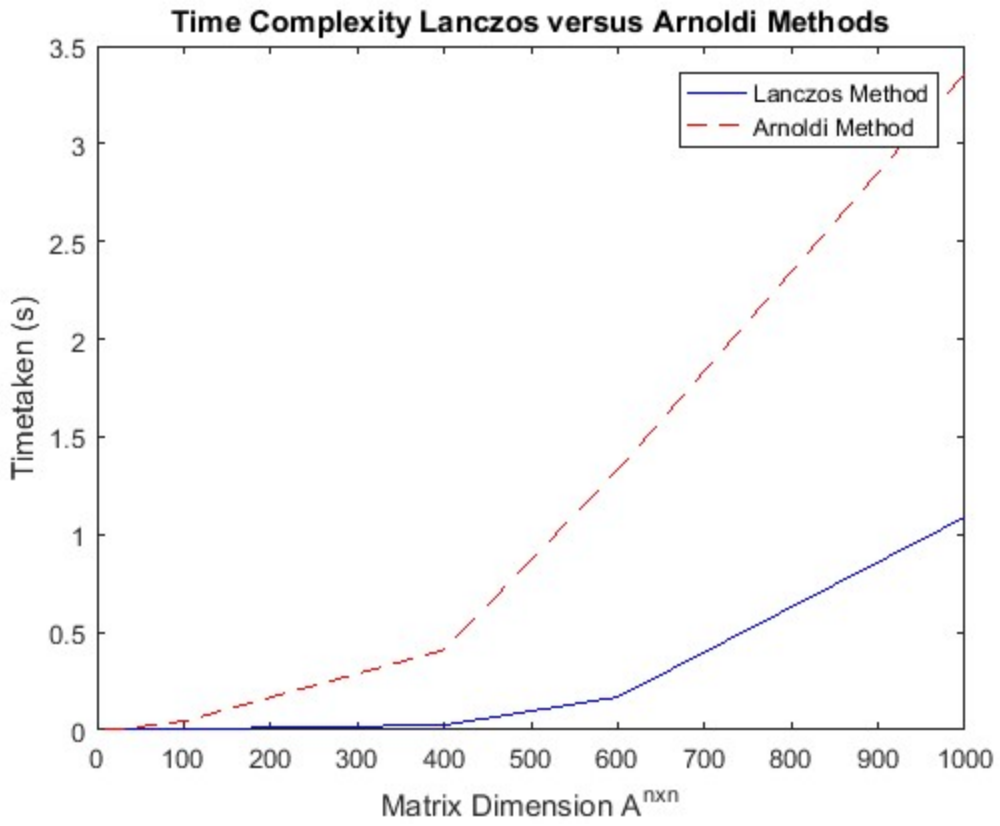
Other Details: For the purpose of confirmation and personal conviction, this proof was tested in file named "**Q4i.m**" for random input matrix, symmetric tridiagonal $\mathbf{H_m}$ was obtained.

Question 4(ii)

Lanczos method was implemented in function named "**lanczos.m**". The driver program for this question is named "**Q4ii.m**". The following results were obtained for different cases tested.

Case 1: For randomly chosen '**m**'

Matrix Dimension	Size of Krylov Subspace (m)	Time taken in seconds	
		Arnoldi Method	**Lanczos Method**
10x10	8	0.00158832017805	0.00113115987732
20x20	11	0.00084607676844	0.000893040266430
100x100	76	0.04512201520370	0.004989871661235
400x400	210	0.40976936152758	0.025197017377595
600x600	490	1.33099212957816	0.168348730389529
1000x1000	694	3.36032083407172	1.089681569138520



