2 Nonsymmetric matrices

In this chapter we look at some iterative methods for solving $Ax = b$, where A is an "arbitrary" n by n matrix.

One should keep in mind, that in most cases preconditioning is required for the algorithms to be competitive.

We will look at the following methods more carefully: CGNR and CGNE (Conjugate gradient applied to the normal equations), Bi-Conjugate Gradient, CGS, BiCGSTAB, QMR, and GMRES — not necessarily in that order.

2.1 CGNR and CGNE

Our first attempt at solving nonsymmetric systems $Ax = b$ is to make them symmetric and use standard conjugate gradient. This can be done in two ways. First we consider the normal equations

$$A^T A x = A^T b. \tag{2.1}$$

Applying conjugate gradient to this equation results in the CGNR algorithm. Denoting by z the exact solution, the minimization property of conjugate gradient implies

$$\begin{aligned} ||z - x||^2_{A^TA} &= (z-x)^T A^T A (z-x) \\ &= (Az - Ax)^T (Az - Ax) = (b - Ax)^T (b - Ax) = ||r||^2 \end{aligned}$$

is minimized over $x_0 + \mathcal{K}_{k-1}$. The "N" in the name stands for normal equations, and the "R" stands for residual (which is minimized).

Alternatively, one can solve $Ax = b$ by applying conjugate gradient to

$$AA^T y = b, \tag{2.2}$$

and setting $x = A^T y$. The minimization property tells us now, that ($z = A^T w$ denotes the exact solution)

$$\begin{aligned} ||w - y||^2_{AA^T} &= (w-y) AA^T (w-y) \\ &= (A^T w - A^T y)^T (A^T w - A^T y) = ||z - x||^2 \end{aligned}$$

is minimized over $x_0 + \mathcal{K}_{k-1}$. Hence the name CGNE, with the "E" standing for minimizing the error.

Analysis can be carried over from the symmetric case. The eigenvalues of $A^T A$ or AA^T, respectively, are the squares of the **singular** values of A, and the performance of CGNR or CGNE therefore depends on the singular values of A.

2.2 BiCG and CGS

We use the polynomial version of the Conjugate Gradient Algorithm to generalize to a nonsymmetric system

$$Ax = b.$$

Let x_0 be the initial guess with residual $r_0 = b - Ax_0$, and let $\tilde{r}_0$ be an arbitrary vector (often chosen to be $\tilde{r}_0 = r_0$). We define the bilinear form

$$(\phi, \psi) = \tilde{r}_0^T \phi(A)\psi(A)r_0 = (\phi(A^T)\tilde{r}_0)^T \psi(A)r_0, \tag{2.3}$$

and define $p_{-1} = \tilde{p}_{-1} = 0$, as well as

$$\begin{aligned} r_k &= \phi_k(A)r_0, & \tilde{r}_k &= \phi_k(A^T)\tilde{r}_0 \\ p_k &= \psi_k(A)r_0, & \tilde{p}_k &= \psi_k(A^T)\tilde{r}_0, \end{aligned} \tag{2.4}$$

where ϕ_k and ψ_k are the "orthogonal" polynomials generated by the polynomial CG algorithm

$k = 0$; $\rho_{-1} = 1$; $\phi_0 = 1$; $\psi_{-1} = 0$;
while " $\rho_k > 0$"
  $\rho_k = (\phi_k, \phi_k)$; $\beta_k = \rho_k / \rho_{k-1}$
  $\psi_k = \phi_k + \beta_k \psi_{k-1}$
  $\sigma_k = (\psi_k, \theta\psi_k)$; $\alpha_k = \rho_k / \sigma_k$
  $\phi_{k+1} = \phi_k - \alpha_k \theta \psi_k$
  $k = k + 1$
end while

Translating this algorithm into its vector form gives the Bi-Conjugate Gradient Method

