Chapter 10

Restarting Arnoldi and Lanczos algorithms

The number of iteration steps can be very high with the Arnoldi or the Lanczos algorithm. This number is, of course, not predictable. The iteration count depends on properties of the matrix, in particular the distribution of its eigenvalues, but also on the initial vectors.

High iteration counts entail a large memory requirement to store the Arnoldi/Lanczos vectors and a high amount of computation because of growing cost of the reorthogonalization.

The idea behind the implicitely restarted Arnoldi (IRA) and implicitely restarted Lanczos (IRL) algorithms is to reduce these costs by limiting the dimension of the search space. This means that the iteration is stopped after a number of steps (which is bigger than the number of desired eigenvalues), reduce the dimension of the search space without destroying the Krylov space tructure, and finally resume the Arnoldi / Lanczos iteration.

The implicitely restarted Arnoldi has first been proposed by Sorensen [7, 8]. It is implemented together with the implicitely restarted Lanczos algorithms in the software package ARPACK [4]. The ARPACK routines are the basis for the sparse matrix eigensolver `eigs` in MATLAB.

10.1 The m-step Arnoldi iteration

Algorithm 10.1 The m-step Arnoldi iteration

```
1: Let $A \in \mathbb{F}^{n\times n}$. This algorithm executes $m$ steps of the Arnoldi algorithm.
2: $\mathbf{q}_1 = \mathbf{x}/\|\mathbf{x}\|$;   $\mathbf{z} = A\mathbf{q}_1$;   $\alpha_1 = \mathbf{q}_1^*\mathbf{z}$;
3: $\mathbf{r}_1 = \mathbf{w} - \alpha_1\mathbf{q}_1$;   $Q_1 = [\mathbf{q}_1]$;   $H_1 = [\alpha_1]$;
4: for $j = 1, \ldots, m-1$ do
5:     $\beta_j := \|\mathbf{r}_j\|$;   $\mathbf{q}_{j+1} = \mathbf{r}_j/\beta_j$;
6:     $Q_{j+1} := [Q_j, \mathbf{q}_{j+1}]$;   $\hat{H}_j := \begin{bmatrix} H_j \\ \beta_j\mathbf{e}_j^T \end{bmatrix} \in \mathbb{F}^{(j+1)\times j}$;
7:     $\mathbf{z} := A\mathbf{q}_j$;
8:     $\mathbf{h} := Q_{j+1}^*\mathbf{z}$;   $\mathbf{r}_{j+1} := \mathbf{z} - Q_{j+1}\mathbf{h}$;
9:     $H_{j+1} := [\hat{H}_j, \mathbf{h}]$;
10: end for
```

We start with the Algorithm 10.1 that is a variant of the Arnoldi Algorithm 9.1. It executes just m Arnoldi iteration steps. We will now show how the dimension of the search space is reduced withouth losing the information regarding the eigenvectors one is looking for.

Remark 10.1. Step 8 in Algorithm 10.1 is classical Gram–Schmidt orthogonalization. As

$$\mathbf{r}_{j+1} = \mathbf{z} - Q_{j+1}\mathbf{h} = \mathbf{z} - Q_{j+1}Q_{j+1}^*\mathbf{z},$$

we formally have $Q_{j+1}^*\mathbf{r}_{j+1} = \mathbf{0}$. However, classical Gram–Schmidt orthogonalization is faster but not so accurate as modified Gram–Schmidt orthogonalization [1]. So, often, $Q_{j+1}^*\mathbf{r}_{j+1}$ is quite large. Therefore, the orthogonalization is iterated to get sufficient orthogonality.

A possible modification of step 8 that incorporates a second iteration is

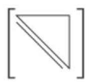
```
8: $\mathbf{h} := Q_{j+1}^*\mathbf{z}$;   $\mathbf{r}_{j+1} := \mathbf{z} - Q_{j+1}\mathbf{h}$;
   $\mathbf{c} := Q_{j+1}^*\mathbf{r}_{j+1}$;   $\mathbf{r}_{j+1} := \mathbf{r}_{j+1} - Q_{j+1}\mathbf{c}$;   $\mathbf{h} = \mathbf{h} + \mathbf{c}$;
```

Now we have,

$$\begin{aligned}\tilde{\mathbf{r}}_{j+1} &= \text{corrected } \mathbf{r}_{j+1} \\ &= \mathbf{r}_{j+1} - Q_{j+1}\underbrace{Q_{j+1}^*\mathbf{r}_{j+1}}_{\mathbf{c}} \\ &= \mathbf{z} - Q_{j+1}\underbrace{Q_{j+1}^*\mathbf{z}}_{\mathbf{h}} - Q_{j+1}\underbrace{Q_{j+1}^*\mathbf{r}_{j+1}}_{\mathbf{c}} = \mathbf{z} - Q_{j+1}(\mathbf{h}+\mathbf{c})\end{aligned}$$

