Chapter 4

Calculation of the Key Parameters

Contents

4.1 The Numerical Problem

The key values in MATSTAB are the eigenvalue λ and the eigenvector $\mathbf{e}_i$ in 2.25. In order to calculate λ and $\mathbf{e}_i$ one must solve the generalized eigenvalue problem 2.15. Using the standard nomenclature from linear algebra textbooks [30], 2.15 can be written as follows.

$$\mathbf{Ae} = \lambda \mathbf{Be} \tag{4.1}$$

$$\mathbf{A} \in C^{nxn}, \quad \mathbf{B} = \begin{bmatrix} 1 & 0 & & \dots & & 0 \\ 0 & \ddots & & & & \\ & & 1 & & & \\ \vdots & & & 0 & & \vdots \\ & & & & \ddots & \\ 0 & & & \dots & & 0 \end{bmatrix}$$

As stated earlier, the analytical solution for 4.1 is well known and can be derived with various decomposition methods [30]. However, the computational effort for large matrices grows with n^2 and becomes very soon infeasible.

In the case where only one or a few eigenvalue/eigenvector pairs are needed, iterative methods are a practical and fast solution. There are numerous efficient algorithms available for reasonable sized systems e.g. [30], [100], [90], [23], [53] and many others. Some robust and general methods are also directly available as FORTRAN programs or modules like ARPACK [50], NSPCG [69] and others.

However, to solve 2.25 an algorithm must fulfill the following requirements in a efficient way.

- Handle a *non-symmetric* matrix **A**
- Handle the *generalized* eigenvalue problem
- Handle a *very large* matrix **A**
- Handle a *complex* matrix **A**
- Be applicable for a *sparse* matrix **A**

This list is ordered in decreasing difficulty and rules out most of the published algorithms. From the remaining possibilities, the LANCZOS method [20],[75] and the ARNOLDI iteration [37],[89] are the most promising. Nevertheless, these state-of-the-art methods were too general to solve 2.15 efficiently. MATSTAB therefore uses a tailored subspace method that takes full advantage of the known and fixed structure of **A**, combined with an extended version of Newton's method. Subspace methods and Newton's method are not new in this field, see [80] and [30]. However, their combination does not appear in the literature.

Basically the matrix $\mathbf{A}$ is divided into suitable sub-matrices $\mathbf{A}_i$ that form a set of linear sets of equations. The sub-matrices are chosen in a way that it becomes easier to solve the resulting set of equations. Unfortunately, it is not possible to decouple the subsets completely and therefore an outer iteration over the sub-solutions is necessary. The details of this approach are described in the six sections of this chapter. The first two sections introduce the standard numerical methods and their modification for MATSTAB while the later four sections describe the algorithms specifically designed for MATSTAB. Please note, that in a section dealing with numerical methods, $\mathbf{A}$ stands for any matrix while speaking about MATSTAB, $\mathbf{A}$ stands for the main matrix in the linearized BWR stability problem.

Structure of this chapter:

1. Solving $\mathbf{Ax} = \mathbf{b}$ for a very large $\mathbf{A}$
2. Solving $\mathbf{Ax} = \lambda\mathbf{Bx}$ for specific λ's
3. Partitioning into subspaces
4. The MATSTAB algorithm for the global mode
5. Calculating the left eigenvector
6. The MATSTAB algorithm for the regional mode

4.2 Solving $\mathbf{Ax} = \mathbf{b}$ for a Very Large Matrix $\mathbf{A}$

Whatever algorithm is used to solve a large eigenvalue problem, one must be able to solve efficiently matrix equations of the form

$$\mathbf{Ax} = \mathbf{b} \tag{4.2}$$

The straightforward approach would be to invert $\mathbf{A}$, but the algorithms to invert a matrix are computational very expensive and only recommendable if the explicit answer for $\mathbf{A}^{-1}$ is needed. To solve 4.2 for $\mathbf{x}$, $\mathbf{A}^{-1}$ is of no interest. Depending on the size and structure of the matrix, MATSTAB uses either Gaussian elimination to solve equation sets with an easy structure, the $\mathbf{LU}$ decomposition for a moderate structured or the iterative conjugate-gradient method for a difficult structured equation set 4.2.

4.2.1 LU - Decomposition

Any square matrix $\mathbf{A}$ can be decomposed into a lower and upper triangular matrix.

$$\mathbf{A} = \mathbf{LU} \tag{4.3}$$

$$\mathbf{L} = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ * & 1 & 0 & & \vdots \\ * & \ddots & \ddots & \ddots & 0 \\ \vdots & & * & 1 & 0 \\ * & \dots & * & * & 1 \end{bmatrix} \qquad \mathbf{U} = \begin{bmatrix} * & * & \dots & * & * \\ 0 & * & & & \vdots \\ 0 & \ddots & \ddots & & * \\ \vdots & & 0 & * & * \\ 0 & \dots & 0 & 0 & * \end{bmatrix} \qquad * \in C$$

Where **L** is the lower triangular matrix and **U** is the upper triangular matrix. The equation

$$\mathbf{Ax} = \mathbf{LUx} = \mathbf{b} \tag{4.4}$$

can thus be solved with

$$\mathbf{x} = \mathbf{U}^{-1}\mathbf{L}^{-1}\mathbf{b} \tag{4.5}$$

Because of the structure of **L** and **U**, equation 4.4 is very efficiently solvable with backward substitution. It is not necessary to calculate $\mathbf{L}^{-1}$ or $\mathbf{U}^{-1}$. The major computational work lies in the decomposition 4.3.

To emphasize the fact that $\mathbf{L}^{-1}$ and $\mathbf{U}^{-1}$ are not calculated, 4.5 is written as

$$\mathbf{x} = \mathbf{U}\backslash(\mathbf{L}\backslash\mathbf{b}) \tag{4.6}$$

or

$$\mathbf{x} = \mathbf{A} \underset{\text{LU}}{\backslash} \mathbf{b} \tag{4.7}$$

To illustrate the problems of the **LU** decomposition, the real matrices encountered in MATSTAB are discussed. Figure 4.1 shows the structure of the matrix $\mathbf{A}_{tu}$

$$\mathbf{A}_{tu} = \begin{bmatrix} \mathbf{u}_{t,1}^T & \mathbf{0} \\ \mathbf{A}_t - \lambda_k \mathbf{B}_t & -\mathbf{B}_t \mathbf{e}_{t,k} \end{bmatrix}$$

which is used in step 2 of algorithm 4.6. The blank part of the matrix represents zero elements and the remaining part represents non-zero elements. The large *black triangle* is not as densely filled with numbers as it looks. In fact the dots should form a grid with two black dots every 50 points, but the resolution of the Figure suppresses the 48 white points in between.

Figures 4.2 and 4.3 show the **L** and **U** factors of $\mathbf{A}_{tu}$. The number of non-zero values in **L** and **U** is approximately 3 times higher than the number of non-zeros in $\mathbf{A}_{tu}$. Hence, the construction of this sub-matrix succeeded and the sparsity is in this case preserved.

Although the **LU** decomposition is much faster than the true inversion of $\mathbf{A}_{tu}$, some drawbacks remain. A very large **A** is normally, and especially in our case, truly sparse. Less than 0.1% of all entries in $\mathbf{A}_{tu}$ are different from zero. The algorithms take profit of this property and conduct no calculations for the ”zero” parts of the resulting matrix. Unfortunately, the **L** and **U** factors of a sparse matrix are not necessarily sparse too. This fact increases extensively the computing time, and more crucial, the memory requirements of **L** and **U**.

