1 The four fundamental subspaces

For a matrix $A \in \mathbb{R}^{m \times n}$, there are four fundamental subspaces that we define as follows:

The column space $R(A)$

The column space of A is composed of all vectors $b \in \mathbb{R}^m$ such that b can be written as a linear combination of the columns of A. In other words, the column space of A is the set

$$R(A) = \{b \in \mathbb{R}^m \mid b = Ax \text{ for some } x \in \mathbb{R}^n\}.$$

The nullspace $N(A)$

The nullspace of A is the set of all solutions of the homogeneous system $Ax = 0$. That is,

$$N(A) = \{x \in \mathbb{R}^n \mid Ax = 0\}.$$

The row space $R(A^T)$

The row space of A is composed of all vectors $x \in \mathbb{R}^n$ such that x can be written as a linear combination of the rows of A. In other words, the row space of A is the column space of A^T,

$$R(A^T) = \left\{x \in \mathbb{R}^n \mid x = A^T y \text{ for some } y \in \mathbb{R}^m\right\}.$$

The left nullspace $N(A^T)$

The left nullspace of A contains all vectors y that solve the homogeneous linear system $A^T y = 0$. That is,

$$N(A^T) = \left\{y \in \mathbb{R}^m \mid A^T y = 0\right\}.$$

1.1 Fundamental properties

- The four fundamental subspaces are indeed subspaces. For the column space, for instance, take $b_1, b_2 \in R(A)$. Notice that $\alpha b_1 + \beta b_2 = \alpha A x_1 + \beta A x_2$ for some $x_1, x_2 \in \mathbb{R}^n$. Hence, $\alpha b_1 + \beta b_2 = A(\alpha x_1 + \beta x_2)$, and therefore, $\alpha b_1 + \beta b_2 \in R(A)$. A similar analysis can be done to $N(A)$, $R(A^T)$, and $N(A^T)$.

- The elements of $R(A)$ are orthogonal to the elements of $N(A^T)$. Likewise, the elements of $R(A^T)$ are orthogonal to the elements of $N(A)$. Take for example $b \in R(A)$, and $y \in N(A^T)$. Notice that $b^T y = (Ax)^T y$ for some $x \in \mathbb{R}^n$. Then, $b^T y = x^T A^T y = 0$.
- The rank of A ($rank(A)$) is defined as the maximum number of independent columns of A. That is, $rank(A) = dim(R(A))$. Then, it is always true that $rank(A) = rank(A^T)$.
- Let $rank(A) = r$. Then $dim(N(A)) = n - r$, and $dim(N(A^T)) = m - r$.
- Any solution x of the linear system $Ax = b$ can be written as

  $$x = x_R + x_N,$$

  where $x_R \in R(A^T)$ and $x_N \in N(A)$.

2 The Gram-Schmidt process

The Gram-Schmidt process is a method that takes a finite linear independent set of vectors $\{w_1, w_2, \cdots, w_k\}$ in $\mathbb{R}^n$ with $k \leq n$, and generates an orthonormal basis $\{v_1, v_2, \cdots, v_k\}$ for the $k-$dimensional subspace of $\mathbb{R}^n$.

2.1 Gram-Schmidt process for $k = 2$.

Let $\{w_1, w_2\}$ a set of two linear independent vectors.

1. To generate v_1, we simply take w_1 and normalize it. That is, $v_1 = w_1/\|w_1\|$.
2. In order to generate v_2, we need to make sure that v_2 is orthogonal to v_1.