More iterations are possible but seldom necessary. □

After the execution of Algorithm 10.1 we have the Arnoldi / Lanczos relation

(10.1) $$AQ_m = Q_mH_m + \mathbf{r}_m\mathbf{e}_m^*, \qquad H_m =$$ 

available with

$$\mathbf{r}_m = \beta_m\mathbf{q}_{m+1}, \qquad \|\mathbf{q}_{m+1}\| = 1.$$

If $\beta_m = 0$ then $\mathcal{R}(Q_m)$ is invariant under A, i.e., $A\mathbf{x} \in \mathcal{R}(Q_m)$ for all $\mathbf{x} \in \mathcal{R}(Q_m)$. This lucky situation implies that $\sigma(H_m) \subset \sigma_m(A)$. So, the Ritz values and vectors are eigenvalues and eigenvectors of A.

What we can realistically hope for is β_m being small. Then,

$$AQ_m - \mathbf{r}_m\mathbf{e}_m^* = (A - \mathbf{r}_m\mathbf{q}_m^*)Q_m = Q_mH_m.$$

Then, $\mathcal{R}(Q_m)$ is invariant under a matrix $A+E$, that differs from A by a perturbation E with $\|E\| = \|\mathbf{r}_m\| = |\beta_m|$. From general eigenvalue theory we know that this in this situation well-conditioned eigenvalues of H_m are good approximations of eigenvalues of A.

In the sequel we investigate how we can find a q_1 such that β_m becomes small?

10.2 Implicit restart

Let us start from the Arnoldi relation

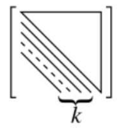
(10.2) $$AQ_m = Q_mH_m + \mathbf{r}_m\mathbf{e}_m^*,$$

Algorithm 10.2 k implicit QR steps applied to H_m

```
1: $H_m^+ := H_m$.
2: for $i := 1, \ldots, k$ do
3:     $H_m^+ := V_i^* H_m^+ V_i$,        where $H_m^+ - \mu_i I = V_i R_i$   (QR factorization)
4: end for
```

that is obtained after calling Algorithm 10.1.

We apply $k < m$ implicit QR steps to H_m with shifts $\mu_1, \ldots, \mu_k$, see Algorithm 10.2. Let $V^+ := V_1 V_2 \cdots V_k$. V^+ is the product of k (unitary) Hessenberg matrices whence it has k nonzero off-diagonals below its main diagonal.

$V_m^+ =$ 

.

We define

$$Q_m^+ := Q_m V^+, \qquad H_m^+ := (V^+)^* H_m V^+.$$

Then, from (10.2) we obtain

$$AQ_m V^+ = Q_m V^+ (V^+)^* H_m V^+ + \mathbf{r}_m \mathbf{e}_m^* V^+,$$

or

$$AQ_m^+ = Q_m^+ H_m^+ + \mathbf{r}_m \mathbf{e}_m^* V^+. \tag{10.3}$$

As V^+ has k nonzero off-diagonals below the main diagonal, the last row of V^+ has the form

$$\mathbf{e}_m^* V^+ = (\underbrace{0, \ldots, 0}_{p-1}, \underbrace{*, \ldots, *}_{k+1}), \qquad k + p = m.$$

We now simply discard the last k columns of (10.3).

$$\begin{aligned} AQ_m^+(:, 1:p) &= Q_m^+ H_m^+(:, 1:p) + \mathbf{r}_m \mathbf{e}_m^* V^+(:, 1:p) \\ &= Q_m^+(:, 1:p) H_m^+(1:p, 1:p) + \underbrace{h_{p+1,p}^+}_{\beta_p^+} \mathbf{q}_{p+1}^+ \mathbf{e}_p^* + v_{m,p}^+ \mathbf{r}_m \mathbf{e}_p^* \\ &= Q_m^+(:, 1:p) H_m^+(1:p, 1:p) + \underbrace{(\mathbf{q}_{p+1}^+ h_{p+1,p}^+ + \mathbf{r}_m v_{m,p}^+)}_{\mathbf{r}_p^+} \mathbf{e}_p^*. \end{aligned}$$

In Algorithm 10.3 we have collected what we have derived so far. We have however left open in step 3 of the algorithm *how* the shifts $\mu_1, \ldots, \mu_k$ should be chosen. In ARPACK [4], all eigenvalues of H_m are computed. Those k eigenvalues that are furthest away from some target value are chosen as shifts. We have not specified how we determine convergence, too.