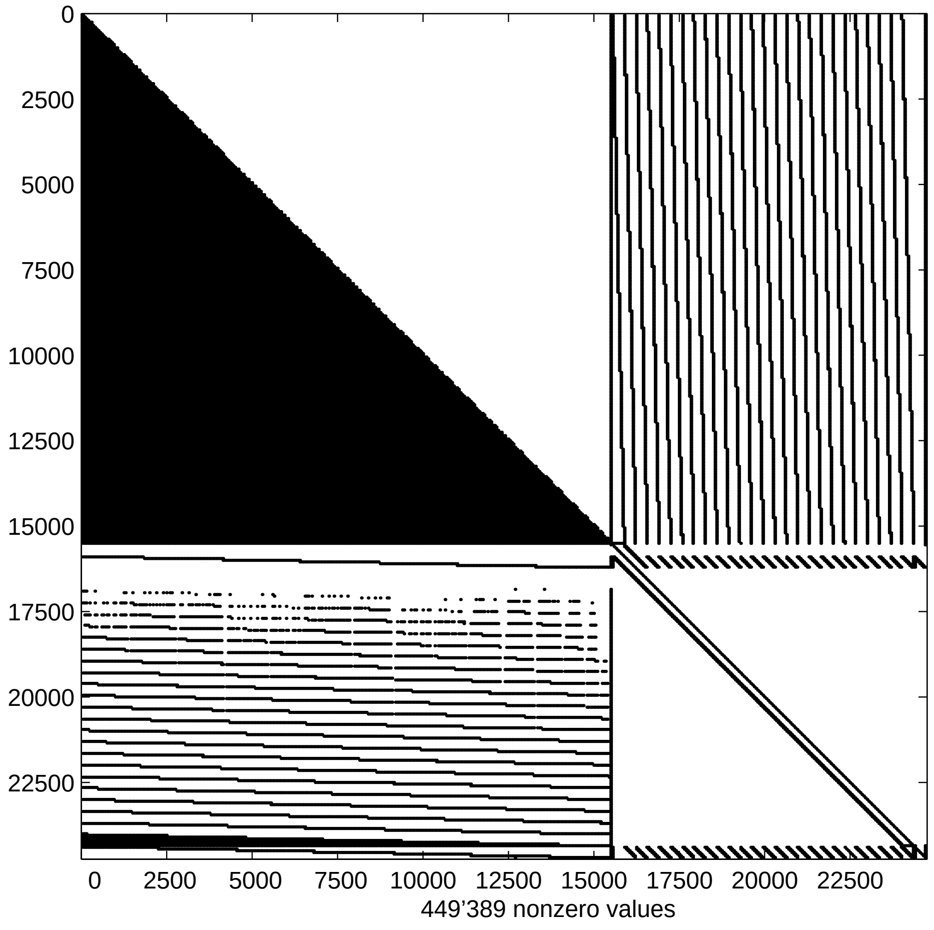
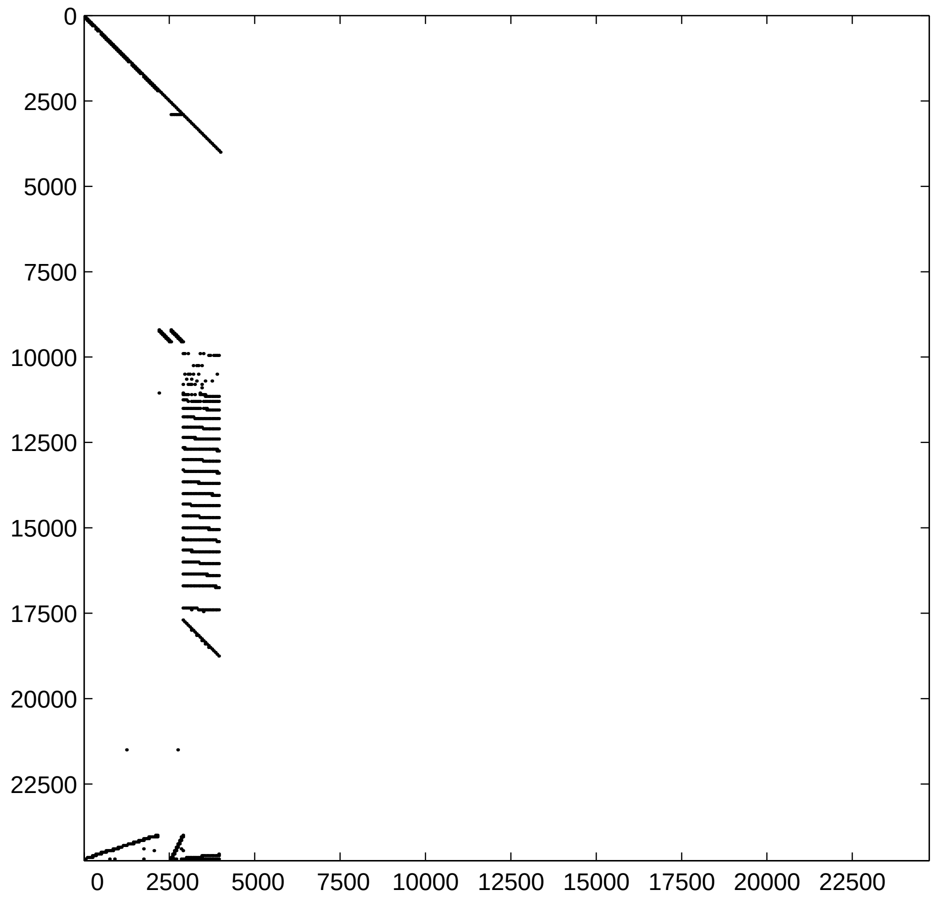
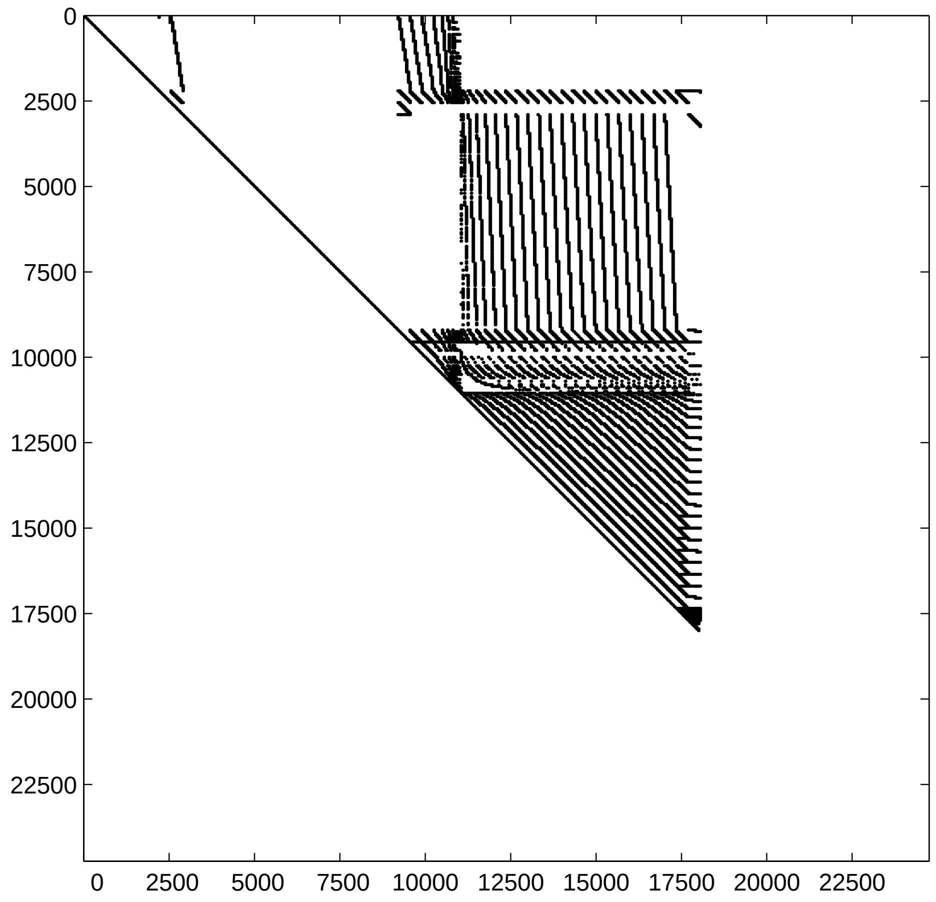


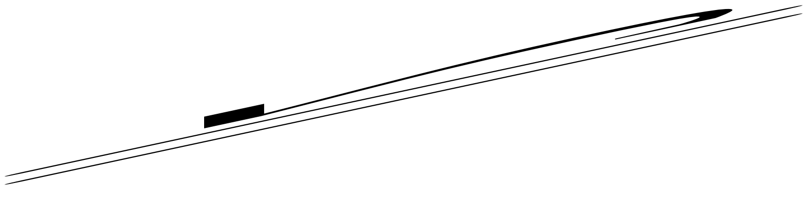
Figure 4.1: Structure of the Matrix $\mathbf{A}_{tu}$ Used in Step 2 of Algorithm 4.6