   To that end, from Figure 1 we observe that the vector $w_2 - P_{v_1} w_2$ meets this requirement, where $P_{v_1} w_2$ stands for the orthogonal projection of the vector w_2 into v_1. Now recall that

   $$P_a b = \left(\frac{a^T b}{a^T a}\right) a.$$

   Therefore,

   $$\hat{v}_2 = w_2 - \left(w_2^T v_1\right) v_1, \text{ and}$$
   $$v_2 = \hat{v}_2/\|\hat{v}_2\|.$$

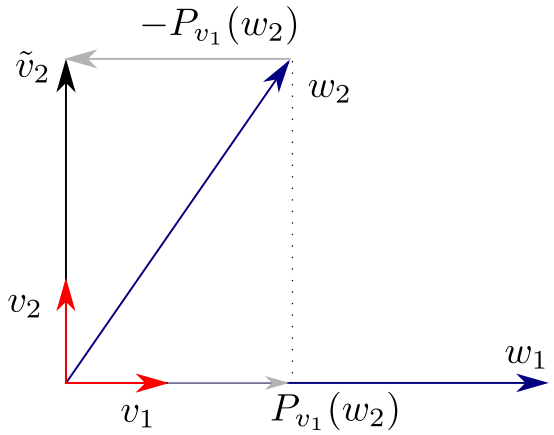

Figure 1: Gram-Schmidt process for $k = 2$.

2.2 Gram-Schmidt process for $k = 3$.

Let $\{w_1, w_2, w_3\}$ a set of three linear independent vectors.

1. To generate v_1 and v_2, we proceed exactly as we did for the case $k = 2$ described above.

2. In order to generate v_3, we need to make sure that v_3 is orthogonal to both v_1 and v_2. From Figure 2, we observe that the vector $w_3 - x$ meets this requirement, where x is the orthogonal projection of w_3 onto the plane that contains w_1 and w_2. Notice that $x = P_{v_1} w_3 + P_{v_2} w_3$. Therefore,

$$\hat{v}_3 = w_3 - \left(w_3^T v_1\right) v_1 - \left(w_3^T v_2\right) v_2, \text{ and}$$
$$v_3 = \hat{v}_3 / \|\hat{v}_3\|.$$

2.3 Gram-Schmidt process for k independent vectors

Let $\{w_1, w_2, \cdots w_k\}$ a set of k linear independent vectors. Following the same ideas as in previous cases, we formulate a general procedure for the Gram-Schmidt process in order to generate an orthonormal set of k vectors $\{v_1, v_2, \cdots v_k\}$. This procedure is described in Algorithm 1.

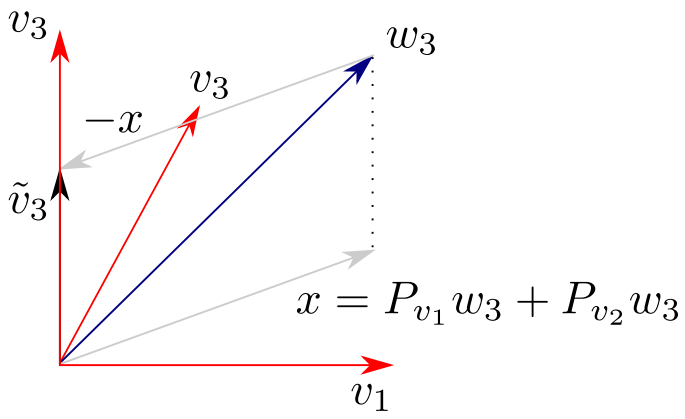

Figure 2: Gram-Schmidt process for $k = 3$.

Algorithm 1 Gram-Schmidt algorithm

Input A set of k linear independent vectors $\{w_1, w_2, \cdots w_k\}$.
Output A set of k orthonormal vectors $\{v_1, v_2, \cdots v_k\}$ that span the same $k-$dimensional subspace as $\{w_1, w_2, \cdots w_k\}$.

Step 1. compute $v_1 = \frac{w_1}{\|w_1\|}$.
Step 2. **For** $i = 1, \ldots, k$
Step 3. $\hat{v}_i = w_i - \sum_{j=1}^{i-1} \left(w_i^T v_j\right) v_j$
Step 4. $v_i = \frac{\hat{v}_i}{\|\hat{v}_i\|}$
Step 5. **End-For**

2.4 MATLAB code for Gram-Schmidt method

The following MATLAB program was written to implement the Gram-Schmidt orthogonalization method.