One can show that a QR step with shift μ_i transforms the vector $\mathbf{q}_1$ in a multiple of $(A - \mu_i I)\mathbf{q}_1$. In fact, a simple modification of the Arnoldi relation (10.2) gives

$$(A - \mu_i I) Q_m = Q_m \underbrace{(H_m - \mu_i I)}_{V_1 R_1} + \mathbf{r}_m \mathbf{e}_m^* = Q_m V_1 R_1 + \mathbf{r}_m \mathbf{e}_m^*.$$

Algorithm 10.3 Implicitely restarted Arnoldi (IRA)

```
 1: Let the Arnoldi relation $AQ_m = Q_m H_m + \mathbf{r}_m \mathbf{e}_m^*$ be given.
 2: repeat
 3:   Determine $k$ shifts $\mu_1, \ldots, \mu_k$;
 4:   $\mathbf{v}^* := \mathbf{e}_m^*$;
 5:   for $i = 1, \ldots, k$ do
 6:     $H_m - \mu_i I = V_i R_i$; /* QR factorization */
 7:     $H_m := V_i^* H_m V_i$;   $Q_m := Q_m V_i$;
 8:     $\mathbf{v}^* := \mathbf{v}^* V_i$;
 9:   end for
10:   $\mathbf{r}_p := \mathbf{q}_{p+1}^+ \beta_p^+ + \mathbf{r}_m v_{m,p}^+$;
11:   $Q_p := Q_m(:, 1:p)$;   $H_p := H_m(1:p, 1:p)$;
12:   Starting with
          $$AQ_p = Q_p H_p + \mathbf{r}_p \mathbf{e}_p^*$$
      execute $k$ additional steps of the Arnoldi algorithm until
          $$AQ_m = Q_m H_m + \mathbf{r}_m \mathbf{e}_m^*.$$
13: until convergence
```

Comparing the first columns in this equation gives

$$(A - \mu_i I)\mathbf{q}_1 = \mathbf{q}_1^{(1)} V_1 \mathbf{e}_1 r_{11} + \mathbf{0} = \mathbf{q}_1^{(1)} r_{11}.$$

By consequence, all k steps combined give

$$\mathbf{q}_1 \longleftarrow \Psi(A)\mathbf{q}_1, \qquad \Psi(\lambda) = \prod_{i=1}^{k} (\lambda - \mu_i).$$

If μ_i were an eigenvalue of A then $(A - \mu_i I)\mathbf{q}_1$ removes components of $\mathbf{q}_1$ in the direction of the corresponding eigenvector. More general, if μ_i is close to an eigenvalue of A then $(A - \mu_i I)\mathbf{q}_1$ will have only small components in the direction of eigenvectors corresponding to nearby eigenvalues. Choosing the μ_i equal to Ritz values far away from the desired part of the spectrum thus enhances the desired component. Still there is the danger that in each sweep on Algorithm 10.3 the same undesired Ritz values are recovered. Therefore, other strategies for choosing the shifts have been proposed [2]. Experimental results indicate however, that the original strategy chosen in ARPACK mostly works best.

10.3 Convergence criterion

Let $H_m \mathbf{s} = \mathbf{s}\vartheta$ with $\|\mathbf{s}\| = 1$. Let $\hat{\mathbf{x}} = Q_m \mathbf{s}$. Then we have as earlier

$$\|A\hat{\mathbf{x}} - \vartheta\hat{\mathbf{x}}\| = \|AQ_m\mathbf{s} - Q_m H_m \mathbf{s}\| = \|\mathbf{r}_m\| |\mathbf{e}_m^* \mathbf{s}| = \beta_m |\mathbf{e}_m^* \mathbf{s}|. \tag{10.4}$$

In the Hermitian case, $A = A^*$, the Theorem 9.1 of Krylov–Bogoliubov provides an interval that contains an eigenvalue of A. In the general case, we have

$$(A + E)\hat{\mathbf{x}} = \vartheta\hat{\mathbf{x}}, \qquad E = -\mathbf{r}_m \mathbf{q}_m^*, \quad \|E\| = \|\mathbf{r}_m\| = \beta_m. \tag{10.5}$$

According to an earlier theorem we know that if $\lambda \in \sigma(A)$ is simple and ϑ is the eigenvalue of $A + E$ closest to λ, then

$$|\lambda - \vartheta| \leq \frac{\|E\|}{\mathbf{y}^*\mathbf{x}} + \mathcal{O}(\|E\|^2). \tag{10.6}$$

Here, $\mathbf{y}$ and $\mathbf{x}$ are left and right eigenvectors of E corresponding to the eigenvalue λ. A similar statement holds for the eigenvectors, but the distance (gap) to the next eigenvalue comes into play as well.

In ARPACK, a Ritz pair $(\vartheta, \hat{\mathbf{x}})$ is considered converged if

$$\beta_m|\mathbf{e}_m^*\mathbf{s}| \leq \max(\varepsilon_M\|H_m\|, \text{tol}\cdot|\vartheta|). \tag{10.7}$$

As $|\vartheta| \leq \|H_m\| \leq \|A\|$, the inequality $\|E\| \leq \text{tol}\cdot\|A\|$ holds at convergence. According to (10.6) well-conditioned eigenvalues are well approximated.