$k = 0$; $r_0 = b - Ax_0$; $\rho_{-1} = 1$;
$p_{-1} = 0$; $\tilde{p}{-1} = 0$; Choose $\tilde{r}_0$
while " $r_k \neq 0$"
  $\rho_k = \tilde{r}_k^T r_k$; $\beta_k = \rho_k / \rho_{k-1}$
  $p_k = r_k + \beta_k p_{k-1}$; $\tilde{p}_k = \tilde{r}_k + \beta_k \tilde{p}_{k-1}$;
  $\sigma_k = \tilde{p}_k^T A p_k$; $\alpha_k = \rho_k / \sigma_k$
  $r_{k+1} = r_k - \alpha_k A p_k$; $\tilde{r}_{k+1} = \tilde{r}_k - \alpha_k A^T \tilde{p}_k$
  $x_{k+1} = x_k + \alpha_k p_k$
  $k = k + 1$
end while

Note that this algorithms generates two sequences,

$$
\begin{aligned}
r_k \in \mathcal{K}_{k-1}(A, r_0) &= \mathrm{span}\{r_0, Ar_0, \ldots, A^{k-1}r_0\}, && (2.5)\\
\tilde{r}_k \in \mathcal{K}_{k-1}(A^T, \tilde{r}_0) &= \mathrm{span}\{\tilde{r}_0, A^T\tilde{r}_0, \ldots, (A^T)^{k-1}\tilde{r}_0\}, && (2.6)
\end{aligned}
$$

which are **bi-orthogonal**, i.e.,

$$r_j^T \tilde{r}_i = 0 \qquad \text{for } i \neq j. \tag{2.7}$$

Bi-CG needs approximately twice as much work in each step as CG; Furthermore, a multiplication by A^T is needed, and this could lead to cumbersome programming. Assuming that Bi-CG converges well (which it sometimes does), we can improve on this method by returning to the polynomial version. Note that the scalars ρ_k and σ_k in our algortihm are calculated by

$$\rho_k = (\phi_k, \phi_k), \qquad \sigma_k = (\psi_k, \theta\psi_k).$$

Now, because of property (1.21), and since $\phi_0 = 1$, we can rewrite this as

$$\rho_k = (\phi_0, \phi_k^2), \qquad \sigma_k = (\phi_0, \theta\psi_k^2).$$

Maybe we can translate this into an algorithm which generates the squares of the original polynomials, and interpret the vectors $\phi_k^2(A)r_0$ as residuals of **new** estimates x_k of the exact solution z. If we rename

$$r_k = \phi_k^2(A)r_0,$$

we can, for example, compute the quantity ρ_k as $\rho_k = \tilde{r}_0^T r_k$. Thus we need no longer generate the sequences $\tilde{r}_k$ and $\tilde{p}_k$.

We take the recurrences for ψ_k and ϕ_{k+1},

$$
\begin{aligned}
\psi_k &= \phi_k + \beta_k\psi_{k-1}, && (2.8)\\
\phi_{k+1} &= \phi_k - \alpha_k\theta\psi_k, && (2.9)
\end{aligned}
$$

and square them:

$$
\begin{aligned}
\psi_k^2 &= \phi_k^2 + 2\beta_k\phi_k\psi_{k-1} + \beta_k^2\psi_{k-1}^2 && (2.10)\\
\phi_{k+1}^2 &= \phi_k^2 - 2\alpha_k\theta\phi_k\psi_k + \alpha_k^2\theta^2\psi_k^2, && (2.11)
\end{aligned}
$$

with $\theta(t) = t$ as before. We must eliminate the cross terms involving ϕ and ψ.

$$
\begin{aligned}
\phi_k\psi_k &= \phi_k(\phi_k + \beta_k\psi_{k-1}) = \phi_k^2 + \beta_k\phi_k\psi_{k-1} && (2.12)\\
\phi_{k+1}\psi_k &= (\phi_k - \alpha_k\theta\psi_k) = \phi_k^2 + \beta_k\phi_k\psi_{k-1} - \alpha_k\theta\psi_k^2. && (2.13)
\end{aligned}
$$

