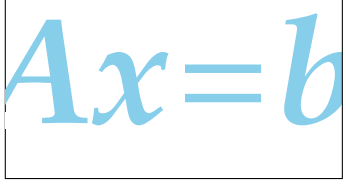
Matrix Methods for Computational Modeling and Data Analytics

Virginia Tech · Spring 2019

Mark Embree
embree@vt.edu



version of 24 April 2019

Chapter 8
Iterations for large linear systems

The circuit and truss networks we investigated in the early chapters of these notes resulted in linear systems $\mathbf{A}^T\mathbf{K}\mathbf{A}\mathbf{x} = \mathbf{f}$, where the dimension of $\mathbf{A}^T\mathbf{K}\mathbf{A}$ corresponds to the number of nodes in an electrical network, or double the number of nodes in a 2d truss. Industrial-scale circuits and trusses will yield matrices $\mathbf{A}^T\mathbf{K}\mathbf{A}$ of extremely large dimension. Moreover, since most nodes are only connected to a few other nodes, the vast majority of the entries in this matrix will be *zero* – so we call the matrix *sparse*. Figure 8.1 shows the nonzero pattern of a matrix modeling a branched neuron, along the lines of the example from Figure 2.4. This is still a small example (just 52×52), but it suggests the structure that exists in much larger matrices. In this chapter we seek modern algorithms that will allow us to solve many such large, sparse systems faster than using Gaussian elimination. As we shall see, the behavior of these methods depends significantly on properties of the coefficient matrix, sometimes in quite subtle ways. The resulting tools are among the most important techniques in modern high-performance scientific computing. A quest for a deeper understanding of these vital algorithms forms a major research frontier.

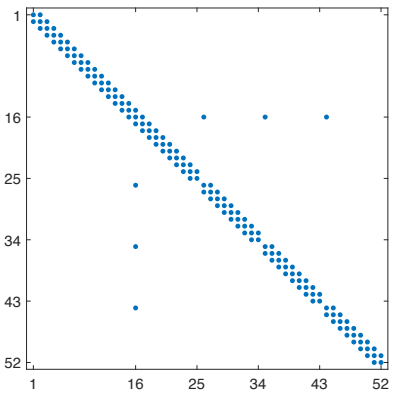


Figure 8.1: Location of nonzero entries (the *sparsity pattern*) of $\mathbf{A}^T\mathbf{K}\mathbf{A}$ for a branched neuron model of the kind illustrated in Figure 2.4, but with 16 compartments making up the left trunk (instead of 2) and four branches on the right, each comprising 8 compartments. The matrix has dimension 52×52, with only 154 of the $52^2 = 2704$ entries nonzero.

8.1 Polynomials and Krylov subspaces

Consider the general linear system of equations

$$\mathbf{A}\mathbf{x} = \mathbf{b},$$

for $\mathbf{A} \in \mathbb{R}^{n\times n}$, which we hope to solve for the unknown $\mathbf{x} \in \mathbb{R}^n$. Throughout this chapter we assume that $\mathbf{A}$ is invertible, allowing the solution to be trivially expressed as

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}.$$

We shall also assume that $\mathbf{A}$ is diagonalizable:

$$\mathbf{A} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^{-1} = \sum_{j=1}^{n} \lambda_j \mathbf{v}_j \widehat{\mathbf{v}}_j^*.$$

Here $\mathbf{v}_j$ is an eigenvector of $\mathbf{A}$ associated with the eigenvalue λ_j. (Do not confuse $\mathbf{v}_j$ with the singular vectors we have been studying in the immediately preceding chapters.) The row vector $\widehat{\mathbf{v}}_j^*$ is the jth row of the matrix $\mathbf{V}^{-1}$; it is a *left eigenvector* for λ_j, meaning

$$\mathbf{A}\mathbf{v}_j = \lambda_j \mathbf{v}_j, \qquad \widehat{\mathbf{v}}_j^* \mathbf{A} = \lambda_j \widehat{\mathbf{v}}_j^*.$$

Since $\mathbf{V}^{-1}\mathbf{V} = \mathbf{I}$, we have the *bi-orthogonality* relationship

$$\widehat{\mathbf{v}}_j^* \mathbf{v}_j = 1, \qquad \widehat{\mathbf{v}}_j^* \mathbf{v}_k = 0 \quad (j \neq k).$$

(We use the conjugate-transpose * here, since λ_j and $\mathbf{v}_j$, $\widehat{\mathbf{v}}_j$ can have complex entries, even when $\mathbf{A} \in \mathbb{R}^{n\times n}$.)

Note that the inverse of $\mathbf{A}$ has a very simple formula, in terms of the diagonalization:

$$\mathbf{A}^{-1} = \mathbf{V}\mathbf{\Lambda}^{-1}\mathbf{V}^{-1} = \sum_{j=1}^{n} \frac{1}{\lambda_j} \mathbf{v}_j \widehat{\mathbf{v}}_j^*. \tag{8.1}$$

We would never directly compute $\mathbf{A}^{-1}$ for a large matrix – that would require far too much work (and, in many cases, a prohibitive amount of storage). Instead, polynomials will provide a very convenient way to *approximate* the inverse. First square the diagonalization above to obtain:

$$\mathbf{A}^2 = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^{-1}\mathbf{V}\mathbf{\Lambda}\mathbf{V}^{-1} = \mathbf{V}\mathbf{\Lambda}^2\mathbf{V}^{-1} = \sum_{j=1}^{n} \lambda_j^2 \mathbf{v}_j \widehat{\mathbf{v}}_j^*.$$

In many applications, a large $n \times n$ matrix will only have $\mathcal{O}(n)$ nonzero entries, but the entries in $\mathbf{A}^{-1}$ will have $\mathcal{O}(n^2)$ nonzero entries. For example, $n = 10^6$, you would need about 7 *terabytes* to simply store $\mathbf{A}$ as a dense matrix.