10.4 The generalized eigenvalue problem

Let us consider now the generalized eigenvalue problem

$$A\mathbf{x} = \lambda M\mathbf{x}. \tag{10.8}$$

Applying a shift-and-invert spectral transformation with shift σ transforms (10.8) into

$$S\mathbf{x} = (A - \sigma M)^{-1}M\mathbf{x} = \mu\mathbf{x}, \qquad \mu = \frac{1}{\lambda - \sigma}. \tag{10.9}$$

We now execute an Arnoldi/Lanczos iteration with S to obtain

$$SQ_m = Q_mH_m + \mathbf{r}_m\mathbf{e}_m^*, \qquad Q_m^*MQ_m = I_m, \quad Q_m^*M\mathbf{r}_m = \mathbf{0}. \tag{10.10}$$

Let $\mathbf{s}$ with $\|\mathbf{s}\| = 1$ be an eigenvector of H_m with Ritz value ϑ. Let $\mathbf{y} = Q_m\mathbf{s}$ be the associated Ritz vector. Then,

$$SQ_m\mathbf{s} = S\mathbf{y} = Q_mH_m\mathbf{s} + \mathbf{r}_m\mathbf{e}_m^*\mathbf{s} = \mathbf{y}\vartheta + \mathbf{r}_m\mathbf{e}_m^*\mathbf{s}. \tag{10.11}$$

So, $\mathbf{y}\vartheta + \mathbf{r}_m\mathbf{e}_m^*\mathbf{s}$ can be considered a vector that is obtained by one step of inverse iteration. This vector is an improved approximation to the desired eigenvector, obtained at negligible cost. This so-called **eigenvector purification** is particularly important if M is singular.

Let us bound the residual norm of the purified vector. With (10.11) we have

$$M\mathbf{y} = (A - \sigma M)\underbrace{(\mathbf{y}\vartheta + \mathbf{r}_m\mathbf{e}_m^*\mathbf{s})}_{\tilde{\mathbf{y}}} \tag{10.12}$$

with

$$\|\tilde{\mathbf{y}}\|_M = \sqrt{\vartheta^2 + \beta_k^2|\mathbf{e}_m^*\mathbf{s}|^2}.$$

This equality holds as $\mathbf{y} \perp_M \mathbf{r}$. By consequence,

$$\begin{aligned} \|A\tilde{\mathbf{y}} - \lambda M\tilde{\mathbf{y}}\| &= \|(A - \sigma M)\tilde{\mathbf{y}} + M\tilde{\mathbf{y}}\underbrace{(\sigma - \lambda)}_{-\frac{1}{\vartheta}}\| \\ &= \|M\mathbf{y} - M(\mathbf{y}\vartheta + \mathbf{r}_m\mathbf{e}_m^*\mathbf{s})/\vartheta\| = \|M\mathbf{r}\|\ |\mathbf{e}_m^*\mathbf{s}|/|\vartheta|. \end{aligned} \tag{10.13}$$