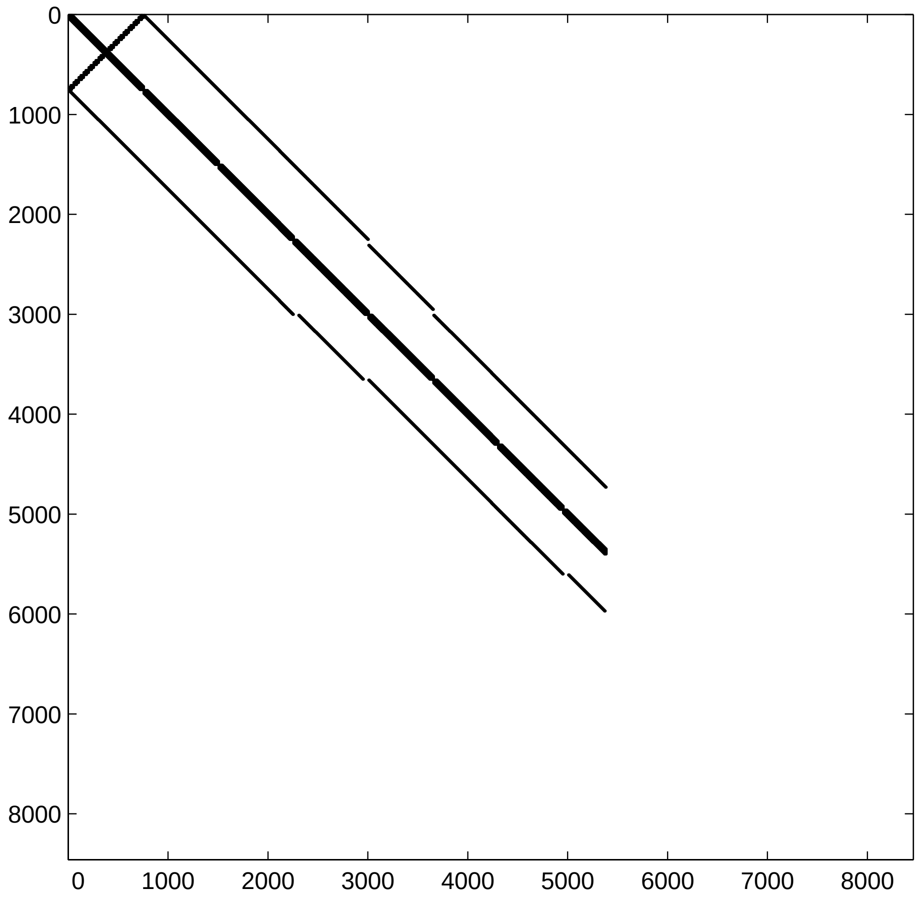
The requirements for a decomposition of some matrices appearing in MATSTAB are prohibitively high. Figures 4.4, 4.5 and 4.6 show the **LU** decomposition of the matrix $\mathbf{A}_n$, the neutronic part of the matrix **A**. Even though the matrix $\mathbf{A}_n$ is relatively small, and the structure of the matrix is more appropriate for an efficient numerical treatment than the structure of $\mathbf{A}_{tu}$, the loss of sparsity is very significant. The non-zeros and, therefore, the memory requirements are multiplied by a factor of 330. So the **LU** decomposition of $\mathbf{A}_n$ takes 250MB of RAM and nearly an hour of computing time. The reason for the different behavior of $\mathbf{A}_{tu}$ and $\mathbf{A}_n$ lies in the physical structure of the underling equations. The thermal hydraulics properties described in $\mathbf{A}_{tu}$ are in most cases only coupled to the up- and downstream neighbors. The neutronic properties of $\mathbf{A}_n$ on the opposite, are coupled with all six spatial neighbors. It is this interconnection with many other nodes, that makes the decomposition of $\mathbf{A}_n$ computationally expensive. To overcome this technical bottle neck, MATSTAB uses the iterative conjugate gradient method explained in the next section to solve equations of this type.


0
2500
5000
7500
10000
12500
15000
17500
20000
22500



is replaced by the one-dimensional minimization problem




0
1000
2000
3000
4000
5000
6000
7000
8000
0
1000
2000
3000
4000
5000
6000
7000
8000

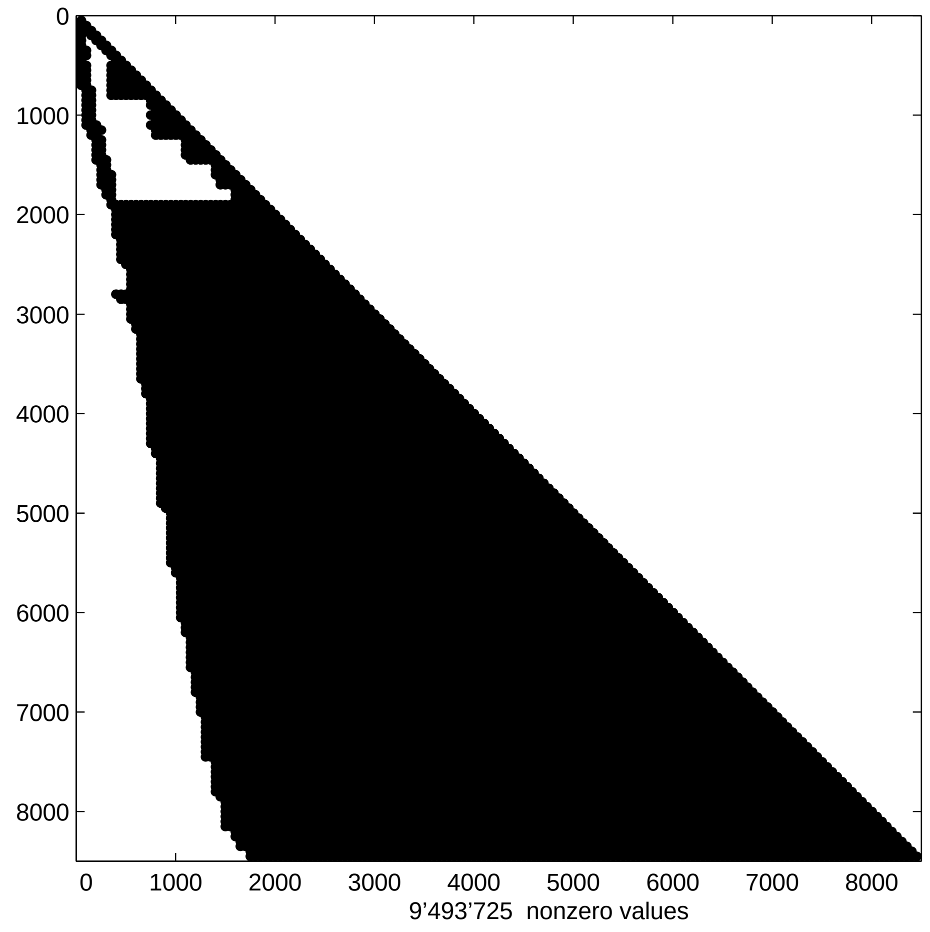


Figure 4.5: Lower Triangular Part of the Matrix $\mathbf{A}_n$

4.3 The Calculation of a Specific Eigenvalue/Eigenvector

From a physical point of view, only the few dominant eigenvalues are of interest. Since the computation time for the eigenvalue/eigenvector is significant, not to say crucial, MATSTAB uses methods which calculate only the dominant eigenvalues. These eigenvalues can be singled out because from equation 2.27 follows that a high decay ratio is equivalent with a large real part of the eigenvalue, therefore the eigenvalue with the largest real part is dominating. The following paragraphs give a stepwise approach to the calculation of selected eigenvalues. These methods are by no means new, but they are necessary to understand the final algorithm 4.6.

The mechanism behind calculating one single eigenvalue/eigenvector pair $\lambda/\mathbf{e}$ for

$$\mathbf{Ae} = \lambda\mathbf{e} \tag{4.13}$$

can be best understood in the power method.

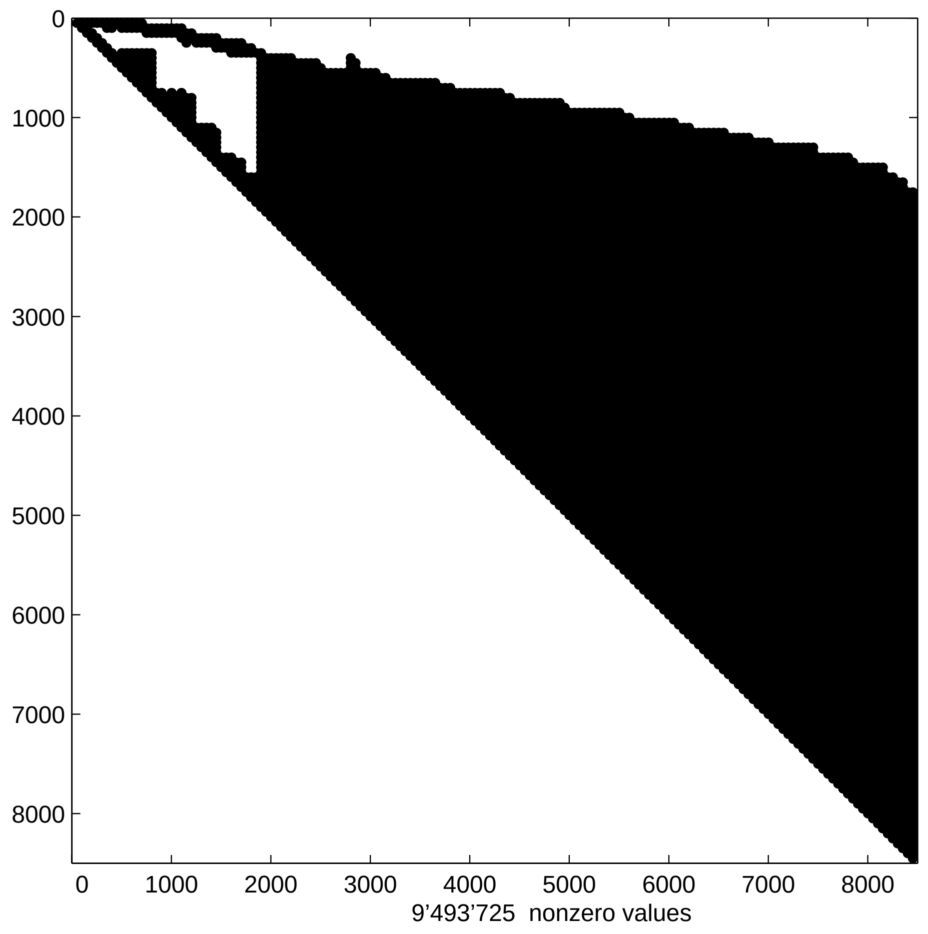


Figure 4.6: Upper Triangular Part of the Matrix $\mathbf{A}_n$

4.3.1 Power Method

The *largest* (in magnitude) eigenvalue and its right eigenvector for the matrix $\mathbf{A} \in \mathbf{C}^{nxn}$ are found with the iteration:

```
Algorithm 4.2: Power Method
```