The first equation (2.12) can be used to eliminate the $\phi_k\psi_k$ term in (2.11); the second equation (2.13) uses the recurrence for ϕ_{k+1} and (2.12); it gives us a recurrence for the cross terms of the form $\phi_k\psi_{k-1}$. We now set

$$\Phi_k = \phi_k^2, \qquad \Psi_k = \psi_k^2, \qquad \Gamma_k = \phi_k\psi_{k-1}, \tag{2.14}$$

rewriting the recurrences (2.10), (2.11) and (2.13) as

$$\begin{aligned}
\Psi_k &= \Phi_k + 2\beta_k\Gamma_k + \beta_k^2\Psi_{k-1} && (2.15)\\
&= \underbrace{\Phi_k + \beta_k\Gamma_k}_{=:\Omega_k} + \beta_k(\Gamma_k + \beta_k\Psi_{k-1}), && (2.16)\\
\Gamma_{k+1} &= \underbrace{\Phi_k + \beta_k\Gamma_k}_{=\Omega_k} - \alpha_k\theta\Psi_k && (2.17)\\
\Phi_{k+1} &= \Phi_k - 2\alpha_k\theta(\underbrace{\Phi_k + \beta_k\Gamma_k}_{=\Omega_k}) + \alpha_k^2\theta^2\Psi_k && (2.18)\\
&= \Phi_k - \alpha_k\theta(\Omega_k + \underbrace{\Omega_k - \alpha_k\theta\Psi_k}_{=\Gamma_{k+1}}) && (2.19)\\
&= \Phi_k - \alpha_k\theta(\Omega_k + \Gamma_{k+1}). && (2.20)
\end{aligned}$$

This gives us the polynomial version of the CGS algorithm

$k = 0$; $\rho_{-1} = 1$; $\Phi_0 = 1$; $\Gamma_0 = \Psi_{-1} = 0$;
while " $\rho_k > 0$"
  $\rho_k = (1, \Phi_k)$; $\beta_k = \rho_k/\rho_{k-1}$
  $\Omega_k = \Phi_k + \beta_k\Gamma_k$
  $\Psi_k = \Omega_k + \beta_k(\Gamma_k + \beta_k\Psi_{k-1})$
  $\sigma_k = (1, \theta\Psi_k)$; $\alpha_k = \rho_k/\sigma_k$
  $\Gamma_{k+1} = \Omega_k - \alpha_k\theta\Psi_k$
  $\Phi_{k+1} = \Phi_k - \alpha_k\theta(\Omega_k + \Gamma_{k+1})$
  $k = k + 1$
end while

When translating into vector form, bear in mind that the following vectors correspond to the polynomials listed:

$$q \leftrightarrow \Gamma, \qquad u \leftrightarrow \Omega, \qquad r \leftrightarrow \Phi, \qquad p \leftrightarrow \Psi, \qquad v = Ap \leftrightarrow \theta\Psi.$$

CGS algorithm

$k = 0$; $r_0 = b - Ax_0$; $\rho_{-1} = 1$;
$q_0 = p_{-1} = 0$; Choose $\tilde{r}_0$
while " $r_k \neq 0$"
  $\rho_k = \tilde{r}_0^T r_k$; $\beta_k = \rho_k / \rho_{k-1}$
  $u_k = r_k + \beta_k q_k$;
  $p_k = u_k + \beta_k (q_k + \beta_k p_{k-1})$;
  $v_k = Ap_k$;
  $\sigma_k = \tilde{r}_0^T A p_k$; $\alpha_k = \rho_k / \sigma_k$
  $q_{k+1} = u_k - \alpha_k v_k$;
  $r_{k+1} = r_k - \alpha_k A(u_k + q_{k+1})$;
  $x_{k+1} = x_k + \alpha_k (u_k + q_{k+1})$;
  $k = k + 1$
end while

As you will see in numerical experiments, the performance of CGS can be quite erratic (it amplifies both good and bad behaviour of Bi-CG). To smooth out the convergence, the following modification was proposed by Van der Vorst.

2.3 BiCGSTAB

In the previous section (§2.2) we derived the CGS algorithm by making the residual satisfy

$$r_k = \phi_k^2(A) r_0,$$

where ϕ_k is the polynomial generated by the Bi-CG algorithm. The idea in BiCGSTAB is to instead have

$$r_k = \tau_k(A)\phi_k(A) r_0,$$

with a simple choice for the τ_k:

$$\tau_k(z) = \prod_{i=0}^{k-1} (1 - \omega_i z), \tag{2.21}$$

where the ω_i's are chosen to minimize $r_k = \tau_k(A)\phi_k(A) r_0$ as a function of ω_{k-1}.