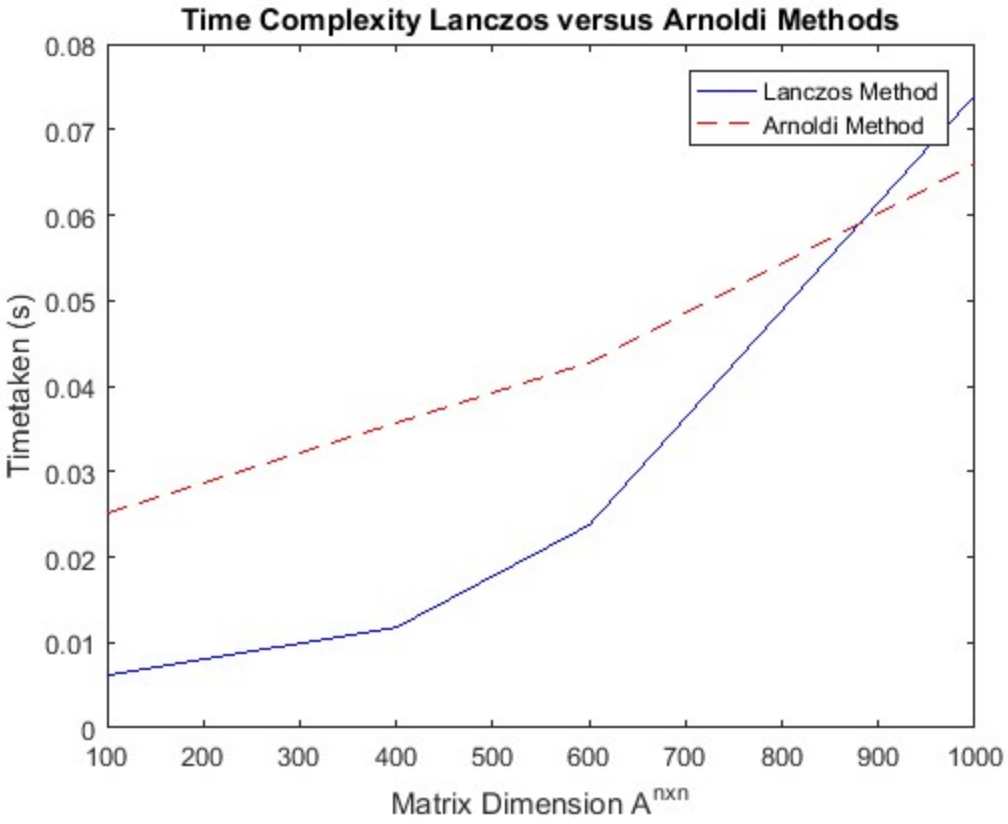
Prove that the algorithm worked was also performed and the result obtained from the various norms were as follows:

```
============= A[10x10] when m = 8 ===================================

*************** Proofs of Orthogonality of Resulting Bases ***************
Norm of Km from Arnoldi:                    1
Norm of Km from Lanczos:                    1
Error: norm(Lanczos(Km) - Arnoldi(Km)):    2.0147e-09
Inner Product (Arnoldi): <v(m), v(m)>:      1
Inner Product (Arnoldi): <v(m), v(m-1)>:   -1.5404e-15
Inner Product (Arnoldi): <v(2), v(m)>:   -5.3257e-12
Inner Product (Lanczos): <v(m), v(m)>:      1
Inner Product (Lanczos): <v1(m), v1(m-1)>: -1.3878e-17
Inner Product (Lanczos): <v1(2), v1(m)>: 6.4199e-14
=========================================================================


============= A[20x20] when m = 11 ===============

*************** Proofs of Orthogonality of Resulting Bases ***************
Norm of Km from Arnoldi:                    1
Norm of Km from Lanczos:                    1
Error: norm(Lanczos(Km) - Arnoldi(Km)):    2.8877e-06
Inner Product (Arnoldi): <v(m), v(m)>:      1
Inner Product (Arnoldi): <v(m), v(m-1)>:   -7.7716e-16
Inner Product (Arnoldi): <v(2), v(m)>:   1.6214e-08
Inner Product (Lanczos): <v(m), v(m)>:      1
Inner Product (Lanczos): <v1(m), v1(m-1)>: 4.3021e-16
Inner Product (Lanczos): <v1(2), v1(m)>: -1.3128e-14
=========================================================================


============= A[100x100] when m = 76 ===============

*************** Proofs of Orthogonality of Resulting Bases ***************
Norm of Km from Arnoldi:                    1.5959
Norm of Km from Lanczos:                    1.4142
Error: norm(Lanczos(Km) - Arnoldi(Km)):    2.18
Inner Product (Arnoldi): <v(m), v(m)>:      1
Inner Product (Arnoldi): <v(m), v(m-1)>:   7.9353e-15
Inner Product (Arnoldi): <v(2), v(m)>:     4.1302e-06
Inner Product (Lanczos): <v(m), v(m)>:      1
Inner Product (Lanczos): <v1(m), v1(m-1)>: 8.2391e-15
Inner Product (Lanczos): <v1(2), v1(m)>: -0.0012023
=========================================================================
```

```
============= A[400x400] when m = 210 ===============

*************** Proofs of Orthogonality of Resulting Bases ****************
Norm of Km from Arnoldi:                    1.8428
Norm of Km from Lanczos:                    1.7321
Error: norm(Lanczos(Km) - Arnoldi(Km)):     2.5176
Inner Product (Arnoldi): <v(m), v(m)>:      1
Inner Product (Arnoldi): <v(m), v(m-1)>:    6.7521e-16
Inner Product (Arnoldi): <v(2), v(m)>:      0.046588
Inner Product (Lanczos): <v(m), v(m)>:      1
Inner Product (Lanczos): <v1(m), v1(m-1)>: -2.8027e-14
Inner Product (Lanczos): <v1(2), v1(m)>:    0.012046
================================================================================


============= A[600x600] when m = 490 ===============

*************** Proofs of Orthogonality of Resulting Bases ****************
Norm of Km from Arnoldi:                    2.0865
Norm of Km from Lanczos:                    2.4495
Error: norm(Lanczos(Km) - Arnoldi(Km)):     3.0614
Inner Product (Arnoldi): <v(m), v(m)>:      1
Inner Product (Arnoldi): <v(m), v(m-1)>:    1.1645e-13
Inner Product (Arnoldi): <v(2), v(m)>:      3.639e-08
Inner Product (Lanczos): <v(m), v(m)>:      1
Inner Product (Lanczos): <v1(m), v1(m-1)>: -2.8271e-14
Inner Product (Lanczos): <v1(2), v1(m)>:    -0.022383
================================================================================


============= A[1000x1000] when m = 694 ===============

*************** Proofs of Orthogonality of Resulting Bases ****************
Norm of Km from Arnoldi:                    1.9411
Norm of Km from Lanczos:                    2.6451
Error: norm(Lanczos(Km) - Arnoldi(Km)):     3.1447
Inner Product (Arnoldi): <v(m), v(m)>:      1
Inner Product (Arnoldi): <v(m), v(m-1)>:    -1.2157e-13
Inner Product (Arnoldi): <v(2), v(m)>:      -2.7093e-07
Inner Product (Lanczos): <v(m), v(m)>:       1
Inner Product (Lanczos): <v1(m), v1(m-1)>: -1.7757e-14
Inner Product (Lanczos): <v1(2), v1(m)>:    -0.034224
================================================================================
```