1. $\mathbf{e}_0$ any starting guess
2. $\mathbf{e}_{k+1} = \mathbf{A}\mathbf{e}_k$
3. $\lambda_{k+1} = \frac{\mathbf{e}_{k+1}^T \mathbf{A}\mathbf{e}_{k+1}}{\mathbf{e}_{k+1}^T \mathbf{e}_{k+1}}$
4. If $||\mathbf{A}\mathbf{e}_{k+1} - \lambda_{k+1}\mathbf{e}_{k+1}|| > tol$ goto step 2

In order to understand why this iteration works, assume that **A** has distinct eigenvalues, with $|_1\lambda| < \cdots < |_i\lambda| < \cdots < |_n\lambda|$. Since **A** then has a full set of eigenvectors, ${}_1\mathbf{e}, {}_2\mathbf{e}, \ldots, {}_n\mathbf{e}$ forming a base, the start vector can be written as

$$\mathbf{e}_0 = \sum_{i=1}^{n} {}_ic_i\mathbf{e} \tag{4.14}$$

The vector $\mathbf{e}_k$, resulting after k iterations can be written as,

$$\mathbf{e}_k = \mathbf{A}^k\mathbf{e}_0 = \sum_{i=1}^{n} {}_ic_i\lambda^k{}_i\mathbf{e} \tag{4.15}$$

Clearly, $\mathbf{e}_k$ will be more and more dominated by ${}_n\mathbf{e}$. The convergence of the power method is linear, with a convergence rate proportional to $\frac{{}_{n-1}\lambda}{{}_n\lambda}$. The computational effort for this algorithm is very small, since the only matrix operations involved are multiplications. However, the eigenvalues with the largest absolute value are not of any interest from a physical point of view. The algorithm has therefore to be modified to calculate the eigenvalues with the desired properties (e.g. smallest real part).

4.3.2 Inverse Iteration

The next simple step is the modification of algorithm 4.2 for the calculation of the *smallest* (in magnitude) eigenvalue.

`Algorithm 4.3: Inverse Iteration`

1. $\mathbf{e}_0$ = any starting guess
2. $\mathbf{A}\mathbf{e}_{k+1} = \mathbf{e}_k$
3. $\lambda_{k+1} = \frac{\mathbf{e}_{k+1}^T\mathbf{A}\mathbf{e}_{k+1}}{\mathbf{e}_{k+1}^T\mathbf{e}_{k+1}}$
4. If $||\mathbf{A}\mathbf{e}_{k+1} - \lambda_{k+1}\mathbf{e}_{k+1}|| > tol$ goto step 2

The major computation here is to solve $\mathbf{A}\mathbf{e}_{k+1}$ = $\mathbf{e}_k$ for $\mathbf{e}_{k+1}$ each iteration. This is somewhat expensive, but doable with either an **LU** decomposition or the conjugate gradient method. Inverse iteration is the power method applied to $\mathbf{A}^{-1}$, and consequently the convergence of the inverse iteration is linear with convergence rate proportional to $\frac{{}_1\lambda}{{}_2\lambda}$.

4.3.3 Inverse Iteration with Shift

If more than one, or a special, eigenvalue is needed, a spectral transformation is necessary.

```
Algorithm 4.4: Inverse Iteration with shift

1. $\mathbf{e}_0$ = any starting guess
   $\lambda_0$ = any starting guess
2. $(\mathbf{A} - \lambda_k \mathbf{I})\mathbf{e}_{k+1} = \mathbf{e}_k$
3. $\lambda_{k+1} = \frac{\mathbf{e}_{k+1}^T \mathbf{A} \mathbf{e}_{k+1}}{\mathbf{e}_{k+1}^T \mathbf{e}_{k+1}} = \lambda_k + \frac{\mathbf{e}_{k+1}^T \mathbf{e}_k}{||\mathbf{e}_{k+1}||^2}$
4. If $||\mathbf{A}\mathbf{e}_{k+1} - \lambda_{k+1}\mathbf{e}_{k+1}|| > tol$ got to step 2
```

Note that it is not necessary to update λ_k in ($\mathbf{A}$ - $\lambda_k\mathbf{I}$) each iteration. In general, the convergence will be faster if one does, but on the other hand without updating λ_k, the **LU** decomposition of ($\mathbf{A}$ - $\lambda_k\mathbf{I}$) from the previous iteration can be used again. The convergence is cubic for the symmetric case.

4.3.4 Newton's Method

Due to the limited machine precision, the algorithm 4.4 has significant error bounds for a very large **A**. Therefore a combination with Newton's method is introduced [80].

The quality of an approximate solution $\mathbf{x}_k$ of $\mathbf{f}(\mathbf{x}) = \mathbf{0}$ may be improved by the update

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \nabla \mathbf{f}(\mathbf{x}_k)^{-1} \mathbf{f}(\mathbf{x}_k) \tag{4.16}$$

Because the magnitude of an eigenvector is not defined, one equation has to be added to 4.13 to define a unique function $\boldsymbol{f}$ for the eigenvalue problem, e.g.

$$\mathbf{u}_1^T \mathbf{e} = \mathbf{e}(1) = 1 \quad \mathbf{u}_1^T = [1, 0, \dots, 0] \tag{4.17}$$

The function $\boldsymbol{f}$ therefore is defined as follows.

$$\mathbf{f}(\lambda, \mathbf{e}) = \mathbf{0} \quad \Leftrightarrow \quad \begin{vmatrix} \mathbf{u}_1^T \mathbf{e} - 1 & = & 0 \\ (\mathbf{A} - \lambda \mathbf{I})\mathbf{e} & = & \mathbf{0} \end{vmatrix} \tag{4.18}$$

The Newton update

$$\begin{bmatrix} \mathbf{e}_{k+1} \\ \lambda_{k+1} \end{bmatrix} = \begin{bmatrix} \mathbf{e}_k \\ \lambda_k \end{bmatrix} - \nabla \mathbf{f}(\lambda_k, \mathbf{e}_k)^{-1} \mathbf{f}(\lambda_k, \mathbf{e}_k) \tag{4.19}$$

for the inverse iteration with shift becomes

$$\begin{bmatrix} \mathbf{e}_{k+1} \\ \lambda_{k+1} \end{bmatrix} = \begin{bmatrix} \mathbf{e}_k \\ \lambda_k \end{bmatrix} - \begin{bmatrix} \mathbf{u}_1^T & 0 \\ \mathbf{A} - \lambda_k \mathbf{I} & -\mathbf{e}_k \end{bmatrix} \Bigg\backslash \begin{bmatrix} \mathbf{u}_1^T \mathbf{e}_k - 1 \\ (\mathbf{A} - \lambda_k \mathbf{I})\mathbf{e}_k \end{bmatrix} \tag{4.20}$$

The meaning of the backslash used above is explained in the comment to equation 4.6.

$$\begin{bmatrix} \mathbf{u}_1^T & 0 \\ \mathbf{A}-\lambda_k\mathbf{I} & -\mathbf{e}_k \end{bmatrix} \begin{bmatrix} \mathbf{e}_{k+1}-\mathbf{e}_k \\ \lambda_{k+1}-\lambda_k \end{bmatrix} = -\begin{bmatrix} 0 \\ (\mathbf{A}-\lambda_k\mathbf{I})\mathbf{e}_k \end{bmatrix} \tag{4.21}$$

Equation 4.21 can be extended naturally for the generalized eigenvalue problem

$$\mathbf{A}\mathbf{e} = \lambda\mathbf{B}\mathbf{e} \tag{4.22}$$

$$f(\lambda,\mathbf{e}) = 0 \quad \Leftrightarrow \quad \left| \begin{array}{lcl} \mathbf{u}_1^T\mathbf{e}-1 & = & 0 \\ (\mathbf{A}-\lambda\mathbf{B})\mathbf{e} & = & 0 \end{array} \right| \tag{4.23}$$

$$\begin{bmatrix} \mathbf{u}_1^T & 0 \\ \mathbf{A}-\lambda_k\mathbf{B} & -\mathbf{B}\mathbf{e}_k \end{bmatrix} \begin{bmatrix} \mathbf{e}_{k+1}-\mathbf{e}_k \\ \lambda_{k+1}-\lambda_k \end{bmatrix} = -\begin{bmatrix} 0 \\ (\mathbf{A}-\lambda_k\mathbf{B})\mathbf{e}_k \end{bmatrix} \tag{4.24}$$

The iteration process for 4.24 becomes now:

`Algorithm 4.5: Generalized Newton's Method`

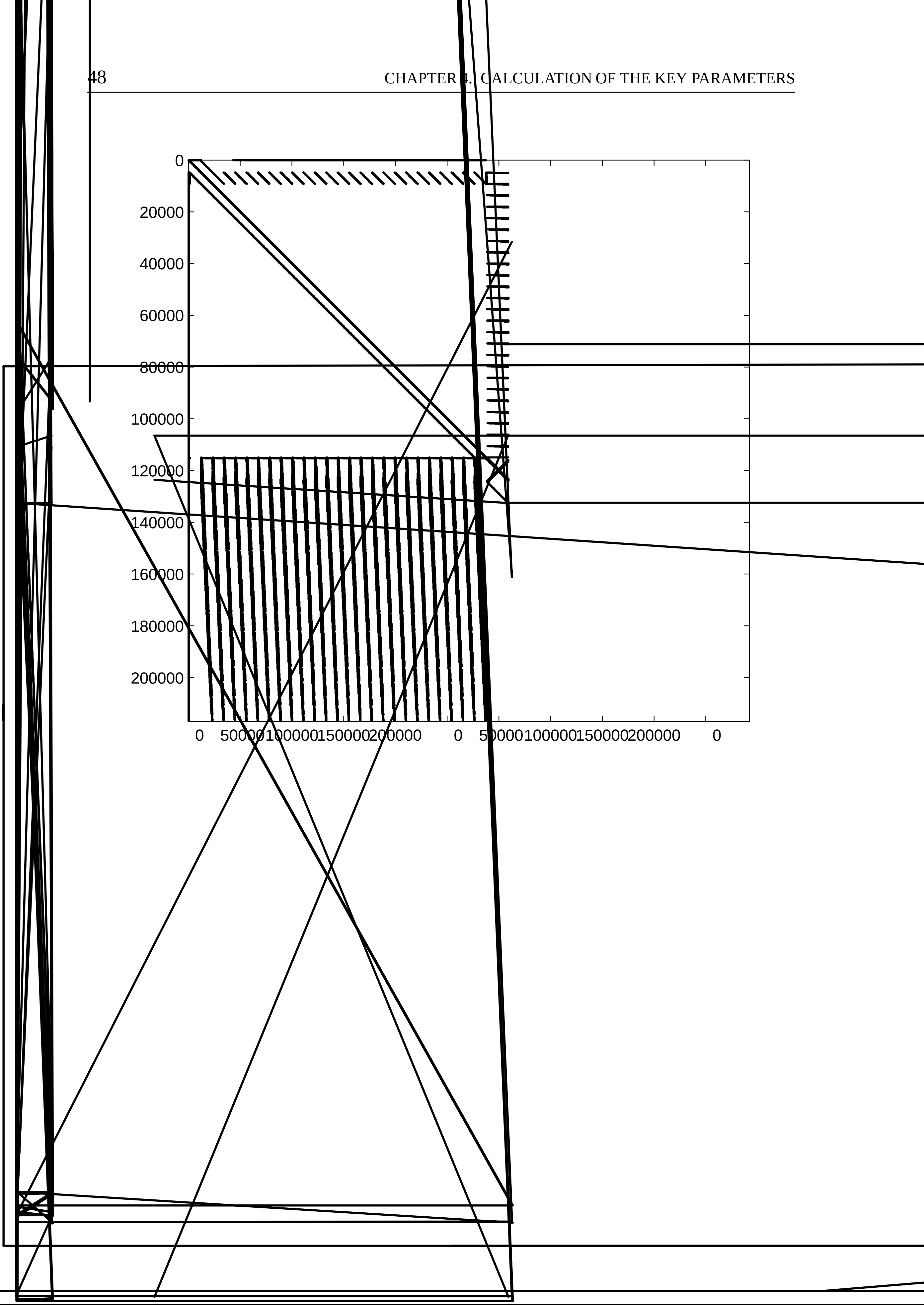
1. $\mathbf{e}_0$ any starting guess
   λ_0 any starting guess
2. $\begin{bmatrix} \Delta\mathbf{e}_{k+1} \\ \Delta\lambda_{k+1} \end{bmatrix} = \begin{bmatrix} \mathbf{e}_{k+1}-\mathbf{e}_k \\ \lambda_{k+1}-\lambda_k \end{bmatrix} = \begin{bmatrix} \mathbf{u}_1^T & 0 \\ \mathbf{A}-\lambda_k\mathbf{B} & -\mathbf{B}_t\mathbf{e}_k \end{bmatrix} \Bigg\backslash \begin{bmatrix} \mathbf{u}_1^T\mathbf{e}_k-1 \\ (\mathbf{A}-\lambda_k\mathbf{B})\mathbf{e}_k \end{bmatrix}$
3. $\mathbf{e}_{k+1} = \mathbf{e}_k + \Delta\mathbf{e}$
4. $\lambda_{k+1} = \lambda_k + \Delta\lambda$
5. If $||\Delta\mathbf{e}|| > tol$ goto step 2

4.4 Partitioning Into Subspaces

The direct implementation of algorithm 4.5 fails due to the sheer size of **A** (see Figure 4.7) in MATSTAB. Step 2 of 4.24 is not feasible in reasonable time. Therefore the matrix **A** is divided into suitable sub-matrices.

The matrices **A**, **B** and the vector **e** are split up as follows.

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_t & \mathbf{A}_{tj} & \mathbf{0} & \mathbf{A}_{tq} & \mathbf{A}_{tf} \\ \mathbf{A}_{jt} & \mathbf{A}_j & \mathbf{0} & \mathbf{0} & \mathbf{A}_{jf} \\ \mathbf{A}_{NT} & \mathbf{0} & \mathbf{A}_N & \mathbf{0} & \mathbf{0} \\ \mathbf{A}_{qt} & \mathbf{0} & \mathbf{A}_{qn} & -\mathbf{I} & \mathbf{0} \\ \mathbf{A}_{ft} & \mathbf{A}_{fj} & \mathbf{0} & \mathbf{A}_{fq} & \mathbf{A}_f \end{bmatrix} \tag{4.25}$$


0
20000
40000
60000
80000
100000
120000
140000
160000
180000
200000
0
50000
100000
150000
200000
0
50000
100000
150000
200000
0

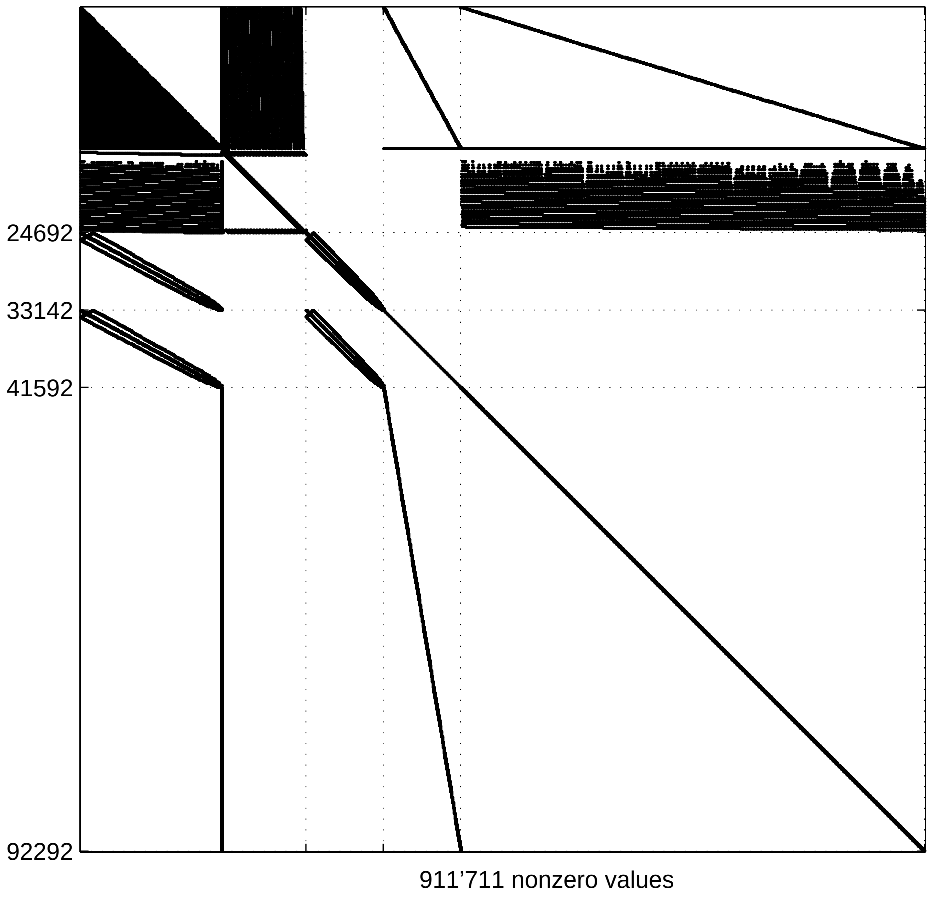


Figure 4.8: The Matrix **A** Divided Into Physical Subspaces

The capital letters N and NT in 4.25 symbolize that the corresponding matrices are complex.

$$\mathbf{A}_N = \mathbf{A}_n + i\mathbf{A}_{nIm} \quad , \mathbf{A}_{NT} = \mathbf{A}_{nt} + i\mathbf{A}_{ntIm} \tag{4.27}$$

The complex part of the neutronics was introduced by A.33.