Using the recurrences (2.9) and (2.8), we now have to find recurrences for the polynomials $\tau_k(z)\phi_k(z)$ and $\tau_k(z)\psi_k(z)$, with ϕ_k and ψ_k our familiar Bi-CG polynomials.

$$\begin{aligned}
\tau_k(A)\phi_k(A)r_0 &= \left[(1-\omega_{k-1}A)\tau_{k-1}(A)(\phi_{k-1}(A)-\alpha_{k-1}(\theta\psi_{k-1})(A)\right]r_0 \\
&= \left[\tau_{k-1}(A)\phi_{k-1}(A)-\alpha_{k-1}A\tau_{k-1}(A)\phi_{k-1}(A)\right]r_0 \\
&\quad -\omega_{k-1}A\left[\tau_{k-1}(A)\phi_{k-1}(A)-\alpha_{k-1}A\tau_{k-1}(A)\phi_{k-1}(A)\right]r_0 \\
\tau_k(A)\psi_k(A)r_0 &= \tau_k(A)\left[\phi_k(A)+\beta_k\psi_{k-1}(A)\right]r_0 \\
&= \left[\tau_k(A)\phi_k(A)+\beta_k(1-\omega_{k-1}A)\tau_{k-1}(A)\psi_{k-1}(A)\right]r_0 \\
&= \left[\tau_k(A)\phi_k(A)+\beta_k\tau_{k-1}(A)\psi_{k-1}(A)\right]r_0 \\
&\quad -\left[\beta_k\omega_{k-1}A\tau_{k-1}(A)\psi_{k-1}(A)\right]r_0
\end{aligned}$$

The recurrences for the polynomials $\tau_k(z)\phi_k(z)$ and $\tau_k(z)\psi_k(z)$ are thus

$$\begin{aligned}
\tau_k\phi_k &= \tau_{k-1}\phi_{k-1}-\alpha_{k-1}z\tau_{k-1}\psi_{k-1} \\
&\quad -\omega_{k-1}z\left[\tau_{k-1}\phi_{k-1}-\alpha_{k-1}z\tau_{k-1}\psi_{k-1}\right] \qquad (2.22)\\
\tau_k\psi_k &= \tau_k\phi_k+\beta_k\tau_{k-1}\psi_{k-1}-\beta_k\omega_{k-1}z\tau_{k-1}\psi_{k-1}. \qquad (2.23)
\end{aligned}$$

Since we will be computing $r_k = \tau_k(A)\phi_k(A)r_0$, the quantities

$$\tilde{\rho}_k = \tilde{r_0}^T r_k = \tilde{r_0}^T\tau_k(A)\phi_k(A)r_0 = (\tau_k(A^T)\tilde{r_0})^T\phi_k(A)r_0 \qquad (2.24)$$

are readily available; however, our algorithm calls for the quantities

$$\rho_k = \tilde{r_0}^T\phi_k(A)\phi_k(A)r_0 = (\phi_k(A^T)\tilde{r_0})^T\phi_k(A)r_0. \qquad (2.25)$$

Note, that by construction (see (2.5) – (2.7)), our residuals r_k are orthogonal to the Krylov subspace

$$\mathcal{K}_{k-1}(A^T,\tilde{r}_0) = \mathrm{span}\{\tilde{r}_0, A^T\tilde{r}_0,\ldots,(A^T)^{k-1}\tilde{r}_0\},$$

therefore, r_k is orthogonal to any vector of the form $U_{k-1}(A^T)\tilde{r}_0$, where U_{k-1} is a polynomial of degree $\leq k-1$. To find out what $\tilde{\rho}_k$ is, we need only the highest order term of $\tau_k(A^T)$, which is

$$(-1)^k\omega_0\omega_1\ldots\omega_{k-1}(A^T)^k.$$

Similarly, for ρ_k only the highest order term of $\phi_k(A^T)$, i.e.,

$$(-1)^k\alpha_0\alpha_1\ldots\alpha_{k-1}(A^T)^k,$$

is needed. Therefore,

$$\begin{aligned}
\tilde{\rho}_k &= (\tau_k(A^T)\tilde{r_0})^T\phi_k(A)r_0 = (-1)^k\omega_0\omega_1\ldots\omega_{k-1}\left[((A^T)^k\tilde{r_0})^T\phi_k(A)r_0\right], \\
\rho_k &= (\phi_k(A^T)\tilde{r_0})^T\phi_k(A)r_0 = (-1)^k\alpha_0\alpha_1\ldots\alpha_{k-1}\left[((A^T)^k\tilde{r_0})^T\phi_k(A)r_0\right],
\end{aligned}$$

and

$$\beta_k = \frac{\rho_k}{\rho_{k-1}} = \frac{\alpha_{k-1}}{\omega_{k-1}} \frac{\tilde{\rho}_k}{\tilde{\rho}_{k-1}}. \tag{2.26}$$