Similarly, for any integer power $p = 0, 1, 2, \ldots$,

$$\mathbf{A}^p = \mathbf{V}\mathbf{\Lambda}^p\mathbf{V}^{-1} = \sum_{j=1}^{n} \lambda_j^p \mathbf{v}_j \widehat{\mathbf{v}}_j^*. \tag{8.2}$$

Now consider the degree $k-1$ polynomial

$$q(z) = c_1 + c_2 z + c_3 z^2 + \cdots + c_k z^{k-1}.$$

To evaluation $q(\mathbf{A})$, simply substitute $\mathbf{A}^p$ for each occurrence of z^p. Using the diagonalization form of $\mathbf{A}^p$ in (8.2), we can write

$$\begin{aligned} q(\mathbf{A}) &= c_1\mathbf{I} + c_2\mathbf{A} + c_3\mathbf{A}^2 + \cdots + c_k\mathbf{A}^{k-1} \\ &= c_1 \sum_{j=1}^{n} \mathbf{v}_j \widehat{\mathbf{v}}_j^* + c_2 \sum_{j=1}^{n} \lambda_j \mathbf{v}_j \widehat{\mathbf{v}}_j^* + c_3 \sum_{j=1}^{n} \lambda_j^2 \mathbf{v}_j \widehat{\mathbf{v}}_j^* \cdots + c_k \sum_{j=1}^{n} \lambda_j^{k-1} \mathbf{v}_j \widehat{\mathbf{v}}_j^* \\ &= \sum_{j=1}^{n} q(\lambda_j) \mathbf{v}_j \widehat{\mathbf{v}}_j^*. \end{aligned}$$

Now compare this last equation to (8.1). If

$$q(\lambda_j) = \frac{1}{\lambda_j}, \qquad j = 1, \ldots, n,$$

then the formulas for $q(\mathbf{A})$ and $\mathbf{A}^{-1}$ match, and

$$q(\mathbf{A}) = \mathbf{A}^{-1}.$$

Think this idea through for a few small values of n. For $n = 2$ we require

$$q(\lambda_1) = \frac{1}{\lambda_1}, \qquad q(\lambda_2) = \frac{1}{\lambda_2}, \tag{8.3}$$

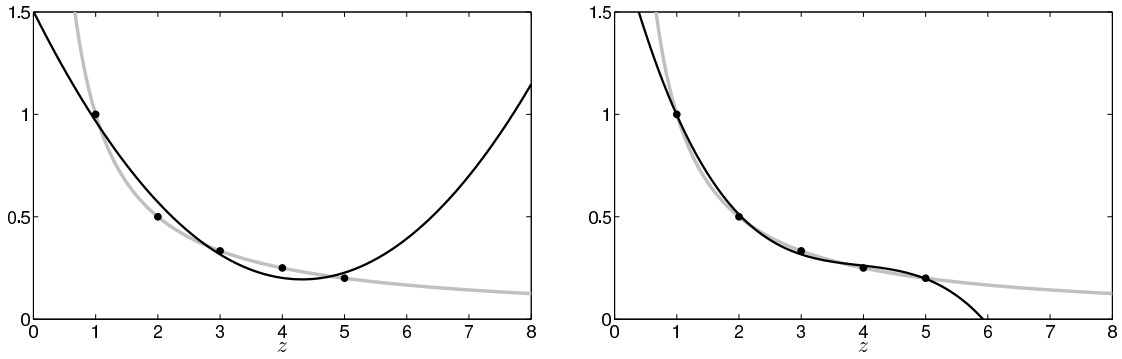


Figure 8.2: Approximations to $1/z$ (gray) at eigenvalues (dots) $1, \ldots, 5$ by quadratic (black line, left plot) and cubic (black line, right plot) polynomials $q(z)$.

which can be satisfied by a *linear* polynomial $q(z) = c_1 + c_2 z$ where the two equations in (8.3) specify the two free parameters c_1 and c_2 in the equation for the line. If $n = 3$, we require

$$q(\lambda_1) = \frac{1}{\lambda_1}, \qquad q(\lambda_2) = \frac{1}{\lambda_2}, \qquad q(\lambda_3) = \frac{1}{\lambda_3},$$

which can be satisfied by a *quadratic* polynomial $q(z) = c_1 + c_2 z + c_3 z^2$: now we need three free parameters c_1, c_2, and c_3 in the equation for the quadratic. The pattern generalizes: n distinct eigenvalues impose n constraints $q(\lambda_j) = 1/\lambda_j$, which can be satisfied by appropriately setting the n parameters $c_0, \ldots, c_{n-1}$ in a degree $n-1$ polynomial q. Hence we can write $\mathbf{A}^{-1} = q(\mathbf{A})$ for some degree $n-1$ polynomial: a fact known as the *Cayley–Hamilton Theorem*.

Step back for a moment. Our ultimate goal is to approximate $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$ as efficiently as possible. While q might need to have degree $n-1$ to give an exact match, $q(\mathbf{A}) = \mathbf{A}^{-1}$, in many cases we might be able to construct a *low-degree polynomial* q that only gives $q(\mathbf{A}) \approx \mathbf{A}^{-1}$, in the sense that

$$q(\lambda_j) \approx \frac{1}{\lambda_j}, \qquad j = 1, \ldots, n.$$

Two such polynomials are shown in in Figure 8.2, where the eigenvalues of $\mathbf{A}$ are $\{1, \ldots, 5\}$ and q has degree 2 and 3. Even for these low-degree polynomials, the approximations are quite good.

There is yet one more refinement to make. Since we really care about $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$, consider the formulations

$$\mathbf{A}^{-1}\mathbf{b} = \sum_{j=1}^{n} \frac{1}{\lambda_j} \mathbf{v}_j (\widehat{\mathbf{v}}_j^* \mathbf{b})$$

$$q(\mathbf{A})\mathbf{b} = \sum_{j=1}^{n} q(\lambda_j) \mathbf{v}_j (\widehat{\mathbf{v}}_j^* \mathbf{b}).$$

The right-hand side $\mathbf{b}$ influences these formulas, in the sense that the coefficients $\widehat{\mathbf{v}}_j^* \mathbf{b}$ can be viewed as *weights* that reveal how important

it is for $q(\lambda_j) \approx 1/\lambda_j$. At the extreme, $\widehat{\mathbf{v}}_j^* \mathbf{b} = 0$ indicates that $\mathbf{b}$ has no component in the $\mathbf{v}_j$ eigenvector direction, and so λ_j does not play a role in the problem, and can be ignored. We want to exploit these weights when we design approximating polynomials q that give

$$q(\mathbf{A})\mathbf{b} \approx \mathbf{A}^{-1}\mathbf{b}.$$

The set of all such approximations $q(\mathbf{A})\mathbf{b}$, where q can have degree up to $k-1$, has a special name.