Figure 4.8 shows the partition of **A** corresponding to 4.25.

Equation 4.28 shows the generalized eigenvalue problem 2.15 as formulated in MATSTAB.

$$\begin{bmatrix} \lambda\mathbf{B}_t\mathbf{e}_t \\ \lambda\mathbf{B}_j\mathbf{e}_j \\ \mathbf{0} \\ \mathbf{0} \\ \lambda\mathbf{B}_f\mathbf{e}_f \end{bmatrix} = \begin{bmatrix} \mathbf{A}_t & \mathbf{A}_{tj} & \mathbf{0} & \mathbf{A}_{tq} & \mathbf{A}_{tf} \\ \mathbf{A}_{jt} & \mathbf{A}_j & \mathbf{0} & \mathbf{0} & \mathbf{A}_{jf} \\ \mathbf{A}_{NT} & \mathbf{0} & \mathbf{A}_N & \mathbf{0} & \mathbf{0} \\ \mathbf{A}_{qt} & \mathbf{0} & \mathbf{A}_{qn} & -\mathbf{I} & \mathbf{0} \\ \mathbf{A}_{ft} & \mathbf{A}_{fj} & \mathbf{0} & \mathbf{A}_{fq} & \mathbf{A}_f \end{bmatrix} \begin{bmatrix} \mathbf{e}_t \\ \mathbf{e}_j \\ \mathbf{e}_n \\ \mathbf{e}_q \\ \mathbf{e}_f \end{bmatrix} \tag{4.28}$$

Equation 4.28 is equivalent to the following set of equations.

$$\lambda \mathbf{B}_t \mathbf{e}_t = \mathbf{A}_t \mathbf{e}_t + \mathbf{A}_{tj} \mathbf{e}_j + \mathbf{A}_{tq} \mathbf{e}_q + \mathbf{A}_{tf} \mathbf{e}_f \tag{4.29}$$

$$\lambda \mathbf{B}_j \mathbf{e}_j = \mathbf{A}_{jt} \mathbf{e}_t + \mathbf{A}_j \mathbf{e}_j + \mathbf{A}_{jf} \mathbf{e}_q \tag{4.30}$$

$$\mathbf{0} = \mathbf{A}_{NT} \mathbf{e}_t + \mathbf{A}_N \mathbf{e}_n \tag{4.31}$$

$$\mathbf{0} = \mathbf{A}_{qt} \mathbf{e}_t + \mathbf{A}_{qn} \mathbf{e}_n - \mathbf{e}_q \tag{4.32}$$

$$\lambda \mathbf{B}_f \mathbf{e}_f = \mathbf{A}_{ft} \mathbf{e}_t + \mathbf{A}_{fj} \mathbf{e}_j + \mathbf{A}_{fq} \mathbf{e}_q + \mathbf{A}_f \mathbf{e}_f \tag{4.33}$$

4.5 The Global Mode

This section explains how Newton's method is applied to the subspace approach introduced above. Since normally the global oscillation mode of the reactor is dominant MATSTAB always starts with calculating this mode. If requested, the eigenvalues/eigenvectors for regional oscillations are calculated afterwards.

4.5.1 The Starting Guesses

Since we use iterative methods, the calculation starts with a guess of the eigenvalue derived from a guess of the decay ratio and from the expected frequency of the oscillation. Equation 2.27 on page 17 is used to transform these values into an eigenvalue.

The starting guess $\mathbf{e}_0$ of the eigenvector is constructed automatically from λ_0 and the starting guess $\mathbf{e}_{n,0}$ for $\mathbf{e}_n$. The vector $\mathbf{e}_{n,0}$ is created from the power distribution of the steady state solution. This implies, that the shape of $\mathbf{e}_n$ is similar to the shape of the power density in the reactor! The equations implemented below, are derived from equations 4.29ff.

$$\begin{array}{lrcl}
\text{Preliminary step:} & \mathbf{e}_{q,0} & = & \mathbf{A}_{qn} \mathbf{e}_{n,0} \\
 & \mathbf{e}_{f,0} & = & (\lambda_0 \mathbf{B}_f - \mathbf{A}_f) \underset{\text{GE}}{\backslash} (\mathbf{A}_{fq} \mathbf{e}_{q,0}) \\
 & & & \\
\text{Construction step:} & \mathbf{e}_{t,0} & = & (\lambda_0 \mathbf{B}_t - \mathbf{A}_t) \underset{\text{LU}}{\backslash} (\mathbf{A}_{tf} \mathbf{e}_{f,0} + \mathbf{A}_{tq} \mathbf{e}_{q,0}) \\
 & \mathbf{e}_{n,0} & = & \mathbf{A}_n \underset{\text{CG}}{\backslash} (\mathbf{A}_{NT} \mathbf{e}_{t,0} + i \mathbf{A}_{nIm} \mathbf{e}_{n,0}) \\
 & \mathbf{e}_{q,0} & = & A_{qn} \mathbf{e}_{n,0} + \mathbf{A}_{qt} \mathbf{e}_{t,0} \\
 & \mathbf{e}_{f,0} & = & (\lambda_0 \mathbf{B}_f - \mathbf{A}_f) \underset{\text{GE}}{\backslash} (\mathbf{A}_{fq} \mathbf{e}_{q,0} + \mathbf{A}_{ft} \mathbf{e}_{t,0})
\end{array} \tag{4.34}$$

The symbols $\underset{\text{GE}}{\backslash}$, $\underset{\text{LU}}{\backslash}$ and $\underset{\text{CG}}{\backslash}$ are to be understood in the way, that the set of equations is solved either with Gaussian elimination, **LU** decomposition or conjugate gradient methods.

The construction step improves the guess for $\mathbf{e}_n$ and is therefore repeated three times to improve the overall quality of the starting guess for the eigenvector. There is no sense to do more iterations since λ_0 is not updated and the algorithm produces only a starting guess.

4.5.2 The Main Iteration

The main iteration process is a combination of Newton's method and the iteration over the subspaces.

Newton's method is applied to the thermal-hydraulic part of the problem.

$$\mathbf{f}(\lambda,\mathbf{e}_t)=\mathbf{0} \quad \Leftrightarrow \quad \left|\begin{array}{cc} \mathbf{u}_{t,1}^T\mathbf{e}_t-1 & =0 \\ (\mathbf{A}_t-\lambda\mathbf{B}_t)\mathbf{e}_t+\mathbf{A}_{tf}\mathbf{e}_f+\mathbf{A}_{tq}\mathbf{e}_q & =\mathbf{0}\end{array}\right| \tag{4.35}$$

This leads to $\Delta\lambda_{k+1}$ and $\Delta\mathbf{e}_{t,k+1}$ which are then used to create $\mathbf{e}_{n,k+1}, \mathbf{e}_{q,k+1}$, and $\mathbf{e}_{f,k+1}$. The algorithm reads now as follows.

```
Algorithm 4.6: Newton's Method with subspaces

1. $\mathbf{e}_{n,0}$ from power distribution
   $\lambda_0$ from DR guess and frequency guess
2. $\begin{bmatrix}\Delta\mathbf{e}_{t,k+1}\\ \Delta\lambda_{k+1}\end{bmatrix}=\begin{bmatrix}\mathbf{u}_{t,1}^T & 0\\ \mathbf{A}_t-\lambda_k\mathbf{B}_t & -\mathbf{B}_t\mathbf{e}_{t,k}\end{bmatrix}\underset{LU}{\Big\backslash}\begin{bmatrix}\mathbf{u}_{t,1}^T\mathbf{e}_{t,k}-1\\ (\mathbf{A}_t-\lambda_k\mathbf{B}_t)\mathbf{e}_{t,k}+\mathbf{A}_{tf}\mathbf{e}_{f,k}+\mathbf{A}_{tq}\mathbf{e}_{q,k}\end{bmatrix}$
3. $\mathbf{e}_{k+1}=\mathbf{e}_k+\Delta\mathbf{e}_{k+1}$
4. $\lambda_{k+1}=\lambda_k+\Delta\lambda_{k+1}$
5. $\mathbf{e}_{n,k+1}=\mathbf{A}_n\underset{\text{CG}}{\backslash}(\mathbf{A}_{NT}\mathbf{e}_{t,k+1}+i\mathbf{A}_{nIm}\mathbf{e}_{n,k})$
6. $\mathbf{e}_{q,k+1}=\mathbf{A}_{qt}\mathbf{e}_{t,k+1}+\mathbf{A}_{qn}\mathbf{e}_{n,k+1}$
7. $\mathbf{e}_{f,k+1}=(\lambda_{k+1}\mathbf{B}_f-\mathbf{A}_f)\underset{\text{GE}}{\backslash}(\mathbf{A}_{ft}\mathbf{e}_{t,k+1}+\mathbf{A}_{fq}\mathbf{e}_{q,k+1})$
8. If $||\Delta\mathbf{e}||>tolerance$ goto step 2
```