Listing 1: Gram-Schmidt MATLAB implementation.

```
function [Q R] = myGramSchmidt(W)
% Gram-Schmidt orthogonalization.
% Given an mxn matrix W with l.i. columns. The function returns a
% QR factorization. R is a non-singular upper triangular matrix,
% and Q is a mxn matrix with orthonormal columns.

[m n] = size(V);

Q = zeros(m,n);
R = zeros(n,n);

for k = 1:n

    Q(:,k) = W(:,k);

    for i =1:k-1

        R(i,k) = Q(:,i)'*W(:,k);
        Q(:,k) = Q(:,k) - R(i,k) * Q(:,i);

    end

    Q(:,k) = Q(:,k)/norm(Q(:,k));

    R(k,k) = Q(:,k)'*W(:,k);

end
```

3 Arnoldi's Process

For a given $A \in \mathbb{R}^{n\times n}$ and $b \in \mathbb{R}^n$, we are interested in generating an orthonormal basis for the $k+1-$order Krylov subspace

$$\mathcal{K}_{k+1}(A,b) = \text{span}\left\{b, Ab, \cdots, A^k b\right\}.$$

The Arnoldi's process relies in the Gram-Schmidt method for generating an orthonormal basis $\{v_1, v_2, \cdots v_k\}$. What makes this process special, is that at each step, w_i is computed as $w_i = Aw_{i-1}$.

In order to generate $\mathcal{K}_k(A,b)$, we proceed as follows.

1. Compute $v_1 = \frac{b}{\|b\|}$ as basis for $\mathcal{K}_1(A,b)$.

2. We use an orthonormal basis for $\mathcal{K}_k(A, b)$ to generate an orthonormal basis for the next Krylov subspace $\mathcal{K}_{k+1}(A, b)$. In other words, we require $Av_k \in \mathcal{K}_{k+1}(A, b)$ to be orthogonal to the subspace $\mathcal{K}_k(A, b)$.

3.1 Gram-Schmidt idea

From the Gram-Schmidt iteration described in Algorithm 1, we have that

$$\hat{v}_{k+1} = Av_k - \sum_{i=1}^{k} h_{i,k} v_i, \tag{1}$$

where $h_{i,k} = v_i^T A v_k$ and v_{k+1} is computed as $v_{k+1} = \frac{\hat{v}_{k+1}}{\|\hat{v}_{k+1}\|}$.

Property 3.1 *Let $V_k = [v_1 \;\; v_2 \;\; \cdots \;\; v_k]$ and let $H_{k+1,k}$ be the $k+1 \times k$ upper Hessenberg matrix whose entries $H_{i,j}$ are defined by $h_{i,j}$, and $j \geq i-1$. Then*

$$AV_k = V_{k+1} H_{k+1,k}. \tag{2}$$

From (1) and letting $\|\hat{v}_{k+1}\| = h_{k+1,k}$ we have that

$$\begin{aligned} Av_k &= \|\hat{v}_{k+1}\| v_{k+1} + \sum_{i=1}^{k} h_{i,k} v_i \\ &= \begin{bmatrix} v_1 & \cdots & v_k & v_{k+1} \end{bmatrix} \begin{bmatrix} h_{1,k} \\ \vdots \\ h_{k,k} \\ h_{k+1,k} \end{bmatrix} \\ &= \begin{bmatrix} v_1 & \cdots & v_j & v_{j+1} & \cdots & v_{k+1} \end{bmatrix} \begin{bmatrix} h_{1,k} \\ \vdots \\ h_{k,k} \\ h_{k+1,k} \end{bmatrix} \end{aligned}$$

Then, we have that

$$Av_j = V_{k+1} H_{k+1,k}(:, j),$$

where $H_{k+1,k}(:,j)$ is the $j-$th column of the matrix

$$H_{k+1,k} = \begin{bmatrix} h_{1,1} & h_{1,2} & \cdots & \cdots & h_{1,k} \\ h_{2,1} & h_{2,2} & \cdots & \cdots & h_{2,k} \\ 0 & h_{3,2} & h_{3,3} & \cdots & h_{3,k} \\ \vdots & 0 & \ddots & \ddots & \vdots \\ \vdots & \vdots & \vdots & h_{k,k-1} & h_{k,k} \\ 0 & \cdots & \cdots & \cdots & h_{k+1,k} \end{bmatrix} \in \mathbb{R}^{k+1\times k}.$$

In order to write all the products

$$[Av_1 \;\; Av_2 \;\; \cdots \;\; Av_j \;\; Av_{j+1} \;\; \cdots \;\; Av_k],$$

using matrix notation, we proceed as follows

$$AV_k = V_{k+1}H_{k+1,k},$$

where $V_k = [v_1 \;\; v_2 \;\; \cdots \;\; v_k]$.