Definition 25 *For $k \geq 1$, the set of vectors*

$$\mathcal{K}_k(\mathbf{A}, \mathbf{b}) = \{q(\mathbf{A})\mathbf{b} : \deg(q) < k\}$$

is called the kth Krylov subspace *generated by* $\mathbf{A}$ *and* $\mathbf{b}$. *It can be equivalently expressed as*

$$\mathcal{K}_k(\mathbf{A}, \mathbf{b}) = \operatorname{span}\{\mathbf{b}, \mathbf{A}\mathbf{b}, \mathbf{A}^2\mathbf{b}, \ldots, \mathbf{A}^{k-1}\mathbf{b}\}.$$

The subspace is named for Aleksey Krylov, a Russian naval architect.

Our goal should be to extract from $\mathcal{K}_k(\mathbf{A}, \mathbf{b})$ the best approximation to $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$.

8.2 *Minimum residual methods*

The best approximation $\mathbf{x}_k$ to $\mathbf{x}$ from $\mathcal{K}_k(\mathbf{A}, \mathbf{b})$ would satisfy

$$\|\mathbf{x} - \mathbf{x}_k\| = \min_{\widehat{\mathbf{x}} \in \mathcal{K}_k(\mathbf{A}, \mathbf{b})} \|\mathbf{x} - \widehat{\mathbf{x}}\|. \tag{8.4}$$

While it would be very appealing to find this optimal vector $\mathbf{x}_k$, we should temper our ambition: To compute the best approximation to $\mathbf{x}$, we would need to know $\mathbf{x}$, the true solution. If we knew $\mathbf{x}$, we would be finished! The next best thing to solving (8.4) is to instead minimize the misfit between $\mathbf{b}$ and $\mathbf{A}\mathbf{x}_k$. This misfit is the *residual* $\mathbf{b} - \mathbf{A}\mathbf{x}_k$. Minimizing the norm of this vector gives

$$\|\mathbf{b} - \mathbf{A}\mathbf{x}_k\| = \min_{\widehat{\mathbf{x}} \in \mathcal{K}_k(\mathbf{A}, \mathbf{b})} \|\mathbf{b} - \mathbf{A}\widehat{\mathbf{x}}\|. \tag{8.5}$$

Since we know $\mathbf{b}$ and $\mathbf{A}$, we can access the residual without knowing the solution $\mathbf{x}$. We shall see that we can solve (8.5) by formulating the minimization as the kind of least squares problem we addressed in Section 4.10 and Chapter 7. To do so, define the *Krylov matrix*

$$\mathbf{K}_k = \begin{bmatrix} \mathbf{b} & \mathbf{A}\mathbf{b} & \cdots & \mathbf{A}^{k-1}\mathbf{b} \end{bmatrix} \in \mathbb{R}^{n \times k},$$

and note that

$$\begin{aligned} \mathcal{R}(\mathbf{K}_k) &= \operatorname{span}\{\mathbf{b}, \mathbf{A}\mathbf{b}, \mathbf{A}^2\mathbf{b}, \ldots, \mathbf{A}^{k-1}\mathbf{b}\} \\ &= \mathcal{K}_k(\mathbf{A}, \mathbf{b}). \end{aligned}$$

Any vector in $\mathcal{K}_k(\mathbf{A}, \mathbf{b})$ can thus be written as $\mathbf{K}_k\mathbf{c}$ for some $\mathbf{c} \in \mathbb{R}^k$. The minimization problem thus becomes

$$\min_{\widehat{\mathbf{x}} \in \mathcal{K}_k(\mathbf{A},\mathbf{b})} \|\mathbf{b} - \mathbf{A}\widehat{\mathbf{x}}\| = \min_{\mathbf{c} \in \mathbb{R}^k} \|\mathbf{b} - \mathbf{A}\mathbf{K}_k\mathbf{c}\|, \tag{8.6}$$

where the problem on the right is a simple least squares problem involving the matrix $\mathbf{A}\mathbf{K}_k$. As discussed in Section 4.10, the optimal $\mathbf{c}$ can thus be found as the solution of the linear system

$$(\mathbf{A}\mathbf{K}_k)^T(\mathbf{A}\mathbf{K}_k)\mathbf{c} = (\mathbf{A}\mathbf{K}_k)^T\mathbf{b},$$

where $(\mathbf{A}\mathbf{K}_k)^T(\mathbf{A}\mathbf{K}_k)$ is a $k \times k$ matrix. In general, we will have $k \ll n$, and so this system will be much smaller than the original n-dimensional system $\mathbf{A}\mathbf{x} = \mathbf{b}$. The residual-minimizing approximation $\mathbf{x}_k$ can then be expressed as

In practice, we might have $n = 10^6$ and $k = 50$ or 100.

$$\mathbf{x}_k = \mathbf{K}_k\mathbf{c} = \mathbf{K}_k\big((\mathbf{A}\mathbf{K}_k)^T(\mathbf{A}\mathbf{K}_k)\big)^{-1}(\mathbf{A}\mathbf{K}_k)^T\mathbf{b}. \tag{8.7}$$

As we increase k, the subspace $\mathcal{K}_k(\mathbf{A}, \mathbf{b})$ from which we extract the optimal approximation $\mathbf{x}_k$ grows ever larger, containing the previous subspaces:

$$\mathcal{K}_1(\mathbf{A}, \mathbf{b}) \subseteq \mathcal{K}_2(\mathbf{A}, \mathbf{b}) \subseteq \mathcal{K}_3(\mathbf{A}, \mathbf{b}) \subseteq \cdots.$$

The nested structure of these subspaces ensures that the quality of our approximation never gets worse as k increases.