4.6 The Left Eigenvector

As mentioned several times before, the decay ratio describes only one aspect of instability. Most information is stored in the eigenvectors. The right eigenvector is a natural byproduct of the eigenvalue calculation. The left eigenvector, however, has to be calculated separately. The methods used to calculate the left eigenvector are similar to the the methods used for the right eigenvector and the eigenvalue. However, the knowledge of the latter makes the process a bit simpler and faster. The starting point is the generalized left eigenvalue equation corresponding to equation 4.28.

$$\begin{bmatrix} \lambda \mathbf{B}_t \mathbf{f}_t \\ \lambda \mathbf{B}_j \mathbf{f}_j \\ 0 \\ 0 \\ \lambda \mathbf{B}_f \mathbf{f}_f \end{bmatrix} = \begin{bmatrix} \mathbf{f}_t \\ \mathbf{f}_j \\ \mathbf{f}_n \\ \mathbf{f}_q \\ \mathbf{f}_f \end{bmatrix}^T \begin{bmatrix} \mathbf{A}_t & \mathbf{A}_{tj} & \mathbf{0} & \mathbf{A}_{tq} & \mathbf{A}_{tf} \\ \mathbf{A}_{jt} & \mathbf{A}_j & \mathbf{0} & \mathbf{0} & \mathbf{A}_{jf} \\ \mathbf{A}_{NT} & \mathbf{0} & \mathbf{A}_N & \mathbf{0} & \mathbf{0} \\ \mathbf{A}_{qt} & \mathbf{0} & \mathbf{A}_{qn} & -\mathbf{I} & \mathbf{0} \\ \mathbf{A}_{ft} & \mathbf{A}_{fj} & \mathbf{0} & \mathbf{A}_{fq} & \mathbf{A}_f \end{bmatrix} \tag{4.36}$$

Transposing the equation leads to a right eigenvector problem.

$$\begin{bmatrix} \lambda \mathbf{B}_t^T \mathbf{f}_t \\ \lambda \mathbf{B}_j^T \mathbf{f}_j \\ \mathbf{0} \\ \mathbf{0} \\ \lambda \mathbf{B}_f^T \mathbf{f}_f \end{bmatrix} = \begin{bmatrix} \mathbf{A}_t^T & \mathbf{A}_{jt}^T & \mathbf{A}_{NT}^T & \mathbf{A}_{qt}^T & \mathbf{A}_{ft}^T \\ \mathbf{A}_{tj}^T & \mathbf{A}_j^T & \mathbf{0} & \mathbf{0} & \mathbf{A}_{fj}^T \\ \mathbf{0} & \mathbf{0} & \mathbf{A}_N^T & \mathbf{A}_{qn}^T & \mathbf{0} \\ \mathbf{A}_{tq}^T & \mathbf{0} & \mathbf{0} & -\mathbf{I} & \mathbf{A}_{fq}^T \\ \mathbf{A}_{tf}^T & \mathbf{A}_{jf}^T & \mathbf{0} & \mathbf{0} & \mathbf{A}_f^T \end{bmatrix} \begin{bmatrix} \mathbf{f}_t \\ \mathbf{f}_j \\ \mathbf{f}_n \\ \mathbf{f}_q \\ \mathbf{f}_f \end{bmatrix} \tag{4.37}$$

Note, that the structure of the equation system in 4.28 is quite different to that of 4.37 due to the fact, that the matrix **A** (see 4.25) is not symmetrical. Since λ is known, the only thing that stops us from solving the system of equations with standard techniques are once more, numerical problems. Nevertheless the number of equations may be reduced to a certain limit, using Gaussian elimination.

The following derivation is by no means trivial. The equations and the order of substitution have to be selected very carefully to maintain sparsity and to keep the computational effort affordable. Even more, some of the intermediate terms appearing below may not be calculated explicitly. It is, however, possible to calculate the product of this term with a vector using the fact that the multiplications of matrices and vectors are associative.

Using equation 4.37 as a starting point, following equations may be derived.

$$\mathbf{f}_f = (\lambda \mathbf{B}_f^T - \mathbf{A}_f^T) \underset{\text{LU}}{\setminus} (\mathbf{A}_{tf}^T \mathbf{f}_t + \mathbf{A}_{jf}^T \mathbf{f}_j) \tag{4.38}$$

$$\mathbf{f}_t = (\lambda \mathbf{B}_t^T - \mathbf{A}_t^T) \underset{\text{LU}}{\setminus} (\mathbf{A}_{jt}^T \mathbf{f}_j + \mathbf{A}_{NT}^T \mathbf{f}_n + \mathbf{A}_{qt}^T \mathbf{f}_q + \mathbf{A}_{ft}^T \mathbf{f}_f) \tag{4.39}$$

$$\mathbf{f}_j = (\lambda \mathbf{B}_j^T - \mathbf{A}_j^T) \underset{\text{GE}}{\setminus} (\mathbf{A}_{tj}^T \mathbf{f}_t + \mathbf{A}_{fj}^T \mathbf{f}_f) \tag{4.40}$$

Inserting 4.38 into 4.39 and 4.40 leads to

$$\begin{aligned} \mathbf{f}_t &= (\lambda \mathbf{B}_t^T - \mathbf{A}_t^T) \underset{\text{LU}}{\setminus} [\mathbf{A}_{jt}^T \mathbf{f}_j + \mathbf{A}_{NT}^T \mathbf{f}_n + \mathbf{A}_{qt}^T \mathbf{f}_q + \mathbf{A}_{ft}^T (\lambda \mathbf{B}_f^T - \mathbf{A}_f^T) \underset{\text{LU}}{\setminus} (\mathbf{A}_{tf}^T \mathbf{f}_t + \mathbf{A}_{jf}^T \mathbf{f}_j)] \\ &= (\lambda \mathbf{B}_t^T - \mathbf{A}_t^T - \mathbf{A}_{fttf}) \underset{\text{LU}}{\setminus} [(\mathbf{A}_{jt}^T + \mathbf{A}_{ftjf}) \mathbf{f}_j + \mathbf{A}_{NT}^T \mathbf{f}_n + \mathbf{A}_{qt}^T \mathbf{f}_q] \\ &= \mathbf{A}_{bj} \mathbf{f}_j + \mathbf{A}_{bNT} \mathbf{f}_n + \mathbf{A}_{bq} \mathbf{f}_q \end{aligned} \tag{4.41}$$

where

$$\begin{aligned}
\mathbf{A}_{fttf} &= \mathbf{A}_{ft}^T(\lambda\mathbf{B}_f^T - \mathbf{A}_f^T) \underset{\text{LU}}{\backslash} \mathbf{A}_{tf}^T \\
\mathbf{A}_{ftjf} &= \mathbf{A}_{ft}^T(\lambda\mathbf{B}_f^T - \mathbf{A}_f^T) \underset{\text{LU}}{\backslash} \mathbf{A}_{jf}^T \\
\mathbf{A}_{bj} &= (\lambda\mathbf{B}_t^T - \mathbf{A}_t^T - \mathbf{A}_{fttf}) \underset{\text{LU}}{\backslash} (\mathbf{A}_{jt}^T + \mathbf{A}_{ftjf}) \\
\mathbf{A}_{bNT} &= (\lambda\mathbf{B}_t^T - \mathbf{A}_t^T - \mathbf{A}_{fttf}) \underset{\text{LU}}{\backslash} \mathbf{A}_{NT}^T \\
\mathbf{A}_{bq} &= (\lambda\mathbf{B}_t^T - \mathbf{A}_t^T - \mathbf{A}_{fttf}) \underset{\text{LU}}{\backslash} \mathbf{A}_{qt}^T
\end{aligned}$$

and

$$\begin{aligned}
\mathbf{f}_j &= (\lambda\mathbf{B}_j^T - \mathbf{A}_j^T) \underset{\text{GE}}{\backslash} [\mathbf{A}_{fj}^T(\lambda\mathbf{B}_f^T - \mathbf{A}_f^T) \underset{\text{LU}}{\backslash} \mathbf{A}_{tf}^T\mathbf{f}_t + \mathbf{A}_{fj}^T(\lambda\mathbf{B}_f^T - \mathbf{A}_f^T) \underset{\text{LU}}{\backslash} \mathbf{A}_{jf}^T\mathbf{f}_j + \mathbf{A}_{tj}^T\mathbf{f}_t] \\
&= (\lambda\mathbf{B}_j^T - \mathbf{A}_j^T) \underset{\text{GE}}{\backslash} [(\mathbf{A}_{fjtf} + \mathbf{A}_{tj}^T)\mathbf{f}_t + \mathbf{A}_{fjjf}\mathbf{f}_j] \qquad (4.42)
\end{aligned}$$

where

$$\begin{aligned}
\mathbf{A}_{fjtf} &= \mathbf{A}_{fj}^T(\lambda\mathbf{B}_f^T - \mathbf{A}_f^T) \underset{\text{LU}}{\backslash} \mathbf{A}_{tf}^T \\
\mathbf{A}_{fjjf} &= \mathbf{A}_{fj}^T(\lambda\mathbf{B}_f^T - \mathbf{A}_f^T) \underset{\text{LU}}{\backslash} \mathbf{A}_{jf}^T
\end{aligned}$$