Since $|\vartheta|$ is large in general, we obtain good bounds for the residual of the purified eigenvectors.

```
EIGS  Find a few eigenvalues and eigenvectors of a matrix using ARPACK
   D = EIGS(A) returns a vector of A's 6 largest magnitude eigenvalues.
   A must be square and should be large and sparse.

   [V,D] = EIGS(A) returns a diagonal matrix D of A's 6 largest magnitude
   eigenvalues and a matrix V whose columns are the corresponding
   eigenvectors.

   [V,D,FLAG] = EIGS(A) also returns a convergence flag. If FLAG is 0 then
   all the eigenvalues converged; otherwise not all converged.

   EIGS(A,B) solves the generalized eigenvalue problem A*V == B*V*D. B
   must be symmetric (or Hermitian) positive definite and the same size as
   A. EIGS(A,[],...) indicates the standard eigenvalue problem A*V == V*D.

   EIGS(A,K) and EIGS(A,B,K) return the K largest magnitude eigenvalues.

   EIGS(A,K,SIGMA) and EIGS(A,B,K,SIGMA) return K eigenvalues. If SIGMA is:
      'LM' or 'SM' - Largest or Smallest Magnitude
   For real symmetric problems, SIGMA may also be:
      'LA' or 'SA' - Largest or Smallest Algebraic
      'BE' - Both Ends, one more from high end if K is odd
   For nonsymmetric and complex problems, SIGMA may also be:
      'LR' or 'SR' - Largest or Smallest Real part
      'LI' or 'SI' - Largest or Smallest Imaginary part
   If SIGMA is a real or complex scalar including 0, EIGS finds the
   eigenvalues closest to SIGMA. For scalar SIGMA, and when SIGMA = 'SM',
   B need only be symmetric (or Hermitian) positive semi-definite since it
   is not Cholesky factored as in the other cases.

   EIGS(A,K,SIGMA,OPTS) and EIGS(A,B,K,SIGMA,OPTS) specify options:
   OPTS.issym: symmetry of A or A-SIGMA*B represented by AFUN [{false} | true]
   OPTS.isreal: complexity of A or A-SIGMA*B represented by AFUN [false | {true}]
   OPTS.tol: convergence: Ritz estimate residual <= tol*NORM(A) [scalar | {eps}]
   OPTS.maxit: maximum number of iterations [integer | {300}]
   OPTS.p: number of Lanczos vectors: K+1<p<=N [integer | {2K}]
   OPTS.v0: starting vector [N-by-1 vector | {randomly generated}]
   OPTS.disp: diagnostic information display level [0 | {1} | 2]
   OPTS.cholB: B is actually its Cholesky factor CHOL(B) [{false} | true]
   OPTS.permB: sparse B is actually CHOL(B(permB,permB)) [permB | {1:N}]
   Use CHOL(B) instead of B when SIGMA is a string other than 'SM'.

   EIGS(AFUN,N) accepts the function AFUN instead of the matrix A. AFUN is
   a function handle and Y = AFUN(X) should return
      A*X            if SIGMA is unspecified, or a string other than 'SM'
      A\X            if SIGMA is 0 or 'SM'
      (A-SIGMA*I)\X  if SIGMA is a nonzero scalar (standard problem)
      (A-SIGMA*B)\X  if SIGMA is a nonzero scalar (generalized problem)
   N is the size of A. The matrix A, A-SIGMA*I or A-SIGMA*B represented by
   AFUN is assumed to be real and nonsymmetric unless specified otherwise
   by OPTS.isreal and OPTS.issym. In all these EIGS syntaxes, EIGS(A,...)
   may be replaced by EIGS(AFUN,N,...).

   Example:
      A = delsq(numgrid('C',15));  d1 = eigs(A,5,'SM');

   Equivalently, if dnRk is the following one-line function:
      %----------------------------%
      function y = dnRk(x,R,k)
      y = (delsq(numgrid(R,k))) \ x;
      %----------------------------%

      n = size(A,1);  opts.issym = 1;
      d2 = eigs(@(x)dnRk(x,'C',15),n,5,'SM',opts);

   See also eig, svds, ARPACKC, function_handle.
```

10.5 A numerical example

This example is taken from the MATLAB document pages regarding `eigs`. `eigs` is the MATLAB interface to the ARPACK code, see page 182. The matrix called `west0479` is a 479×479 matrix originating in a chemical engineering plant model. The matrix is available from the Matrix Market [5], a web site that provides numerous test matrices. Its nonzero structure is given in Fig. 10.1