Case 2: For fixed '**m**' and chosen to be 50, the following time complexity results were obtained:

Matrix Dimension	Size of Krylov Subspace (m)	Time taken in seconds	
		Arnoldi Method	**Lanczos Method**
100x100	50	0.0251170326700840	0.00617166405982563
400x400	50	0.0356834528150765	0.0117577520041122
600x600	50	0.0427276106109621	0.0237334439910755
1000x1000	50	0.0659936210361248	0.0740232784851043



Observation:

From the foregoing, it can be observed from the results and plots that the implementation worked and the error tends to be very small. However, a close look at the results showed also that the bases produced are truly orthogonal but there seems to be loss of orthogonality when comparing says 1^{st} and 500^{th} vector as the error obtained, for **Case 1** for matrix dimensions [**600x600 1000x1000]** for **m = [690 694],** clearly shown.

Moreover, comparing the complexity, it can be seen from the graphs and tables presented that in overall Lanczos method is fantastically faster than Arnoldi. However, when the **m** is fixed at **50** Arnoldi tends to be faster. Why? This misery was uncovered when a new test was performed and the size of Krylov subspace **m = 500** and matrix dimension is **1000x1000**. The time obtained was **2.6839** for Arnoldi and **0.79408** for Lanczos; this thereby led to the conclusion that, when **m** is chosen to **m << n** there is possibility that Arnoldi would be faster than Lanczos.