We now set

$$r_k = \tau_k(A)\phi_k(A)r_0, \qquad p_{k-1} = \tau_{k-1}(A)\psi_{k-1}(A)r_0,$$

to obtain the BiCGSTAB algorithm (note, that we write ρ instead of $\tilde{\rho}$)

$k = 0$; $r_0 = b - Ax_0$; $\rho_{-1} = 1$;
$v_{-1} = p_{-1} = 0$; $\omega_{-1} = \alpha_{-1} = 1$; Choose $\tilde{r}_0$
while " $r_k \neq 0$"
  $\rho_k = \tilde{r}_0^T r_k$; $\beta_k = (\rho_k/\rho_{k-1})(\alpha_{k-1}/\omega_{k-1})$
  $p_k = r_k + \beta_k(p_{k-1} - \omega_{k-1}v_{k-1})$;
  $v_k = Ap_k$;
  $\alpha_k = \rho_k/(\tilde{r}_0^T v_k)$
  $s_k = r_k - \alpha_k v_k$; $t_k = As_k$;
  $\omega_k = (t_k^T s_k)/(t_k^T t_k)$;
  $r_{k+1} = s_k - \omega_k t_k$;
  $x_{k+1} = x_k + \alpha_k p_k + \omega_k s_k$;
  $k = k + 1$
end while

The value of ω_k is chosen to minimize $||r_{k+1}||$, by means of a simple one-dimensional orthogonal projection.

2.4 GMRES - Generalized Minimum Residual

The basic idea of GMRES can be explained in one line: Mimize the residual over the space $x_0 + \mathcal{K}_{k-1}(A, r_0)$. Since we have again a Krylov subspace method, we have

$$r_k = \phi_k(A)r_0,$$

with ϕ_k a polynomial of degree k or less, again $\phi_k(0) = 1$. Therefore,

$$\frac{||r_k||}{||r_0||} \leq ||\phi_k(A)||_2.$$

Theorem 2.1 *If A is a nonsingular n by n matrix, then GMRES will terminate in no less than n iterations with the exact solution. Furthermore, if*

A is diagonalizable, i.e., $A = V\Lambda V^{-1}$, then

$$\frac{||r_k||}{||r_0||} \leq \kappa_2(V) \max_{z\in\sigma(A)} |\phi_k(z)|$$

for any polynomial ϕ_k of degree k or less with $\phi_k(0) = 1$.

In the sequel we will restrict our attention to the case $x_0 = 0$. Step k of GMRES is equivalent to

$$\min_{x\in\mathcal{K}_{k-1}} ||b - Ax||_2.$$

We have

$$\mathcal{K}_{k-1} = \mathrm{span}\{b, Ab, \ldots, A^{k-1}b\},$$

and the n by k matrices

$$K_k = [b|Ab|\cdots|A^{k-1}b], \qquad AK_k = [Ab|A^2b|\cdots|A^kb], \tag{2.27}$$

and thus we need to solve

$$||AK_kc - b|| \to \min, \;\; c \in \mathrm{I\!R}^k; \qquad x_k = K_kc.$$

We could solve this problem via

1. normal equations $(AK_k)^T(AK_k)c = (AK_k)^Tb$,
2. QR-factorization, $AK_k = Q_kR_k$, $R_kc = Q_k^Tb$, or
3. SVD of AK_k.