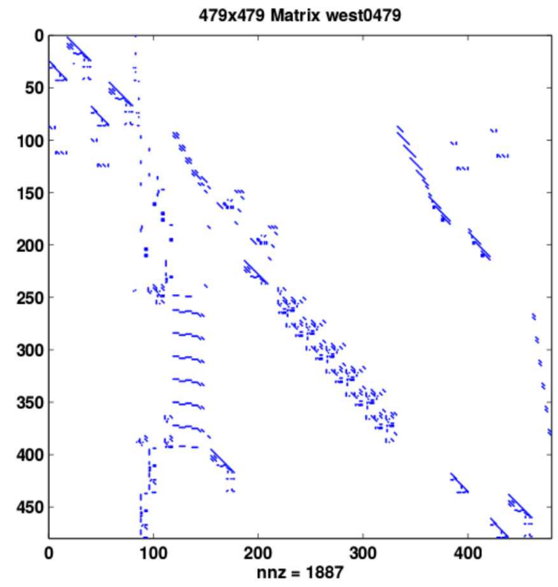


Figure 10.1: Nonzero structure of the 479 × 479 matrix west0479

To compute the eight largest eigenvalues of this matrix we issue the following MATLAB commands.

```
>> load west0479
>> d = eig(full(west0479));
>> dlm=eigs(west0479,8);
Iteration 1: a few Ritz values of the 20-by-20 matrix:
     0
     0
     0
     0
     0
     0
     0
     0
     0

Iteration 2: a few Ritz values of the 20-by-20 matrix:
   1.0e+03 *

  -0.0561 - 0.0536i
```

```
   0.1081 + 0.0541i
   0.1081 - 0.0541i
  -0.1009 - 0.0666i
  -0.1009 + 0.0666i
  -0.0072 + 0.1207i
  -0.0072 - 0.1207i
   0.0000 - 1.7007i
   0.0000 + 1.7007i

Iteration 3: a few Ritz values of the 20-by-20 matrix:
   1.0e+03 *

  -0.0866
  -0.1009 - 0.0666i
  -0.1009 + 0.0666i
  -0.0072 + 0.1207i
  -0.0072 - 0.1207i
   0.1081 - 0.0541i
   0.1081 + 0.0541i
   0.0000 - 1.7007i
   0.0000 + 1.7007i

Iteration 4: a few Ritz values of the 20-by-20 matrix:
   1.0e+03 *

   0.0614 - 0.0465i
  -0.0072 - 0.1207i
  -0.0072 + 0.1207i
   0.1081 + 0.0541i
   0.1081 - 0.0541i
  -0.1009 + 0.0666i
  -0.1009 - 0.0666i
   0.0000 - 1.7007i
   0.0000 + 1.7007i

Iteration 5: a few Ritz values of the 20-by-20 matrix:
   1.0e+03 *

  -0.0808
  -0.0072 + 0.1207i
  -0.0072 - 0.1207i
  -0.1009 - 0.0666i
  -0.1009 + 0.0666i
   0.1081 + 0.0541i
   0.1081 - 0.0541i
   0.0000 + 1.7007i
   0.0000 - 1.7007i

Iteration 6: a few Ritz values of the 20-by-20 matrix:
   1.0e+03 *

   0.0734 - 0.0095i
  -0.0072 + 0.1207i
  -0.0072 - 0.1207i
   0.1081 - 0.0541i
```

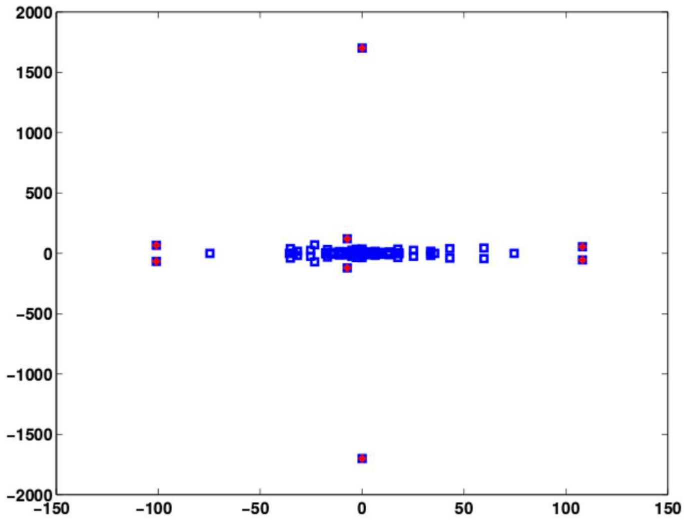


Figure 10.2: Spectrum of the matrix west0479

```
   0.1081 + 0.0541i
  -0.1009 - 0.0666i
  -0.1009 + 0.0666i
   0.0000 - 1.7007i
   0.0000 + 1.7007i

Iteration 7: a few Ritz values of the 20-by-20 matrix:
   1.0e+03 *

  -0.0747
  -0.0072 - 0.1207i
  -0.0072 + 0.1207i
   0.1081 + 0.0541i
   0.1081 - 0.0541i
  -0.1009 + 0.0666i
  -0.1009 - 0.0666i
   0.0000 + 1.7007i
   0.0000 - 1.7007i
```

The output indicates that `eigs` needs seven sweeps to compute the eigenvalues to the default accuracy of macheps$\|A\|$. The Ritz values given are the approximations of the eigenvalues we want to compute. The complete spectrum of `west0479` is given in Fig. 10.2. Notice the different scales of the axes! Fig. 10.3 is a zoom that shows all eigenvalues except the two very large ones. Here the axes are equally scaled. From the two figures it becomes clear that `eigs` has computed the eight eigenvalues (and corresponding eigenvectors) of *largest modulus*.

To compute the eigenvalues smallest in modulus we issue the following command.

```
dsm=eigs(west0479,8,'sm');
Iteration 1: a few Ritz values of the 20-by-20 matrix:
     0
```