Question 4(iii)

Recall,

$$\mathbf{H_m = V_m^T A_m V_m}$$

For symmetric trdiagonal let $H_m = T_m$

$$T_m = V_m^T A V_m$$

Such that,

$$y_m = T_m^{-1} A \beta e_1$$

$$x_m = x_0 + V_m y_m \qquad \text{---------- (iii)}$$

Using LU decomposition, $T_m = L_m U_m$

Where L_m = Unit lower bi-diagonal

U_m = Upper bi-diagonal matrix

equation (iii) becomes: $x_m = x_0 + V_m U_m^{-1} L_m^{-1} (\beta e_1)$

let $P_m = V_m U_m^{-1}$

$$z_m = L_m^{-1} (\beta e_1)$$

Therefore, $x_m = x_0 + P_m z_m$

The last column of matrix P_m is given by:

$$p_m = \eta_m^{-1} [V_m - \beta_m p_{m-1}] \qquad \text{where} \quad \eta_m = mth\ G.E.$$

$$\lambda_m = \frac{\beta_m}{\eta_m}$$

$$\eta_m = \alpha_m - \lambda_m \beta_m$$

Recall that,

$$z_m = \begin{bmatrix} z_{m-1} \\ \zeta_m \end{bmatrix}$$

$$\zeta_m = -\lambda_m \zeta_{m-1}$$

Finally, update equation is given by: $x_m = x_0 + \zeta_m p_m$

Implementation Note

The proved algorithm above was implemented in the file named "**lanczosfom.m**" as a function. The driver program for this function is named "**Q4iii.m**". As contained in the driver program, random matrices can be used or an already loaded matrix and right hand side obtained from Finite Element Method solution to a particular problem. Again, for clear comparison, Restarted FOM was implemented to serve as basis for our comparison here; meanwhile the two implementation are compared to the solution obtained from MATLAB's backslash operator which now serves as analytic solution for validation and comparison purposes. While the input matrix dimension is fixed, 157x157, random matrix generation provided a good alternative, though the former is a real time data.

Results and Discussions

The following results were obtained when the implemented function was compared against Restarted Full Orthogonalization Method (FOM):

Case 1: For Krylov subspace, **m = 20** and error tolerance, **tol = 1e-06**

Where **x** = solution from **lanczosfom.m**, **x1** = solution from restarted FOM, **xe** = A\b
LLU is used here to refer to the solution we obtained based on LU
RFOM is used here to refer to the solution obtained from restarted FOM

Matrix Dimension	Time taken (s)		Proof that it works		Number of iterations	
	LLU	RFOM	norm(x-xe)	norm(x1-xe)	LLU	RFOM
50x50	0.0179180	0.0085334	6.7281e-08	6.7281e-08	8	8
100x100	0.0025287	0.0036320	9.048e-08	9.048e-08	7	7
200x200	0.0026633	0.0036283	1.2478e-08	1.2478e-08	7	7
500x500	0.005898	0.005949	2.1109e-08	2.1109e-08	6	6

Case 2: For Krylov subspace, **m = 20** and error tolerance, **tol = 1e-10**

Matrix Dimension	Time taken (s)		Proof that it works		Number of iterations	
	LLU	RFOM	norm(x-xe)	norm(x1-xe)	LLU	RFOM
50x50	0.0015623	0.0032273	2.7607e-12	2.7608e-12	12	12
100x100	0.0031587	0.0032486	2.5796e-12	2.5798e-12	11	11
200x200	0.0030508	0.0037120	9.0409e-13	9.0451e-13	10	10
500x500	0.0033564	0.0027470	7.0452e-13	7.0452e-13	9	9

Case 3: For Krylov subspace, **m = 50** and error tolerance, **tol = 1e-6**

Matrix Dimension	Time taken (s)		Proof that it works		Number of iterations	
	LLU	RFOM	norm(x-xe)	norm(x1-xe)	LLU	RFOM
50x50	0.0015847	0.0019662	6.3966e-08	6.3966e-08	8	8
100x100	0.0023119	0.0034027	8.8003e-08	8.8003e-08	7	7
200x200	0.0025459	0.0024315	8.7779e-09	8.7779e-09	7	7
500x500	0.0043801	0.0046013	1.9508e-08	1.9508e-08	6	6

Observation:

It can be observed from the table that the implemented algorithm worked. It can be seen result wise that increasing the size of Krylov subspace **m** decreases the number of iterations and vice versa. It can also be said that, setting error tolerance very low increases the number of iterations.