Theorem 16 *The iterates $\mathbf{x}_k$ produced by the minimum residual method (8.7) produce residuals $\mathbf{r}_k = \mathbf{b} - \mathbf{A}\mathbf{x}_k$ that are monotonically decreasing in norm:*

$$\|\mathbf{r}_{k+1}\| \leq \|\mathbf{r}_k\| \leq \cdots \leq \|\mathbf{r}_1\| \leq \|\mathbf{r}_0\| = \|\mathbf{b}\|.$$

In theory, the formula (8.7) will deliver a fine approximation: the mathematics ensures that $\mathbf{x}_k$ in (8.7) is the unique residual-minimizing approximation from the kth Krylov subspace. There is one vital but subtle wrinkle. The columns of

$$\mathbf{A}\mathbf{K}_k = [\,\mathbf{A}\mathbf{b} \quad \mathbf{A}^2\mathbf{b} \quad \cdots \mathbf{A}^k\mathbf{b}\,]$$

are exactly the iterates of the *power method* for computing the eigenvector of $\mathbf{A}$ associated with the largest magnitude eigenvalue. Using the formula (8.2) for the diagonalization of matrix powers, these columns have the form

$$\mathbf{A}^p\mathbf{b} = \sum_{j=1}^{n} \lambda_j^p \mathbf{v}_j(\widehat{\mathbf{v}}_j^*\mathbf{b}),$$

which will increasingly align with the eigenvector $\mathbf{v}_\ell$ associated with the eigenvalue λ_ℓ having largest magnitude, $|\lambda_\ell| > |\lambda_j|$ for $j \neq \ell$.

This observation has nasty implications for any computer code, like the one in Figure **??**), that seeks to find $\mathbf{x}_k$ using the least squares process described above. When the columns of $\mathbf{AK}_k$ are nearly aligned with each other, the singular values of $\mathbf{AK}_k$ will decay rapidly, making the least squares formulation in (8.7) disastrously ill-posed.

Based on your experience in Chapter 7, you might think to instead use *regularization* to tackle the least-squares problem in (8.12). That is not a bad instinct, but it turns out there is a better way to formulate (8.12) that mostly avoids this ill-posedness.

```
tol = 1e-10;        % convergence tolerance
maxit = 20;         % max number of iterations
k  = 1;
rk = b;
K  = [b];
while (norm(rk)/norm(b) < tol) && (k <= maxit)
   AK = A*K;
   c = AK\b;        % solve least squares problem
   xk = K*c;        % construct iterate x_k
   rk = b - A*xk;   % compute residual vector
   K = [AK b];      % update basis for K_{k+1}(A,b)
   k = k+1;         % increment index
end
```

Figure 8.3: A naive implementation of the minimum residual method (8.7) in MATLAB. *This algorithm is badly unstable and should never be used in practice!*

To appreciate the peril of this procedure, consider

$$\mathbf{A} = \mathrm{diag}(1, 1.1, 1.2, \ldots, 10), \tag{8.8}$$

a diagonal matrix of dimension $n = 91$. The result of running the algorithm in Figure 8.3 on $\mathbf{A}$ with $\mathbf{b} = [1, 1, \ldots, 1]^T$ is shown in Figure 8.4. For the first dozen iterations, convergence proceeds in a steady fashion: the norm of the kth residual gets steadily smaller. At iteration $k = 13$, however, something goes terribly wrong, for $\|\mathbf{r}_{13}\| \gg \|\mathbf{r}_{12}\|$, in violation of the monotonic convergence guaranteed by Theorem 16. This performance is typical of the algorithm given in Figure 8.3, which should never be used for practical computations.

Of course, since $\mathbf{A}$ is diagonal we could immediately write down the answer to $\mathbf{Ax} = \mathbf{b}$ by inspection: $x_j = b_j/a_{j,j}$. Since iterative methods do not take any advantage of this diagonal structure, and it is easy to see the eigenvalues of such matrices, they make convenient examples.

At fault is the poor basis $\{\mathbf{b}, \mathbf{Ab}, \ldots, \mathbf{A}^{k-1}\mathbf{b}\}$ used for $\mathbf{K}_k(\mathbf{A}, \mathbf{b})$. (In fact, MATLAB often issues a warning message when trying to solve the least squares problem in the line `c = AK\b`, suggesting that something might be wrong with our algorithm.) The fast initial convergence seen in Figure 8.4 suggests that if we can fix this problem with the basis, we could have a very powerful method.

The method in (8.7) was first proposed by I. M. Khabaza in 1963, and received no attention in the scientific literature at the time. Khabaza imagined that one would restrict k to a small number, which would avoid the worst of the instabilities seen here.

8.3 Arnoldi process

From our studies in Chapter 5, you might already have a good idea for addressing the problem: construct an *orthonormal basis* for

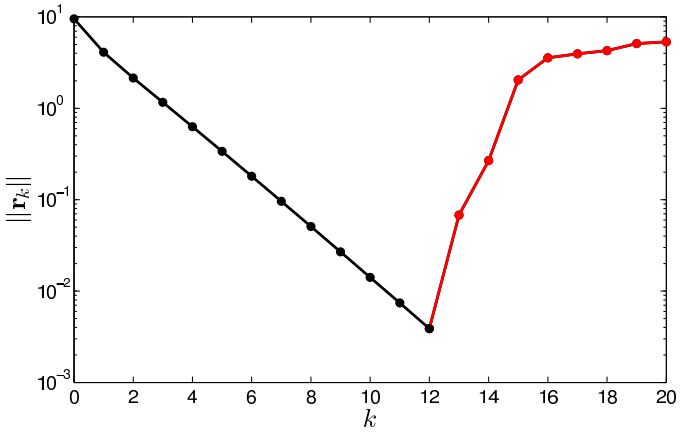


Figure 8.4: Convergence history for the bad implementation of the minimum residual method given in (8.3) applied to the matrix $\mathbf{A}$ in (8.8). The portion of the convergence curve shown in red must be a computational error, since it violates the monotonicity of the residual norms given in Theorem 16.

$\mathcal{K}_k(\mathbf{A}, \mathbf{b})$. This is a good idea, but it turns out that much has already gone wrong simply in constructing the basis vectors $\{\mathbf{b}, \mathbf{Ab}, \ldots, \mathbf{A}^{k-1}\mathbf{b}\}$ to which we would apply the Gram–Schmidt method: the errors are already rampant. A more clever – and subtle – approach is needed.

The first two steps of the refined procedure will look much like the first two steps of the Gram–Schmidt process applied to $\mathbf{b}$ and $\mathbf{Ab}$. (They are two steps of Gram–Schmidt applied to $\mathbf{b}$ and $\mathbf{Ab}/\|\mathbf{b}\|$.)