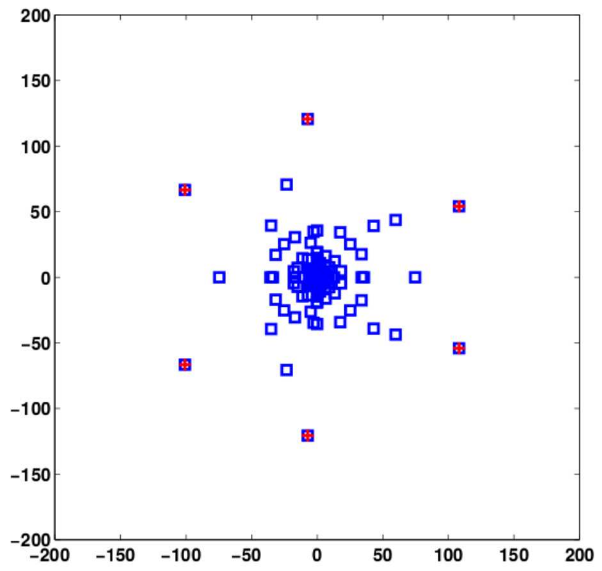


Figure 10.3: A zoom to the center of the spectrum of matrix west0479 that excludes the largest two eigenvalues on the imaginary axis

```
         0
         0
         0
         0
         0
         0
         0
         0

Iteration 2: a few Ritz values of the 20-by-20 matrix:
   1.0e+03 *

  -0.0228 - 0.0334i
   0.0444
  -0.0473
   0.0116 + 0.0573i
   0.0116 - 0.0573i
  -0.0136 - 0.1752i
  -0.0136 + 0.1752i
  -3.4455
   5.8308

Iteration 3: a few Ritz values of the 20-by-20 matrix:
   1.0e+03 *
  -0.0228 - 0.0334i
   0.0444
  -0.0473
   0.0116 + 0.0573i
   0.0116 - 0.0573i
  -0.0136 + 0.1752i
  -0.0136 - 0.1752i
  -3.4455
   5.8308

Iteration 4: a few Ritz values of the 20-by-20 matrix:
   1.0e+03 *

  -0.0228 + 0.0334i
   0.0444
  -0.0473
   0.0116 - 0.0573i
   0.0116 + 0.0573i
  -0.0136 + 0.1752i
  -0.0136 - 0.1752i
  -3.4455
   5.8308

Iteration 5: a few Ritz values of the 20-by-20 matrix:
   1.0e+03 *

  -0.0228 + 0.0334i
   0.0444
  -0.0473
   0.0116 - 0.0573i
   0.0116 + 0.0573i
  -0.0136 + 0.1752i
  -0.0136 - 0.1752i
  -3.4455
   5.8308

>> dsm

dsm =

   0.0002
  -0.0003
  -0.0004 - 0.0057i
  -0.0004 + 0.0057i
   0.0034 - 0.0168i
   0.0034 + 0.0168i
  -0.0211
   0.0225
```

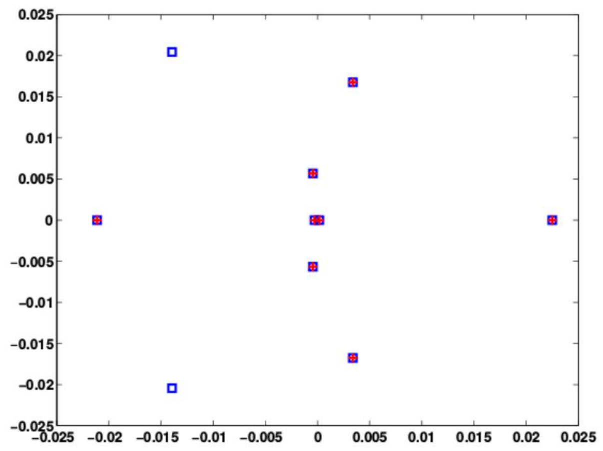


Figure 10.4: Smallest eigenvalues of the matrix west0479

```
>> 1./dsm

ans =

   1.0e+03 *

   5.8308
  -3.4455
  -0.0136 + 0.1752i
  -0.0136 - 0.1752i
   0.0116 + 0.0573i
   0.0116 - 0.0573i
  -0.0473
   0.0444
```

The computed eigenvalues are depicted in Fig. 10.4

10.6 Another numerical example

We revisit the determination the acoustic eigenfrequencies and modes in the interior of a car, see section 1.6.3. The computations are done with the finest grid depicted in Fig. 1.9. We first compute the lowest ten eigenpairs with simultaneous inverse vector iteration (`sivit`). The dimension of the search space is 15.

```
>> [p,e,t]=initmesh('auto');
>> [p,e,t]=refinemesh('auto',p,e,t);
>> [p,e,t]=refinemesh('auto',p,e,t);
>> p=jigglemesh(p,e,t);
>> [A,M]=assema(p,t,1,1,0);
>> whos
  Name      Size                    Bytes  Class
  A      1095x1095                91540  double array (sparse)
  M      1095x1095                91780  double array (sparse)
  e         7x188                 10528  double array
  p         2x1095                17520  double array
  t         4x2000                64000  double array

Grand total is 26052 elements using 275368 bytes

>> sigma=-.01;
>> p=10; tol=1e-6; X0=rand(size(A,1),15);
>> [V,L] = sivit(A,M,p,X0,sigma,tol);

 ||Res(0)|| = 0.998973
 ||Res(5)|| = 0.603809
 ||Res(10)|| = 0.0171238
 ||Res(15)|| = 0.00156298
 ||Res(20)|| = 3.69725e-05
 ||Res(25)|| = 7.11911e-07
>> %  25 x 15 =  375 matrix - vektor - multiplications until convergence
>>
>> format long, L

L =

   0.00000000000000
   0.01269007628847
   0.04438457596824
   0.05663501055565
   0.11663116522140
   0.13759210393200
   0.14273438015546
   0.20097619880776
   0.27263682280769
   0.29266080747831

>> format short
>> norm(V'*M*V - eye(10))

ans =

     1.8382e-15
```