Regarding comparison with restarted FOM, it can be observed that this approach tends to reduce the computational time when compared to the restarted FOM as shown in the table above. However, regard the error obtained, there was also slight but not obvious improvement in the error, this is not really obvious because of the chosen precision, though, **Case 2** above somehow show this (for matrix dimension 100x100).

PROBLEM 5

Question 5(i)

Solving Ax = b

Such that: $A \in \mathbb{R}^{nxm}$ $where\ n > m$, that is, over determined system

It is what noting that over determined systems fall under least square problem.

Why QR for this system?

QR provides efficient way in solving non-square system such that Q factor provides orthogonal bases for the span of the columns of given matrix A.

How do we do this?

Soving $Ax \approx b$ as given means, minimizing the functional $\|b - Ax\|_2$

It should be noted that transforming that difference by an orthogonal matrix preserve the norm (Euclidean norm).

Therefore, $\|Q^T(b - Ax)\|_2 = \|Q^T b - Q^T Ax\|_2 = \|Q^T b - Rx\|_2$

Which can be represented in block form as:

$$\left\| \begin{bmatrix} Q_1^T b \\ Q_2^T b \end{bmatrix} - \begin{bmatrix} R_1 \\ 0 \end{bmatrix} \right\|_2 = \left\| \begin{bmatrix} Q_1^T b - R_1 x \\ Q_2^T b \end{bmatrix} \right\|_2$$

⇨ $\|Q_1^T b - R_1 x\|_2 + \|Q_2^T b\|_2$

Since the second term above contains no 'x' then it is not relevant to the functional minimization.

Therefore, minimizing the first term is equivalent to minimizing the norm of the first term.

Given that R_1 to be non – singular (once A is not singular itself)

⇨
$$Q_1^T b - R_1 x = 0$$
$$Q_1^T b = R_1 x$$
$$R_1 x = Q_1^T b$$

Such that the solution, x, can be obtained from the above expression by applying backward substitution. Conclusively, QR factorization reduces the over determined system to a square one. It should also be stated that Q_2 need not to be computed as the QR can be obtained more efficiently if only Q_1 is calculated rather than obtaining all of the Q. This is what "economy size" QR in MATLAB does.

Question 5(ii)

Given:

$$x = x_0 + V(W^T AV)^{-1} W^T r \qquad \text{_____}(i)$$

where, W = AV and V = V_m

Equation (i) can be written as:

$$x = x_0 + V(W^T W)^{-1} W^T r$$

Let $y_m = (W^T W)^{-1} W^T r$ such that, $x = x_0 + V_m y_m$

Thus: simplify y_m

Recall from least-square that, $(A^T A)^{-1} A = A^{-1}(A^T)^{-1} A^T = A^{-1}$

Therefore, $y_m = (W^T W)^{-1} W^T r = W^{-1} r$

Since W = AV as given above, then,

$$y_m = (AV_m)^{-1} r$$

Recall from Arnoldi, $AV_m = V_{m+1} H_{m+1} \approx V_m H_m$

⇨ $$y_m = (V_m H_m)^{-1} r$$
$$y_m = H_m^{-1} (V_m^T r)$$

Where $\mathbf{H_m}$ is **mxm**, $\mathbf{V_m}$ is **nxm**, $\mathbf{H_{m+1}}$ is **(m+1)xm** and $\mathbf{V_{m+1}}$ is **nx(m+1)**

This result can be said to be approximately and theoretically equal to the result of minimizing the functional $\|b - A(x^{(0)} + V_m y_m)\|_2$ which gives $x = x^{(0)} + V_m y_m$