1. Define $\mathbf{u}_1 := \mathbf{b}/\|\mathbf{b}\|$.
2. Define $\widehat{\mathbf{u}}_2 := \mathbf{Au}_1 - \mathbf{u}_1\mathbf{u}_1^T(\mathbf{Au}_1) = (\mathbf{I} - \mathbf{u}_1\mathbf{u}_1^T)(\mathbf{Au}_1)$.
   Normalize: $\mathbf{u}_2 := \widehat{\mathbf{u}}_2/\|\widehat{\mathbf{u}}_2\|$.

Notice that $\mathbf{u}_1$ and $\mathbf{u}_2$ are orthonormal vectors with

$$\text{span}\{\mathbf{u}_1, \mathbf{u}_2\} = \text{span}\{\mathbf{b}, \mathbf{Au}_1\} = \text{span}\{\mathbf{b}, \mathbf{Ab}\} = \mathcal{K}_2(\mathbf{A}, \mathbf{b}).$$

The next step is the pivotal one:

problematic approach: orthogonalize $\mathbf{A}^2\mathbf{b}$ against $\mathbf{u}_1$ and $\mathbf{u}_2$;

more stable approach: orthogonalize $\mathbf{Au}_2$ against $\mathbf{u}_1$ and $\mathbf{u}_2$.

These two different choices span the same space, $\mathcal{K}_3(\mathbf{A}, \mathbf{b})$, since

$$\begin{aligned}\mathbf{Au}_2 = \frac{\mathbf{A}\widehat{\mathbf{u}}_2}{\|\widehat{\mathbf{u}}_2\|} = \frac{\mathbf{A}(\mathbf{I} - \mathbf{u}_1\mathbf{u}_1^T)\mathbf{Au}_1}{\|\widehat{\mathbf{u}}_2\|} &= e\frac{\mathbf{A}^2\mathbf{u}_1 - (\mathbf{u}_1^T\mathbf{Au}_1)\mathbf{Au}_1}{\|\widehat{\mathbf{u}}_2\|} \\ &= \frac{1}{\|\mathbf{b}\|\,\|\widehat{\mathbf{u}}_2\|}\mathbf{A}^2\mathbf{b} - \frac{\mathbf{u}_1^*\mathbf{Au}_1}{\|\mathbf{b}\|\,\|\widehat{\mathbf{u}}_2\|}\mathbf{Ab} \\ &\in \text{span}\{\mathbf{Ab}, \mathbf{A}^2\mathbf{b}\}.\end{aligned}$$

However, we will usually find that $\mathbf{A}^2\mathbf{b}$ forms a smaller angle with $\mathcal{K}_2(\mathbf{A}, \mathbf{b})$ than does $\mathbf{Au}_2$:

$$\angle(\mathbf{A}^2\mathbf{b}, \mathcal{K}_2(\mathbf{A}, \mathbf{b})) \ll \angle(\mathbf{Au}_2, \mathcal{K}_2(\mathbf{A}, \mathbf{b})).$$

Perhaps a small example will help illustrate this point. Let

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 \\ 1/4 & 1/4 & 0 \\ 0 & 1/2 & 1/2 \end{bmatrix}, \qquad \mathbf{b} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}.$$

Then

$$\mathbf{u}_1 = \frac{\mathbf{b}}{\|\mathbf{b}\|} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix},$$

and

$$\widehat{\mathbf{u}_2} = \begin{bmatrix} 0 \\ 1/4 \\ 0 \end{bmatrix}, \qquad \mathbf{u}_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}.$$

Now at the third step, we must orthogonalize either $\mathbf{A}^2\mathbf{b}$ or $\mathbf{A}\mathbf{u}_2$ against $\mathbf{u}_1$ and $\mathbf{u}_2$. Note that

$$\mathbf{A}^2\mathbf{b} = \begin{bmatrix} 1 \\ 5/16 \\ 1/8 \end{bmatrix}, \qquad \mathbf{A}\mathbf{u}_2 = \begin{bmatrix} 0 \\ 1/4 \\ 1/2 \end{bmatrix}.$$

These two vectors are shown in Figure 8.5, where it is evident that $\mathbf{A}^2\mathbf{b}$ forms a small angle with $\mathcal{K}_2(\mathbf{A}, \mathbf{b}) = \operatorname{span}\{\mathbf{u}_1, \mathbf{u}_2\}$, while $\mathbf{A}\mathbf{u}_2$ makes a much larger angle with this space. When $\mathbf{A}^2\mathbf{b}$ is orthogonalized against $\mathcal{K}_2(\mathbf{A}, \mathbf{b})$, one obtains

$$\widehat{\mathbf{u}}_3 = \begin{bmatrix} 0 \\ 0 \\ 1/8 \end{bmatrix}, \qquad \mathbf{u}_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix},$$

while when $\mathbf{A}\mathbf{u}_2$ is orthogonalized against $\mathcal{K}_2(\mathbf{A}, \mathbf{b})$, we have

$$\widehat{\mathbf{u}}_3 = \begin{bmatrix} 0 \\ 0 \\ 1/2 \end{bmatrix}, \qquad \mathbf{u}_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}.$$

Both procedures result (in exact arithmetic) in the same vector $\mathbf{u}_3$, but when we subtracted the $\mathbf{u}_1$ and $\mathbf{u}_2$ components from $\mathbf{A}^2\mathbf{b}$, a much smaller $\widehat{\mathbf{u}}_3$ vector was left over than when we removed the same components from $\mathbf{A}\mathbf{u}_2$. In both cases MATLAB will make small rounding errors when computing $\widehat{\mathbf{u}}_3$; when $\widehat{\mathbf{u}}_3$ has small entries, those rounding errors will be more influential than they will be when $\widehat{\mathbf{u}}_3$ is big. Hence, we often get a much more robust procedure by orthogonalizing $\mathbf{A}\mathbf{u}_k$ against $\mathcal{K}_k(\mathbf{A}, \mathbf{b})$ to get $\mathbf{u}_{k+1}$, rather than $\mathbf{A}^k\mathbf{b}$.

Small changes to a vector with small entries can change its direction substantially; small changes to a vector with large entries will not change its direction very much.