Property 3.2 *Let the vector $e_k = [0, \cdots, 1] \in \mathbb{R}^k$, and $H_k = H_{k+1,k}(1 : k, 1 : k) \in \mathbb{R}^{k\times k}$, Then equation (2) can be written as*

$$AV_k = V_kH_k + (h_{k+1,k})v_{k+1}e_k^T, \tag{3}$$

and

$$V_k^TAV_k = H_k. \tag{4}$$

Notice that by construction, the matrices H_{k+1} and $H_{k+1,k}$ can be written as

$$V_{k+1} = \begin{bmatrix} V_k & v_{k+1} \end{bmatrix} \quad H_{k+1,k} = \begin{bmatrix} H_k \\ h_{k+1,k}e_k^T \end{bmatrix}.$$

Now from equation (2), it follows that

$$\begin{aligned} AV_k &= \begin{bmatrix} V_k & v_{k+1} \end{bmatrix} \begin{bmatrix} H_k \\ h_{k+1,k}e_k^T \end{bmatrix} \\ &= V_kH_k + (h_{k+1,k})v_{k+1}e_k^T. \end{aligned}$$

Finally, (3) yields (4) by multiplying both left-sides of (3) by V_k^T and realizing that $V_k^Tv_{k+1} = 0$, since all the columns of V_k are orthogonal to v_{k+1}.

Algorithm 2 Arnoldi's algorithm

Input An $n \times n$ matrix A and $b \in \mathbb{R}^n$.
Output A set of k orthonormal vectors $\{v_1, v_2, \cdots v_k\}$ that span the $k-$dimensional Krylov subspace $\mathcal{K}_{k+1}(A, b) = \text{span}\left\{b, Ab, \cdots, A^k b\right\}$.

Step 1. Compute $v_1 = \frac{b}{\|b\|}$.
Step 2. **For** $i = 1, \ldots, k$
Step 3. $v = Av_i$
Step 4. **For** $j = 1, \ldots, i$
Step 5. $h_{j,i} = q_j^T v$
Step 6. $v = v - hj, iv_j$
Step 7. **End-For**
Step 8. $h_{i+1,i} = \|v\|$, if $h_{i+1,i} = 0$, stop
Step 9. $v_{i+1} = \frac{v}{h_{i+1,i}}$
Step 10. **End-For**

3.2 Arnoldi's Algorithm

We now present the Arnoldi's algorithm in the following pseudocode.

3.3 MATLAB code for Arnoldi method

Based on the discussion above, we present a MATLAB implementation for the Arnoldi process. The user can tell the program to return the appropriate matrices satisfying any of the two relations in (4) and (5) .

Listing 2: Arnoldi Method MATLAB implementation.

```
function [V H E] = myArnoldi(A,b,k,opt)
% -- Arnoldi process --
% Given an nxn matrix A and vector nx1 b.
% The function returns a V, H and E such that:
%
% if opt = 0
% AV = VH + E,
% and V is an nxk orthogonal matrix, H is an kxk upper Hessenberg and
% E is a rank one nxk matrix with zero in its entries except the last column
% k is the number of Arnoldi iterations to perform
% Property:
% V'AV = H
%
```

```
% if opt = 1
% AV(1:end,1:end-1) = VH,
% and V is an nxk+1 orthogonal matrix, H is an (k+1)xk upper Hessenberg.
% k is the number of Arnoldi iterations to perform

if nargin < 4
    opt = 1;
end

n = size(A,1);

V = zeros(n,k);
H = zeros(k,k);

V(:,1) = b/norm(b);

for i = 1:k

    w = A*V(:,i);

    for j =1:i
        H(j,i) = V(:,j)'*w;
        w      = w - H(j,i)*V(:,j);
    end

    H(i+1,i) = norm(w);

    if H(i+1,i) == 0
        break;
    end

    V(:,i+1) = w/H(i+1,i);

end

if opt == 1
    E = [];
    return;
end

if opt == 0
    E = zeros(n,k);
    E(:,end) = H(k+1,k)*V(:,k+1);

    H = H(1:end-1,:);
    V = V(:,1:end-1);
end
```