Where $y_m = H_m^{-1} (V_m^T r)$ and letting $\beta = \|r\|_2$

$$v_1 = \frac{r}{\|r\|_2} = \frac{r}{\beta}$$

$$V_m^T r = V_m^T \beta v_1 = \beta e_1$$

⇨ $$y_m = H_m^{-1} (\beta e_1)$$

Question 5(iii)

Implementation Note

Restarted GMRES was implemented based on the guidelines provided, however Modified Gram-Schmidt variant of Lanczos was used. The QR decomposition is obtained via Givens Rotation – which is implemented as a function with name "**givens.m**" and called within the main function. The main function implementation file name is "**gmresrestart.m**" and the driver program for it is named "**Q5iii.m**". Another implementation setback is that inner residual tracking was not implemented, thereby returning inner iteration as the same value as the chosen size of Krylov subspace "**m**". The screen dump of the outer residual is implemented while there is no screen dump for the inner residual. The solution obtained is compared against solution calculated from MATLAB's backslash operator to confirm that the implementation worked.

Results and Discussions

Case 1: For size of Krylov subspace, m = 3, maximum iteration = 40, error tolerance, tol = $1e^{-12}$ and matrix dimension 1000x1000 the following results were obtained:

Where, **x** = solution obtained from implemented Restarted GMRES
xe = solution obtained from the backslash operator

```
===============   ERROR AT EACH OUTER ITERATION =================
Error at outer iteration 1: 0.012427
Error at outer iteration 2: 2.0107e-07
Error at outer iteration 3: 3.7193e-12
Error at outer iteration 4: 1.3936e-15

*************** Proofs that Implementation works  ****************
Norm(x - xe):                   4.3216e-17
```

Case 2: For size of Krylov subspace, m = 10, maximum iteration = 40, error tolerance, tol = $1e^{-12}$ and matrix dimension 1000x1000 the following results were obtained:

```
===============   ERROR AT EACH OUTER ITERATION =================
Error at outer iteration 1: 1.5644e-12
Error at outer iteration 2: 1.4406e-15

*************** Proofs that Implementation works  ****************
Norm(x - xe):                   5.033e-17
```

Case 3: For size of Krylov subspace, m = 3, maximum iteration = 40, error tolerance, tol = $1e^{-6}$ and matrix dimension 1000x1000 the following results were obtained:

```
===============   ERROR AT EACH OUTER ITERATION =================
Error at outer iteration 1: 0.011386
Error at outer iteration 2: 1.7538e-07

*************** Proofs that Implementation works  ***************
Norm(x - xe):                   5.4418e-09
```

Case 4: For size of Krylov subspace, m = 10, maximum iteration = 40, error tolerance, tol = $1e^{-6}$ and matrix dimension 1000x1000 the following results were obtained:

```
===============   ERROR AT EACH OUTER ITERATION =================
Error at outer iteration 1: 1.777e-12

*************** Proofs that Implementation works  ***************
Norm(x - xe):                   5.4878e-14
```

Observation:

From the screen dumps of the results presented above, it can be concluded that the implemented Restarted GMRES worked as the norm of the obtained solution compared to the solution computed by backslash operator tend to be negligible – it is to the machine precision. Having said that, it can also be observed that as the outer iteration increases the residual decreases, that is, technically speaking, the solution tends to be converging as the number of iterations increases until it finally converged.

Finally, another thing that can be noted from the result is what has been previously asserted, that number of iterations increase with the decrease in the size of "m" for a given fixed tolerance value and vice versa. On the same note, setting tolerance very small for fixed "m" and fixed matrix dimension increases the number of iterations and vice versa.

Question 5(iv)

GMRES will converge for any positive definite matrix. It's often used in practice because most practical application problems resulted in either symmetric or non – symmetric positive definite matrices. And owing to the fact that it is efficient compared to other techniques for the same problem makes it a better choice.
In fact, as used in this exam, there is a matrix with its RHS included in the folder named "data" that was obtained from charge distribution on a square plate by Second Order Finite Element Method which turned out to be symmetric and positive definite. This matrix was used throughout the testing phase of this exam. For this reason, it also makes me a fan of Restarted GMRES if a problem of such is to be solved.