8.3.1 The Arnoldi algorithm

Given the preceding justification, we now develop an algorithm that will build an orthonormal basis for $\mathcal{K}_k(\mathbf{A}, \mathbf{b})$ by successively orthogonalizing $\mathbf{A}\mathbf{u}_j$ against $\mathbf{u}_1, \ldots, \mathbf{u}_j$ for $j = 1, \ldots, k$. At its most basic level, the algorithm is just an instance of the Gram–Schmidt procedure:

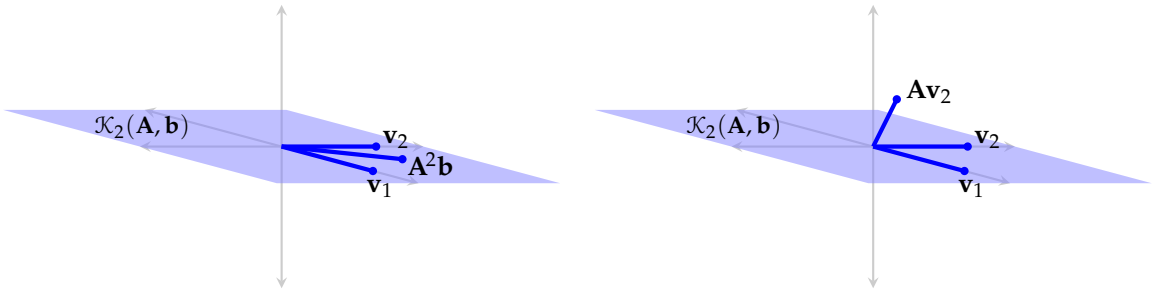


Figure 8.5: Constructing the third basis vector of a Krylov subspace: on the left, $\mathbf{A}^2\mathbf{b}$ forms a small angle with $\mathcal{K}_2(\mathbf{A},\mathbf{b})$, while on the right, $\mathbf{A}\mathbf{u}_2$ forms a larger angle with $\mathcal{K}_2(\mathbf{A},\mathbf{b})$. In general, orthogonalizing $\mathbf{A}\mathbf{u}_k$ against $\mathcal{K}_k(\mathbf{A},\mathbf{b})$ is more computationally robust than orthogonalizing $\mathbf{A}^k\mathbf{b}$.

$$
\begin{aligned}
\mathbf{u}_1 &:= \mathbf{b}/\|\mathbf{b}\| \\
\widehat{\mathbf{u}}_2 &:= (\mathbf{I} - \mathbf{u}_1\mathbf{u}_1^T)\mathbf{A}\mathbf{u}_1 \\
&= \mathbf{A}\mathbf{u}_1 - (\mathbf{u}_1^T\mathbf{A}\mathbf{u}_1)\mathbf{u}_1 \\
\mathbf{u}_2 &:= \widehat{\mathbf{u}}_2/\|\widehat{\mathbf{u}}_2\| \\
\widehat{\mathbf{u}}_3 &:= (\mathbf{I} - \mathbf{u}_1\mathbf{u}_1^T - \mathbf{u}_2\mathbf{u}_2^T)\mathbf{A}\mathbf{u}_2 \\
&= \mathbf{A}\mathbf{u}_2 - (\mathbf{u}_1^T\mathbf{A}\mathbf{u}_2)\mathbf{u}_1 - (\mathbf{u}_2^T\mathbf{A}\mathbf{u}_2)\mathbf{u}_2 \\
\mathbf{u}_3 &:= \widehat{\mathbf{u}}_3/\|\widehat{\mathbf{u}}_3\| \\
\widehat{\mathbf{u}}_4 &:= (\mathbf{I} - \mathbf{u}_1\mathbf{u}_1^T - \mathbf{u}_2\mathbf{u}_2^T - \mathbf{u}_3\mathbf{u}_3^T)\mathbf{A}\mathbf{u}_3 \\
&= \mathbf{A}\mathbf{u}_3 - (\mathbf{u}_1^T\mathbf{A}\mathbf{u}_3)\mathbf{u}_1 - (\mathbf{u}_2^T\mathbf{A}\mathbf{u}_3)\mathbf{u}_2 - (\mathbf{u}_3^T\mathbf{A}\mathbf{u}_3)\mathbf{u}_3 \\
\mathbf{u}_4 &:= \widehat{\mathbf{u}}_4/\|\widehat{\mathbf{u}}_4\| \\
&\vdots
\end{aligned}
$$

Notice that each new vector $\mathbf{u}_{\ell+1}$ takes more work to construct than did the one before it: just as in the Gram–Schmidt process, with each new basis vector we get one more direction we must orthogonalize future vectors against. As ℓ grows, this process gets quite expensive.

In Chapter 5 we found it convenient to organize the Gram–Schmidt process as a QR factorization of the matrix whose columns were the basis vectors that we orthogonalized. We would like to do the same thing now, but since the vectors we orthogonalize, $\mathbf{b}$, $\mathbf{A}\mathbf{u}_1$, $\mathbf{A}\mathbf{u}_2$, …, $\mathbf{A}\mathbf{u}_k$, involve the orthogonalized basis vectors $\mathbf{u}_1, \ldots, \mathbf{u}_k$, a special structure emerges. Label the Gram–Schmidt coefficients as

$$h_{j,\ell} := \mathbf{u}_j^T\mathbf{A}\mathbf{u}_\ell, \qquad 1 \le j \le \ell \le k,$$

with the normalizing factors given by

$$h_{\ell+1,\ell} := \|\widehat{\mathbf{u}}_\ell\|, \qquad 1 \le \ell \le k.$$

Thus a generic step of the Gram–Schmidt process becomes

$$h_{\ell+1,\ell}\mathbf{u}_{\ell+1} = \mathbf{A}\mathbf{u}_\ell - h_{1,\ell}\mathbf{u}_1 - h_{2,\ell}\mathbf{u}_2 - \cdots - h_{\ell,\ell}\mathbf{u}_\ell$$

for $\ell = 1, \ldots, k$, which we immediately rearrange to the form

$$\mathbf{A}\mathbf{u}_\ell = h_{1,\ell}\mathbf{u}_1 + h_{2,\ell}\mathbf{u}_2 + \cdots + h_{\ell,\ell}\mathbf{u}_\ell + h_{\ell+1,\ell}\mathbf{u}_{\ell+1}. \tag{8.9}$$