In effect we will use a variant of the second approach. Note that the columns of the matrix (AK_k) are nearly linearly dependent. We will generate the vectors of the Krylov subspace $\mathcal{K}_{k-1} = \mathrm{span}\{b, Ab, \ldots, A^{k-1}b\}$, and orthogonalize simultaneously (we are not particularly interested in the matrix AK_k as such, i.e., we do not need the R_k factor). This is called the

Arnoldi process:

$q_1 = b/||b||$;
for $j = 1, 2, 3, \ldots$
  $v_j = Aq_j$
  for $i = 1$ **to** j,
    $h_{ij} = q_i^T v_j$
    $v_j = v_j - h_{ij}q_i$
  $h_{j+1,j} = ||v_j||$
  $q_{j+1} = v_j/h_{j+1,j}$

A careful look at this algorithm reveals its similarity to the modified Gram-Schmidt method for orthonormalizing vectors. The result of this algorithm can be summarized in the matrix relation

$$A \left(\begin{array}{c|c|c|c|c} | & | & & & | \\ q_1 & q_2 & \cdots & \cdots & q_k \\ | & | & & & | \end{array} \right) = \left(\begin{array}{c|c|c|c|c} | & | & & | & | \\ q_1 & \cdots & \cdots & q_k & q_{k+1} \\ | & | & & | & | \end{array} \right) \tilde{H}_k, \tag{2.28}$$

or, with

$$Q_k = \left(\begin{array}{c|c|c|c|c} | & | & & & | \\ q_1 & q_2 & \cdots & \cdots & q_k \\ | & | & & & | \end{array} \right)$$

$$AQ_k = Q_{k+1}\tilde{H}_k, \tag{2.29}$$

where $\tilde{H}_k$ is the $(k+1)$ by k Hessenberg matrix

$$\tilde{H}_k = \begin{bmatrix} h_{11} & h_{12} & \dots & & & h_{1k} \\ h_{21} & h_{22} & \dots & & & h_{2k} \\ 0 & h_{32} & h_{33} & & & h_{3k} \\ \vdots & 0 & \ddots & \ddots & & \vdots \\ & & \ddots & \ddots & & \\ & & & & h_{k,k-1} & h_{kk} \\ & & & & 0 & h_{k+1,k} \end{bmatrix} \tag{2.30}$$

We denote by H_k the first k rows of $\tilde{H}_k$, i.e., H_k is the k by k Hessenberg matrix

$$H_k = \begin{bmatrix} h_{11} & h_{12} & \dots & & & h_{1k} \\ h_{21} & h_{22} & \dots & & & h_{2k} \\ 0 & h_{32} & h_{33} & & & h_{3k} \\ \vdots & 0 & \ddots & \ddots & & \vdots \\ & & \ddots & \ddots & & \\ & & & & h_{k,k-1} & h_{kk} \end{bmatrix} \tag{2.31}$$

We observe, that $Q_k^T Q_{k+1}$ is the k by $(k+1)$ identity matrix, and therefore

$$Q_k^T Q_{k+1} \tilde{H}_k = H_k.$$

If we now multiply the relation $AQ_k = Q_{k+1}\tilde{H}_k$ by Q_k^T, we obtain

$$Q_k^T A Q_k = Q_k^T Q_{k+1} \tilde{H}_k = H_k,$$

and this matrix describes the orthogonal projection of "A onto $\mathcal{K}_{k-1}$", in the sense of projecting vectors Av onto the Krylov subspace.

Theorem 2.2 *The matrices Q_k generated by the Arnoldi process are the "QR-factors" of the Krylov matrix $K_k = Q_k R_k$. The Hessenberg matrix H_k describes the projection of "A onto $\mathcal{K}_{k-1}$",*

$$H_k = Q_k^T A Q_k,$$

and we have the relation

$$AQ_k = Q_{k+1}\tilde{H}_k.$$

Also note that $h_{jj} = q_j^T A q_j$.

We can now return to GMRES, where we minimize

$$\min_{x \in \mathcal{K}_{k-1}} ||Ax - b|| \quad \Leftrightarrow \quad \min_{c \in \mathbb{R}^k} ||AK_k c - b|| \quad \Leftrightarrow \quad \min_{y \in \mathbb{R}^k} ||AQ_k y - b||;$$

note that AQ_k is still an n by k matrix. Because of (2.29) we can now replace AQ_k by $Q_{k+1}\tilde{H}_k$, and then multiply by Q_{k+1}^T:

$$\min ||Q_{k+1}\tilde{H}_k y - b|| \quad \Leftrightarrow \quad \min ||\tilde{H}_k y - Q_{k+1}^T b||.$$