4 The Galerkin Method

We now focus in solving a linear system of equations $Ax = b$ using the Arnoldi method introduced in the previous section. The basic idea consists

in applying a projection method in the following way:

Give an initial approximation x_0 and compute the corresponding residual, $r_0 = b - Ax_0$. Find a *correction* vector z_k by solving an m-dimensional problem ($m \ll n$) such that

$$A(x_0 + z_k) = Ax_k \equiv b. \tag{5}$$

The vector z_k is determined by computing k steps of the Arnoldi method starting with the normalized initial residual as in $v_1 = r_0/\|r_0\|$. If Algorithm 2 is used, we generate the matrices H_k, V_k, and $H_{k+1,k}$ as in (5). Therefore, the vector z_k can be expressed as

$$z_k = V_k y, \tag{6}$$

for some $y \in \mathbb{R}^k$.

In the Galerkin Method, the residual vector $r_k = b - Ax_k$ is required to be orthogonal to the Krylov space $\mathcal{K}_k(A, r_0)$.

4.1 Hessenberg system for the Galerkin Method

We need to guarantee that the residual $r_k = b - Ax_k$ is orthogonal to the space $\mathcal{K}_k(A, r_0)$. Using (6) we have that $r_k = b - A(x_0 + z_k) = r_0 - AV_k z_k$, so the condition we must enforce is

$$V_k^T(r_0 - AV_k z_k) = 0,$$

that is, $V_k^T AV_k y_k = V_k^T r_0$. Since we are using the Arnoldi method starting with $v_1 = r_0/\|r_0\|$, we get

$$V_k^T r_0 \;=\; \begin{bmatrix} v_1^T \\ v_2^T \\ \vdots \\ v_k^T \end{bmatrix} r_0 = \begin{bmatrix} \frac{r_0^T r_0}{\|r_0\|} \\ v_2^T \|r_0\| v_1 \\ \vdots \\ v_k^T \|r_0\| v_1 \end{bmatrix} = \begin{bmatrix} \|r_0\| \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \|r_0\| e_1,$$

since $v_i^T v_1 = 0$ for $i = 2, ..., k$. Also, notice that from (4), the matrix $V_k^T AV_k$ is just the upper Hessenberg matrix H_k.

Thus, in order to obtain the *correction* vector z_k, we must solve the $k \times k$ Hessenberg system

$$H_k y_k = \|r_0\| e_1, \tag{7}$$

with $e_1 = (1, ..., 0)^T \in \mathbb{R}^k$.

Once the solution of the system (7) is found, then we compute $z_k = V_k y_k$ and the improved solution vector is given by $x_k = x_0 + z_k$. The method is also known as the Full Orthogonalization Method (FOM).

4.2 Residual for the Galerkin Method

We need to compute the residual r_k at every iteration of the Galerkin Method (after k steps of the Arnoldi method have been computed), in order to check for convergence of the method or if it needs to be restarted.