Now we will obtain a QR-like factorization by collecting equation (8.9) for $\ell = 1, \ldots, k$. First write (8.9) as

$$\begin{aligned}
\mathbf{A}\mathbf{u}_\ell &= \begin{bmatrix} \mathbf{u}_1 & \cdots & \mathbf{u}_\ell & \mathbf{u}_{\ell+1} \end{bmatrix} \begin{bmatrix} h_{1,\ell} \\ \vdots \\ h_{\ell,\ell} \\ h_{\ell+1,\ell} \end{bmatrix} \\
&= \begin{bmatrix} \mathbf{u}_1 & \cdots & \mathbf{u}_\ell & \mathbf{u}_{\ell+1} & \mathbf{u}_{\ell+2} & \cdots & \mathbf{u}_{k+1} \end{bmatrix} \begin{bmatrix} h_{1,\ell} \\ \vdots \\ h_{\ell,\ell} \\ h_{\ell+1,\ell} \\ 0 \\ \vdots \\ 0 \end{bmatrix}.
\end{aligned}$$

Stacking these equations side-by-side for $\ell = 1, \ldots, k$ gives

$$\mathbf{A}\begin{bmatrix} \mathbf{u}_1 & \cdots & \mathbf{u}_k \end{bmatrix} = \begin{bmatrix} \mathbf{u}_1 & \mathbf{u}_2 & \cdots & \mathbf{u}_k & \mathbf{u}_{k+1} \end{bmatrix} \begin{bmatrix} h_{1,1} & h_{1,2} & \cdots & h_{1,k} \\ h_{2,1} & h_{2,2} & \cdots & h_{2,k} \\ & \ddots & & \vdots \\ & & h_{k,k-1} & h_{k,k} \\ & & & h_{k+1,k} \end{bmatrix},$$

which we summarize as

$$\mathbf{A}\mathbf{U}_k = \mathbf{U}_{k+1}\widetilde{\mathbf{H}}_k, \tag{8.10}$$

where the matrix

$$\widetilde{\mathbf{H}}_k = \begin{bmatrix} h_{1,1} & h_{1,2} & \cdots & h_{1,k} \\ h_{2,1} & h_{2,2} & \cdots & h_{2,k} \\ & \ddots & & \vdots \\ & & h_{k,k-1} & h_{k,k} \\ & & & h_{k+1,k} \end{bmatrix} \in \mathbb{R}^{(k+1)\times k}$$

is zero in all entries *below the first subdiagonal*. (This structure is called *upper Hessenberg*.) Since the columns of $\mathbf{U}_k \in \mathbb{R}^{n\times k}$ and $\mathbf{U}_{k+1} \in \mathbb{R}^{n\times(k+1)}$ are orthonormal, we have

$$\mathbf{U}_k^T\mathbf{A}\mathbf{U}_k = \mathbf{H}_k, \tag{8.11}$$

where $\mathbf{H}_k$ is the square upper Hessenberg matrix

$$\mathbf{H}_k = \begin{bmatrix} h_{1,1} & h_{1,2} & \cdots & h_{1,k} \\ h_{2,1} & h_{2,2} & \cdots & h_{2,k} \\ & \ddots & & \vdots \\ & & h_{k,k-1} & h_{k,k} \end{bmatrix} \in \mathbb{R}^{k\times k}.$$

8.3.2 *Lucky breakdown*

We must address one subtle point: the Arnoldi process constructs $\mathbf{u}_{k+1}$ by normalizing $\widehat{\mathbf{u}}_{k+1}$,

$$\mathbf{u}_{k+1} = \frac{1}{\|\widehat{\mathbf{u}}_{k+1}\|}\widehat{\mathbf{u}}_{k+1} = \frac{1}{h_{k+1,k}}\widehat{\mathbf{v}}_{k+1}.$$

Is there a chance of dividing by zero here?

8.4 *GMRES*

We are ready to develop a robust alternative to the algorithm in Figure 8.3 that approximates the solution $\mathbf{x}$ to $\mathbf{A}\mathbf{x} = \mathbf{b}$ with the iterate from $\mathcal{K}_k(\mathbf{A}, \mathbf{b})$ that minimizes the residual:

$$\|\mathbf{r}_k\| = \|\mathbf{b} - \mathbf{A}\mathbf{x}_k\| = \min_{\widehat{\mathbf{x}} \in \mathcal{K}_k(\mathbf{A},\mathbf{b})} \|\mathbf{b} - \mathbf{A}\widehat{\mathbf{x}}\|.$$

Since $\mathcal{R}(\mathbf{U}_k) = \mathcal{K}_k(\mathbf{A}, \mathbf{b})$, any vector $\widehat{\mathbf{x}} \in \mathcal{K}_k(\mathbf{A}, \mathbf{b})$ can be written as $\mathbf{U}_k\mathbf{c}$ for some $\mathbf{c} \in \mathbb{R}^k$. Hence

$$\|\mathbf{r}_k\| = \min_{\widehat{\mathbf{x}} \in \mathcal{K}_k(\mathbf{A},\mathbf{b})} \|\mathbf{b} - \mathbf{A}\widehat{\mathbf{x}}\| = \min_{\mathbf{c} \in \mathbb{R}^k} \|\mathbf{b} - \mathbf{A}\mathbf{U}_k\mathbf{c}\|.$$

We could now make a small modification to the code in Figure 8.3 to replace the poor Krylov basis matrix $\mathbf{K}_k$ with the orthonormal basis $\mathbf{U}_k$. This would lead to a solid, robust algorithm, *but it would still be quite expensive*. Instead of the least squares problem (8.6), at each step we now solve (8.12). Since the matrix $\mathbf{A}\mathbf{U}_k$ has dimension $n \times k$, and we are most interested in the case of very large n, this will still be very costly. It turns out that there is a slick trick that solves this least squares problem by working only with a $(k+1) \times k$ matrix in place of the $n \times k$ matrix: an enormous savings.