Finally, from the Arnoldi process we have $b = \beta q_1$, with $\beta = ||b||$, and since the q_j are orthogonal, we have

$$Q_{k+1}^T b = \beta e_1^{(k+1)},$$

where $e_1^{(k+1)}$ is the first standard basis vector in $\mathbb{R}^{k+1}$. The residual r_k can therefore be minimized by solving the $k+1$ by k least squares problem

$$\min ||\tilde{H}_k y - \beta e_1^{(k+1)}|| = \min ||r_k||, \qquad x_k = Q_k y. \tag{2.32}$$

GMRES:

$\beta = ||b||$; $q_1 = b/\beta$; $k = 0$; $\rho_0 = 1$;
while $(\rho_k > 0)$
  $k = k + 1$;
  $v_k = Aq_k$
  for $i = 1$ **to** k,
    $h_{ik} = q_i^T v_k$
    $v_k = v_k - h_{ik} q_i$
  $h_{k+1,k} = ||v_k||$
  $q_{k+1} = v_k / h_{k+1,k}$
  $\min ||\tilde{H}_k y - \beta e_1^{(k+1)}||$;
  $\rho_k = ||\tilde{H}_k y - \beta e_1^{(k+1)}||$;
end while
$x_k = Q_k y$;

Note, that the residual norm can be computed **without** updating the solution x_k, so x_k needs only be formed once a satisfactory residual norm has been achieved. The convergence is monotone, i.e. $||r_{k+1}|| \leq ||r_k||$, and $r_k = 0$ for some $k \leq n$. The obvious drawbacks of the algorithm are the requirement to store the matrices Q_k, and the increasing size of the linear least squares problem. One way around the first problem is to restart the Algorithm after a fixed, say m, number of iterations (this is called GMRES(m)); however, even though this may work well in practice, there is no convergence theory for this method. The second problem can be alleviated by a more efficient implementation, which we will describe now.

We use Givens rotations to update the QR factorization of $\tilde{H}_k$. The 2 by 2 Givens rotation is

$$G = \begin{pmatrix} c & -s \\ s & c \end{pmatrix}, \qquad c = \cos\theta,\ s = \sin\theta,\ \ \theta \in [-\pi, \pi].$$

Such a matrix rotates a vector which forms an angle $-\theta$ with the x-axis into one which is aligned with the x-axis.

In the higher dimensional case we apply this rotation in the (x_j, x_{j+1})-coordinate plane, i.e.,

$$G_j = \begin{array}{cc} \begin{pmatrix} 1 & & & & & & & \\ & \ddots & & & & & & \\ & & 1 & & & & & \\ & & & c & -s & & & \\ & & & s & c & & & \\ & & & & & 1 & & \\ & & & & & & \ddots & \\ & & & & & & & 1 \end{pmatrix} & \begin{array}{l} \\ \\ \\ \leftarrow j\text{-th row} \\ \\ \\ \\ \\ \end{array} \end{array} .$$

In the first step we annihilate the element h_{21} in $\tilde{H}_k$; this is achieved by choosing

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

for the matrix G_1. This gives

$$G_1 \tilde{H}_k = \begin{bmatrix} \hat{h}_{11} & \hat{h}_{12} & \dots & & \dots \\ 0 & \hat{h}_{22} & \hat{h}_{23} & & \dots \\ 0 & \hat{h}_{32} & \hat{h}_{33} & & \\ \vdots & 0 & \ddots & \ddots & \\ & & \ddots & \ddots & \end{bmatrix};$$

proceeding, by annihilating $\hat{h}_{32}$, we set

$$c_2 = \frac{\hat{h}_{22}}{\sqrt{\hat{h}_{22}^2 + \hat{h}_{32}^2}}, \qquad s_1 = -\frac{\hat{h}_{32}}{\sqrt{\hat{h}_{22}^2 + \hat{h}_{32}^2}},$$

to construct G_2. After k steps we have

$$\underbrace{G_k \dots G_2 G_1}_{=:Q^{(k)}} \tilde{H}_k = R_k,$$

an upper triangular matrix. The least-squares problem becomes now

$$||\tilde{H}_k y - \beta e_1^{(k+1)}|| = ||Q^{(k)} \tilde{H}_k y - \beta Q^{(k)} e_1^{(k+1)}|| = ||R_k y - g^{(k+1)}||. \to \min$$