The residual in this case is given by

$$r_k = b - Ax_k = b - A(x_0 + z_k) = r_0 - AV_k y_k.$$

From (5) we obtain

$$\begin{aligned} r_k &= r_0 - (V_k H_k + (h_{k+1,k}) v_{k+1} e_k^T) y_k, \\ &= r_0 - (V_k H_k y_k + (h_{k+1,k}) v_{k+1} e_k^T y_k), \\ &= r_0 - (V_k \|r_0\| e_1 + (h_{k+1,k})(e_k^T y_k) v_{k+1}), \end{aligned}$$

where we have used the relation from (7).
Notice that $V_k \|r_0\| e_1 = \|r_0\| V_k(:, 1) = \|r_0\|(r_0/\|r_0\|)$. Therefore,

$$r_k = -(h_{k+1,k})(e_k^T y_k) v_{k+1}. \tag{8}$$

Finally we can compute the norm of the residual easily since

$$\begin{aligned} r_k^T r_k &= (h_{k+1,k})^2 (e_k^T y_k) v_{k+1}^T (e_k^T y_k) v_{k+1}, \\ &= (h_{k+1,k})^2 (e_k^T y_k)^2 v_{k+1}^T v_{k+1}, \\ &= (h_{k+1,k})^2 (e_k^T y_k)^2. \end{aligned}$$

Thus we have that

$$\|r_k\| = h_{k+1,k} |(e_k^T y_k)|. \tag{9}$$

4.3 Galerkin Algorithm

We now present a Galerkin-type algorithm for solving the system $Ax = b$.

Algorithm 3 Galerkin Algorithm

Input (i) A, an $n \times n$ matrix, (ii) k, a positive integer less than n, (iii) ϵ, the tolerance (> 0), and (iv) x_0 an initial approximation.
Output An approximate solution x_k such that the associated residual vector $r_k = b - Ax_k$ is orthogonal to $\mathcal{K}_k(A, r_0)$

Step 0. compute $r_0 = b - Ax_0$.
Step 1. Run k steps of the Arnoldi Algorithm 2 to generate the matrices V_k, H_k, and $h_{k+1,k}$ using $v_1 = r_0/\|r_0\|$.
Step 2. Solve the $k \times k$ system $H_k y_k = \|r_0\| e_1$.
Step 3. Compute the correction vector $z_k = V_k y_k$.
Step 4. Compute the new solution vector $x_k = x_0 + z_k$.
Step 5. Compute the norm of the new residual vector, $\|r_k\| = h_{k+1,k}|e_k^T y_k|$. Stop if $\|r_k\| < \epsilon$ and accept x_k as the approximate solution.
Step 6. Compute $r_m = -h_{m+1,m} e_m^T y_m v_{m+1}$ and set $x_0 \equiv x_m$ and $r_0 \equiv r_k$. Return to Step 1.

4.4 MATLAB code for the Galerkin method

A MATLAB implementation of the Galerking method for solving the linear system of equations $Ax = b$ is proposed, using Program 2 presented before.

Listing 3: Galerkin Method MATLAB implementation.

```
function [x i] = myGalerkin(A,b,k,epsilon,RESTART)
% Galerkin (iterative) method for solving the linear system
% Ax = b, where A is a nxn matrix, b is a nx1 vector.
% k is the dimension of the space in which the Galerkin method
% will approximate the solution.
% epsilon is tolerance, and RESTART is the number of iterations
% to be performed. The initial point is always the zero vector.

n = size(A,1);
x = zeros(n,1);
r = b;

for i = 1:RESTART

    [V H] = myArnoldi(A,r,k);
    H_k = H(1:end-1,:);
    V_k = V(:,1:end-1);
```

```
    nrm_r = norm(r); e1 = zeros(k,1); e1(1) = 1;

    y = H_k\(nrm_r*e1);
    x = x + V_k*y;

    nrm_r = H(k+1,k)*abs(y(k));

    if nrm_r < epsilon
        break;
    end

    r = -H(k+1,k)*(y(k))*V(:,k+1);

end
```

5 The GMRES Method

The Generalized Minimal Residual Method (GMRES) requires the residual r_k to be minimized. The method was developed by Saad and Schultz in 1986.