Then we use MATLAB's solver `eigs`. We set the tolerance and the shift to be the same as with `sivit`. Notice that ARPACK applies a shift-and-invert spectral transformation if a shift is given.

```
>> options.tol=tol; options.issym=1;
>> [v,l,flag]=eigs(A,M,p,sigma,options);
Iteration 1: a few Ritz values of the 20-by-20 matrix:
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0

Iteration 2: a few Ritz values of the 20-by-20 matrix:
    3.3039
    3.5381
    4.7399
    6.5473
    6.7754
    7.8970
   15.0071
   18.3876
   44.0721
  100.0000

Iteration 3: a few Ritz values of the 20-by-20 matrix:
    3.3040
    3.5381
    4.7399
    6.5473
    6.7754
    7.8970
   15.0071
   18.3876
   44.0721
  100.0000

>> flag

flag =

     0

>> l=diag(l);  l=l(end:-1:1); norm(l-L)

ans =

   3.7671e-14

>> norm(v'*M*v - eye(10))

ans = 8.0575e-15
```

Clearly the eigenvectors are mutually m-orthogonal. Notice that `eigs` returns the eigenvalues sorted from large to small such that they have to be reordered before comparing with those `sivit` computed.

In the next step we compute the largest eigenvalues of the matrix

$$S = R(A - \sigma M)^{-1} R^T, \tag{10.14}$$

where $R^T R = M$ is the Cholesky factorization of M. The matrix in (10.14) is transferred to `eigs` as a function.

```
>> type afun
```

```
function x = afun(x)
global RA RB

x = RB*(RA\(RA'\(RB'*x)));

>> global RA RB
>> RA = chol(A-sigma*M);
>> RB = chol(M);
>> [v,l1,flag]=eigs('afun',n,10,'lm',options);
Iteration 1: a few Ritz values of the 20-by-20 matrix:
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0

Iteration 2: a few Ritz values of the 20-by-20 matrix:
    3.3030
    3.5380
    4.7399
    6.5473
    6.7754
    7.8970
   15.0071
   18.3876
   44.0721
  100.0000

Iteration 3: a few Ritz values of the 20-by-20 matrix:
    3.3040
    3.5381
    4.7399
    6.5473
    6.7754
    7.8970
   15.0071
   18.3876
   44.0721
  100.0000

>> flag

flag =

     0

>> l1 = diag(l1)

l1 =
  100.0000
   44.0721
   18.3876
   15.0071
    7.8970
    6.7754
    6.5473
    4.7399
    3.5381
    3.3040

>> sigma + 1./l1

ans =

    0.0000
    0.0127
    0.0444
    0.0566
    0.1166
    0.1376
    0.1427
    0.2010
    0.2726
    0.2927

>> norm(sigma + 1./l1 - 1)

ans =

   4.4047e-14
```

10.7 The Lanczos algorithm with thick restarts

The implicit restarting procedures discussed so far are very clever ways to get rid of unwanted directions in the search space and still keeping a Lanczos or Arnoldi basis. The latter admits to continue the iteration in a known framework. The Lanczos or Arnoldi relations hold that admit very efficient checks for convergence. The restart has the effect of altering the starting vector.

In this and the next section we discuss algorithms that work with Krylov spaces but are not restricted to Krylov or Arnoldi bases. Before continuing we make a step back and consider how we can determine if a given subspace of $\mathbb{F}^n$ is a Krylov space at all.

Let A be an n-by-n matrix and let $\mathbf{v}_1, \ldots, \mathbf{v}_k$ be linearly independent n-vectors. Is the subspace $\mathcal{V} := \text{span}\{\mathbf{v}_1, \ldots, \mathbf{v}_k\}$ a Krylov space, i.e., is there a vector $\mathbf{q} \in \mathcal{V}$ such that $\mathcal{V} = \mathcal{K}_k(A, \mathbf{q})$? The following theorem gives the answer [3, 10].

Theorem 10.1 *$\mathcal{V} = span\{\mathbf{v}_1, \ldots, \mathbf{v}_k\}$ is a Krylov space if and only if there is a k-by-k matrix M such that*

$$R := AV - VM, \qquad V = [\mathbf{v}_1, \ldots, \mathbf{v}_k], \tag{10.15}$$

has rank one and $span\{\mathbf{v}_1, \ldots, \mathbf{v}_k, \mathcal{R}(R)\}$ has dimension $k + 1$.