This major reduction in work follows from a slick use of the Arnoldi relationship $\mathbf{A}\mathbf{U}_k = \mathbf{U}_{k+1}\widetilde{\mathbf{H}}_k$. Substitute the right-hand side in place of $\mathbf{A}\mathbf{U}_k$ in (8.12) to get

$$\|\mathbf{r}_k\| = \min_{\mathbf{c} \in \mathbb{R}^k} \|\mathbf{b} - \mathbf{U}_{k+1}\widetilde{\mathbf{H}}_k\mathbf{c}\|. \tag{8.12}$$

Recall now that the first column of $\mathbf{U}_{k+1}$ is the normalized starting vector, $\mathbf{U}_1 = \mathbf{b}/\|\mathbf{b}\|$. Hence we can write $\mathbf{b}$ in (8.12) as

$$\mathbf{b} = \|\mathbf{b}\|\mathbf{U}_1 = \|\mathbf{b}\|\mathbf{U}_{k+1}\mathbf{e}_1,$$

where $\mathbf{e}_1$ is the first column of the $(k+1) \times (k+1)$ identity. This looks like a complication, but it allows for a key simplification

in (8.12). First recast (8.12) as

$$\begin{aligned}\|\mathbf{r}_k\| &= \min_{\mathbf{c}\in\mathbb{R}^k} \left\| \|\mathbf{b}\|\mathbf{U}_{k+1}\mathbf{e}_1 - \mathbf{U}_{k+1}\widetilde{\mathbf{H}}_k\mathbf{c} \right\| \\ &= \min_{\mathbf{c}\in\mathbb{R}^k} \left\| \mathbf{U}_{k+1}\Big(\|\mathbf{b}\|\mathbf{e}_1 - \widetilde{\mathbf{H}}_k\mathbf{c} \Big) \right\|. \end{aligned} \tag{8.13}$$

Now notice this neat implication of the orthonormality of the columns of $\mathbf{U}_{k+1}$: for any vector $\mathbf{y} \in \mathbb{R}^{k+1}$,

$$\|\mathbf{U}_{k+1}\mathbf{y}\|^2 = \mathbf{y}^T\mathbf{U}_{k+1}^T\mathbf{U}_{k+1}\mathbf{y} = \mathbf{y}^T\mathbf{I}\mathbf{y} = \|\mathbf{y}\|^2.$$

Applying this observation to (8.13) with $\mathbf{y} = \|\mathbf{b}\|\mathbf{e}_1 - \mathbf{U}_{k+1}\widetilde{\mathbf{H}}_k\mathbf{c}$ gives

$$\|\mathbf{r}_k\| = \min_{\mathbf{c}\in\mathbb{R}^k} \left\| \|\mathbf{b}\|\mathbf{e}_1 - \widetilde{\mathbf{H}}_k\mathbf{c} \right\|, \tag{8.14}$$

which is a small least squares problem involving only the $(k+1)\times k$ matrix $\widetilde{\mathbf{H}}_k$: beyond the work of constructing the orthonormal Arnoldi basis for $\mathcal{K}_k(\mathbf{A},\mathbf{b})$, there is no work with n-dimensional objects. If we denote by $\mathbf{c}_k$ the optimal $\mathbf{c}$ in (8.14), then

$$\|\mathbf{r}_k\| = \left\| \|\mathbf{b}\|\mathbf{e}_1 - \widetilde{\mathbf{H}}_k\mathbf{c}_k \right\|.$$

This formula allows us to compute $\|\mathbf{r}_k\|$ without actually constructing $\mathbf{x}_k$ and $\mathbf{r}_k = \mathbf{b} - \mathbf{A}\mathbf{x}_k$: thus we can monitor the size of $\|\mathbf{r}_k\|$ as we iterate through increasing values of k until $\|\mathbf{r}_k\| \leq$ TOLERANCE, and then compute $\mathbf{x}_k$ and declare victory.

8.5 Convergence theory

8.6 Related algorithms

8.6.1 Suboptimal but (potentially) fast iterations

Restarted GMRES, BiCGSTAB, QMR, etc.

8.6.2 Optimal, fast iterations for symmetric matrices

Many applications lead to symmetric matrices, $\mathbf{A}^T = \mathbf{A}$. In this case, the Arnoldi relationship (8.11) has a fascinating implication:

$$\mathbf{H}_k^T = (\mathbf{U}_k^T\mathbf{A}\mathbf{U}_k)^T = \mathbf{U}_k^T\mathbf{A}^T\mathbf{U}_k = \mathbf{U}_k^T\mathbf{A}\mathbf{U}_k = \mathbf{H}_k,$$

so $\mathbf{H}_k$ is also symmetric. But all entries of $\mathbf{H}_k$ below the first subdiagonal are always zero by construction: $h_{j,\ell} = 0$ if $j > \ell + 1$. The symmetry of $\mathbf{H}_k$ then also implies that

$$h_{\ell,j} = 0, \qquad j > \ell + 1.$$

This means that

$$\mathbf{u}_\ell^T \mathbf{A}\mathbf{u}_j = 0, \qquad j > \ell + 1,$$

and $\mathbf{H}_k$ is a *tridiagonal matrix*:

$$\mathbf{H}_k = \begin{bmatrix} h_{1,1} & h_{1,2} & & \\ h_{2,1} & h_{2,2} & \ddots & \\ & \ddots & \ddots & h_{k-1,k} \\ & & h_{k,k-1} & h_{k,k} \end{bmatrix} \in \mathbb{R}^{k \times k}.$$

This observation has an enormous implication for the Arnoldi process: we know, without performing any computation, that $\mathbf{A}\mathbf{u}_\ell$ has no component in the $\mathbf{u}_j$ direction when $j < \ell - 1$:

$$\mathbf{u}_j \mathbf{u}_j^T \mathbf{A}\mathbf{u}_\ell = (\mathbf{u}_j^T \mathbf{A}\mathbf{u}_\ell)\mathbf{u}_j = \mathbf{0}, \qquad j < \ell - 1.$$

Since we know these terms are zero, we need not bother to compute them. This observation greatly expedites the Arnoldi process, for it means that we can construct each new vector $\mathbf{u}_{\ell+1}$ with the same amount of work, as ℓ increases.

MINRES and the conjugate gradient method for symmetric (postitive definite) $\mathbf{A}$.

8.7 Preconditioning

8.8 Sparse direct methods

While iterative methods play an important role in scientific computing, especially for large-scale models (e.g., in three physical dimensions, or incorporating multiple physical phenomena), we should take note of the important class of *sparse direct methods*. These algorithms are variants of traditional Gaussian elimination (LU factorization) that seek to exploit the sparse structure of $\mathbf{A}$. MATLAB includes such methods via the built-in UMFPACK (unsymmetric multi-frontal package) software suite.