For the residual we have the following

$$\begin{aligned} r_k &= b - Ax_k = b - A(x_0 + V_k y_k), \\ &= r_0 - AV_k y_k = r_0 - V_{k+1} H_{k+1,k} y_k, \end{aligned}$$

where we have used the relation in (2). Since, as discussed before, $r_0 = v_1/\|v_1\| = V_{k+1} e_1 \|r_0\|$, we get

$$r_k = V_{k+1}(\|r_0\| e_1 - H_{k+1,k} y_k).$$

The matrix V_{k+1} has orthonormal columns, which means that the norm of the residual is given by

$$\|r_k\|_2 = \left\| \|r_0\| e_1 - H_{k+1,k} y_k \right\|_2 . \tag{10}$$

Therefore, the GMRES method aims to solve the least-squares problem

$$\min_{y \in \mathbb{R}^k} J(y) = \min_{y \in \mathbb{R}^k} \left\{ \left\| H_{k+1,k} y_k - e_1 \|r_0\| \right\|_2 \right\} . \tag{11}$$

Notice that $H_{k+1,k} \in \mathbb{R}^{(k+1)\times k}$, so the equation $H_{k+1,k} y_k = H_{k+1,k} y_k$ describes an overdetermined linear system of equations. Thus, our goal is to solve the least-squares problem (11), whose solution can be obtained using a QR factorization method with Givens rotations (Givens rotations are appropriate for the QR factorization in this case since $H_{k+1,k}$ is an upper-Hessenberg matrix).

5.1 Least-squares problem for the GMRES method

Let $H_{k+1,k} = QR$ be the QR factorization for the Hessenberg matrix $H_{k+1,k}$. This QR factorization has the following form:

$$\begin{aligned} H_{k+1,k} &= Q_{(k+1)\times(k+1)} R_{(k+1)\times k} \\ &= \left[\begin{array}{c|c} \tilde{Q} & q_{k+1} \end{array}\right] \left[\begin{array}{c} \tilde{R} \\ -- \\ 0 \end{array}\right]. \end{aligned}$$

Define the vector $\tilde{g_k}$ by

$$\begin{aligned} \tilde{g_k} &= Q^T(\|r_0\|e_1) \\ &= \left[\begin{array}{c} \tilde{Q}^T(\|r_0\|e_1) \\ -- \\ q_{k+1}^T(\|r_0\|e_1) \end{array}\right] = \left[\begin{array}{c} \gamma_1 \\ \vdots \\ \gamma_k \\ \gamma_{k+1} \end{array}\right], \end{aligned} \tag{12}$$

and let $g_k \in \mathbb{R}^k$ be the vector obtained from $\tilde{g_k}$ by deleting its last component.

The solution of the least-squares problem (11) is characterized by the normal equations

$$\begin{aligned} {H_{k+1,k}}^T H_{k+1,k} y_k &= {H_{k+1,k}}^T \|r_0\|e_1, \\ (QR)^T QR y_k &= R^T Q^T \|r_0\|e_1, \\ R^T R y_k &= R^T \tilde{g_k}, \end{aligned}$$

since

$$\begin{aligned} R^T R &= \left[\begin{array}{c|c} \tilde{R}^T & 0 \end{array}\right] \left[\begin{array}{c} \tilde{R} \\ -- \\ 0 \end{array}\right], \\ &= \tilde{R}^T \tilde{R}. \end{aligned}$$

Thus, the vector y_k that minimizes $\|H_{k+1,k} y_k - \|r_0\|e_1\|_2$ is given by

$$\tilde{R} y_k = g_k, \tag{13}$$

5.2 Residual for the GMRES method

The residual r_k is given by

$$\begin{aligned}
r_k &= r_0 - V_{k+1} H_{k+1,k} y_k, \\
&= V_{k+1}(\|r_0\| e_1 - H_{k+1,k} y_k), \\
&= V_{k+1}\left(\|r_0\| e_1 - \left[\begin{array}{c|c} \tilde{Q} & q_{k+1} \end{array} \right] \left[\begin{array}{c} \tilde{R} \\ \hline 0 \end{array} \right] y_k \right) \\
&= V_{k+1} Q \left(\left[\begin{array}{c} \tilde{Q}^T \\ \hline q_{k+1}^T \end{array} \right] \|r_0\| e_1 - \left[\begin{array}{c} \tilde{R} \\ \hline 0 \end{array} \right] y_k \right), \\
&= V_{k+1} Q \left(\left[\begin{array}{c} \tilde{Q}^T \|r_0\| e_1 - \tilde{R} y \\ \hline \|r_0\| q_{k+1}^T e_1 \end{array} \right] \right), \\
&= V_{k+1} Q \left(\left[\begin{array}{c} \mathbf{0}_{k\times 1} \\ \hline \|r_0\| q_{k+1}^T e_1 \end{array} \right] \right),
\end{aligned}$$