This whole process can be carried out via updates; if

$$\tilde{H}_k = Q^{(k)} R_k,$$

then compute the updated $(k+1)$st column of $\tilde{H}_{k+1}$ by means of

$$\tilde{h}_{k+1} = Q^{(k)} h_{k+1},$$

and find G_{k+1} to annihilate the element $\tilde{h}_{k+2,k+1}$. Then update

$$Q^{(k+1)} = G_{k+1} Q^{(k)},$$

$$R_{k+1} = \left(\begin{array}{c|c} R_k & G_{k+1}\tilde{h}_{k+1} \end{array} \right), \qquad g^{(k+2)} = G_{k+1} \begin{pmatrix} | \\ g^{(k+1)} \\ | \\ 0 \end{pmatrix}.$$

Also note, that it is not actually necessary to compute y, as

$$\rho_k = ||r_k|| = ||R_k y - g^{(k+1)}|| = g^{(k+1)}_{k+1},$$

the last component of the vector $g^{(k+1)}$.

2.5 GMRES - reorthogonalization, algorithm

GMRES uses the Arnoldi process to find an orthonormal basis for the Krylov subspace. Such a process does not work perfectly, and a gradual loss of orthogonality may occur. To ensure good performance of the algorithm, re-orthogonalization must be employed. A reasonable test to detect loss of orthogonality, is to check whether the vector v_j after the orthogonalization but **before** normalization is small compared to the vector $v = Aq_j$; if yes, re-orthogonalization is advised. For the test we use a small constant δ, $\delta = 10^{-3}$, for example.

The complete GMRES algorithm with re-orthogonalization and update of the QR factorization with Givens rotations is as follows:

GMRES:

$r_0 = b - Ax_0$; $\beta = ||r_0||$; $q_1 = r_0/\beta$;
$k = 0$; $\rho_0 = 1$; $g = \beta(1, 0, \ldots, 0)^T$;
while $(\rho_k > 0)$
    $k = k + 1$;
    $v = Aq_k$; $v_k = v$;
    for $i = 1$ **to** k,
        $h_{ik} = q_i^T v_k$
        $v_k = v_k - h_{ik} q_i$
    $h_{k+1,k} = ||v_k||$
    Test for orthogonality, and re-orthogonalize if necessary:
    if $||v|| + \delta ||v_k|| = ||v||$
        for $i = 1$ **to** k,
            $\mu = q_i^T v_k$
            $h_{ik} = h_{ik} - \mu$
            $v_k = v_k - \mu q_i$
        $h_{k+1,k} = ||v_k||$
        end if
    $q_{k+1} = v_k / h_{k+1,k}$
    $q_{k+1} = v_k / h_{k+1,k}$
    Least squares: find residual norm,
        without computing the solution or residual:
    if $k > 1$ $h_{*,k} = Q_{k-1} h_{*,k}$;
    $\nu = \sqrt{h_{k,k}^2 + h_{k+1,k}^2}$; $c_k = h_{k,k}/\nu$; $s_k = -h_{k+1,k}/\nu$;
    $h_{k,k} = c_k h_{k,k} - s_k h_{k+1,k}$, $h_{k+1,k} = 0$;
    $g = G_k g$; $\rho_k = |g_{k+1}| = |e_{k+1}^T g|$;
end while
Take R the upper triangular k by k matrix h_{ij}; $i, j = 1, \ldots, k$
Take w_k the vector with the first k components of g, $w_k = g(1 : k)$;
Solve $Ry_k = w_k$; set $x_k = Q_k y_k$;

Observe that the linear least squares system is only solved once, at the end, and x_j is only updated then as well. The action of the matrix $Q^{(k)}$ on a vector x is computed via

$$Q^{(k)} x = G_k G_{k-1} \ldots G_1 x,$$

so the sequences c_k and s_k must be stored. Also please note that the matrices Q_k are the ones corresponding to the orthonormal basis of the Krylov subspace $\mathcal{K}_{k-1}$, whereas the matrices

$$Q^{(k)} = G_k G_{k-1} \dots G_1,$$

are the orthogonal matrices occurring in the QR-factorization of the linear least squares problem.