where in the last step we have used the relation from equation (13). Finally we can express the residual r_k as

$$r_k = V_{k+1} Q(\gamma_{k+1} e_{k+1}), \tag{14}$$

using the notation introduced in (12) for the vector $\tilde{g_k}$. With this result, we can easily compute the norm of the residual vector since

$$\begin{aligned}
{r_k}^T r_k &= (\gamma_{k+1})^2 e_{k+1}^T (Q^T V_{k+1}^T V_{k+1} Q) e_{k+1}, \\
&= (\gamma_{k+1})^2 e_{k+1}^T e_{k+1} = (\gamma_{k+1})^2,
\end{aligned}$$

given that the columns of the matrices Q and V_{k+1} form orthonormal sets of vectors. Thus

$$\|r_k\| = |\gamma_{k+1}|, \tag{15}$$

5.3 The GMRES Algorithm

In the following we describe the GMRES algorithm for solving the linear system $Ax = b$.

Algorithm 4 GMRES Algorithm

Input (i) A, an $n \times n$ matrix, (ii) k, a positive integer less than n, (iii) ϵ, the tolerance (> 0), and (iv) x_0 an initial approximation.
Output An approximate solution x_k such that the associated residual vector $r_k = b - Ax_k$ satisfies $\|r_k\| < \epsilon$.

Step 0. Compute $r_0 = b - Ax_0$.

Step 1. Run k steps of the Arnoldi Algorithm 2 to generate the $(k+1) \times k$ Hessenberg matricex $\tilde{H}_k$ and the orthonormal matrix V_{k+1}, using $v_1 = r_0/\|r_0\|$.

Step 2. Find the vector y_k such that the function

$$J(y) = \left\| \|r_0\| e_1 - \tilde{H}_k y \right\|$$

is minimized over all vector $y \in \mathbb{R}^k$; $e_1 = (1, 0, \cdots, 0)^T \in \mathbb{R}^{k+1}$ by solving the $k \times k$ upper triangular system

$$\tilde{R}_k y_k = g_k,$$

where $\tilde{R}_k$ and g_k are defined as in (13).

Step 3. Compute the correction vector $z_k = V_k y_k$.

Step 4. Compute the new solution vector $x_k = x_0 + z_k$.

Step 5. Compute the norm of the new residual vector, $\|r_k\| = |\gamma_{k+1}|$, where γ_{k+1} is given by (12). If $\|r_k\| < \epsilon$, then stop and accept x_k as the approximate solution.

Step 6. Compute $r_k = b - Ax_k = V_{k+1} Q^T (\gamma_{k+1} e_{k+1})$. Set $x_0 \equiv x_k$ and $r_0 \equiv r_k$. Return to Step 1.

5.4 MATLAB code for the GMRES method

A MATLAB implementation of the GMRES method described before is presented. Again we make use of Program 2 to obtain the factorization of the matrix A generated by the Arnoldi process.

Listing 4: GMRES MATLAB implementation.

```
function [x i] = myGMRES(A,b,k,epsilon,RESTART)
% GMRES method for solving Ax = b

n = size(A,1);
x = zeros(n,1);

r = b;

for i = 1:RESTART

    [V H] = myArnoldi(A,r,k);
     V_k  = V(:,1:end-1);

    nrm_r = norm(r);

    [Q R] = qr(H); % QR factorization for upper-Hessenberg
                   % matrix H
    R = R(1:end-1,:);
    y = R\(nrm_r*Q(1,1:end-1)');
    x = x + V_k*y;

    nrm_r = nrm_r*abs(Q(1,end)); % gamma = nrm_r*abs(qq(1,end));

    if nrm_r < epsilon
        break;
    end

    r = b - A*x;

end
```