Inserting 4.41 into 4.42 leads to

$$\begin{aligned}
\mathbf{f}_j &= (\lambda\mathbf{B}_j^T - \mathbf{A}_j^T - \mathbf{A}_{fjjf}) \underset{\text{GE}}{\backslash} [(\mathbf{A}_{fjtf} + \mathbf{A}_{tj}^T)(\mathbf{A}_{bj}\mathbf{f}_j + \mathbf{A}_{bNT}\mathbf{f}_n + \mathbf{A}_{bq}\mathbf{f}_q)] \\
&= (\lambda\mathbf{B}_j^T - \mathbf{A}_j^T - \mathbf{A}_{fjjf} - (\mathbf{A}_{fjtf} + \mathbf{A}_{tj}^T)\mathbf{A}_{bj}) \underset{\text{GE}}{\backslash} (\mathbf{A}_{fjtf} + \mathbf{A}_{tj}^T)(\mathbf{A}_{bNT}\mathbf{f}_n + \mathbf{A}_{bq}\mathbf{f}_q) \\
&= \mathbf{A}_{Jnq}(\mathbf{A}_{bNT}\mathbf{f}_n + \mathbf{A}_{bq}\mathbf{f}_q) \qquad (4.43)
\end{aligned}$$

where

$$\mathbf{A}_{Jnq} = (\lambda\mathbf{B}_j^T - \mathbf{A}_j^T - \mathbf{A}_{fjjf} - (\mathbf{A}_{fjtf} + \mathbf{A}_{tj}^T)\mathbf{A}_{bj}) \underset{\text{GE}}{\backslash} (\mathbf{A}_{fjtf} + \mathbf{A}_{tj}^T)$$

Equation 4.43 expresses $\mathbf{f}_j$ as a function of $\mathbf{f}_q$ and $\mathbf{f}_n$. These vectors depend also on $\mathbf{f}_f$,$\mathbf{f}_t$ and $\mathbf{f}_j$.

$$\begin{aligned}
\mathbf{f}_n &= -\mathbf{A}_N^T \underset{\text{PCG}}{\backslash} \mathbf{A}_{qn}^T\mathbf{f}_q \qquad (4.44) \\
\mathbf{f}_q &= \mathbf{A}_{tq}^T\mathbf{f}_t + \mathbf{A}_{fq}^T\mathbf{f}_f \qquad (4.45)
\end{aligned}$$

Inserting 4.38 into 4.45 leads to

$$\begin{aligned}
\mathbf{f}_q &= \mathbf{A}_{tq}^T\mathbf{f}_t + \mathbf{A}_{fq}^T(\lambda\mathbf{B}_f^T - \mathbf{A}_f^T) \underset{\text{LU}}{\backslash} (\mathbf{A}_{tf}^T\mathbf{f}_t + \mathbf{A}_{jf}^T\mathbf{f}_j) \\
&= \mathbf{A}_{tq}^T\mathbf{f}_t + \mathbf{A}_{fqtf}\mathbf{f}_t + \mathbf{A}_{fqjf}\mathbf{f}_j \\
&= (\mathbf{A}_{tq}^T + \mathbf{A}_{fqtf})\mathbf{f}_t + \mathbf{A}_{fqjf}\mathbf{f}_j \qquad (4.46)
\end{aligned}$$

where

$$\begin{aligned}
\mathbf{A}_{fqtf} &= \mathbf{A}_{fq}^{T}(\lambda\mathbf{B}_{f}^{T} - \mathbf{A}_{f}^{T}) \underset{\text{LU}}{\backslash} \mathbf{A}_{tf}^{T} \\
\mathbf{A}_{fqjf} &= \mathbf{A}_{fq}^{T}(\lambda\mathbf{B}_{f}^{T} - \mathbf{A}_{f}^{T}) \underset{\text{LU}}{\backslash} \mathbf{A}_{jf}^{T}
\end{aligned}$$

Inserting 4.41 into 4.46 leads to

$$\begin{aligned}
\mathbf{f}_q &= (\mathbf{A}_{tq}^{T} + \mathbf{A}_{fqtf})(\mathbf{A}_{bj}\mathbf{f}_j + \mathbf{A}_{bNT}\mathbf{f}_n + \mathbf{A}_{bq}\mathbf{f}_q) + \mathbf{A}_{fqjf}\mathbf{f}_j \\
&= [(\mathbf{A}_{tq}^{T} + \mathbf{A}_{fqtf})\mathbf{A}_{bj} + \mathbf{A}_{fqjf}]\mathbf{f}_j + (\mathbf{A}_{tq}^{T} + \mathbf{A}_{fqtf})(\mathbf{A}_{bNT}\mathbf{f}_n + \mathbf{A}_{bq}\mathbf{f}_q) \qquad (4.47)
\end{aligned}$$

Inserting 4.43 into 4.47 leads to

$$\begin{aligned}
\mathbf{f}_q &= ([(\mathbf{A}_{tq}^{T} + \mathbf{A}_{fqtf})\mathbf{A}_{bj} + \mathbf{A}_{fqjf}]\mathbf{A}_{Jnq} + (\mathbf{A}_{tq}^{T} + \mathbf{A}_{fqtf}))(\mathbf{A}_{bNT}\mathbf{f}_n + \mathbf{A}_{bq}\mathbf{f}_q) \\
&= \mathbf{A}_{QN}\mathbf{f}_n + \mathbf{A}_{QQ}\mathbf{f}_q \qquad (4.48)
\end{aligned}$$

where

$$\mathbf{A}_{QN} = ([(\mathbf{A}_{tq}^{T} + \mathbf{A}_{fqtf})\mathbf{A}_{bj} + \mathbf{A}_{fqjf}]\mathbf{A}_{Jnq} + (\mathbf{A}_{tq}^{T} + \mathbf{A}_{fqtf}))\mathbf{A}_{bNT} \qquad (4.49)$$

$$\mathbf{A}_{QQ} = ([(\mathbf{A}_{tq}^{T} + \mathbf{A}_{fqtf})\mathbf{A}_{bj} + \mathbf{A}_{fqjf}]\mathbf{A}_{Jnq} + (\mathbf{A}_{tq}^{T} + \mathbf{A}_{fqtf}))\mathbf{A}_{bq} \qquad (4.50)$$

Equation 4.48 is the base of the calculation of the left eigenvector, since the set of equations 4.48 and 4.44

$$\begin{aligned}
\mathbf{f}_q &= (\mathbf{I} - \mathbf{A}_{QQ})\backslash\mathbf{A}_{QN}\mathbf{f}_n \\
\mathbf{f}_n &= -\mathbf{A}_{N}^{T}\backslash\mathbf{A}_{qn}^{T}\mathbf{f}_q \qquad (4.51)
\end{aligned}$$

leads to

$$\mathbf{0} = [\mathbf{A}_{N}^{T} + \mathbf{A}_{qn}^{T}(\mathbf{I} - \mathbf{A}_{QQ})\backslash\mathbf{A}_{QN}]\mathbf{f}_n \qquad (4.52)$$

which is not yet tractable from a computational point of view. The easily drawn backslash is an unsurmountable obstacle at the time being. Knowing this, it's not at all obvious, that the matrices $\mathbf{A}_{QN}$ and $\mathbf{A}_{QQ}$ in 4.48 may be calculated in a efficient manner. On the contrary, a detailed investigation shows, that the two matrices cannot be calculated with reasonable effort. However, an iterative approach to equation 4.48 does not require the explicit values of $\mathbf{A}_{QN}$ and $\mathbf{A}_{QQ}$. It is much easier to calculate $\mathbf{A}_1\backslash(\mathbf{A}_2\mathbf{f})$ which can be reduced to $\mathbf{A}_1\backslash\tilde{\mathbf{f}}$ instead of $\mathbf{A}_1\backslash\mathbf{A}_2$. This implies, that some of the above mentioned matrices cannot be given explicitly, but the resulting vector of a matrix/vector multiplication may be calculated nevertheless.

The final algorithm looks as follows.

```
Algorithm 4.7: Left Eigenvector

1. $\mathbf{f}_{n,k} = \mathbf{e}_n$ or any starting guess
2. $\mathbf{f}_{q,k} = \mathbf{e}_q$ or any starting guess
3. $\mathbf{f}_{q,k+1} = \mathbf{A}_{QN}\mathbf{f}_{n,k} + \mathbf{A}_{QQ}\mathbf{f}_{q,k}$    (4.48)
4. $\mathbf{f}_{n,k+1} = -\mathbf{A}_N^T \underset{\text{PCG}}{\backslash} \mathbf{A}_{qn}^T\mathbf{f}_{q,k+1}$    (4.44)
5. go back to step 3 until convergence of $\mathbf{f}_q$ and $\mathbf{f}_n$ is reached.
6. $\mathbf{f}_{j,k+1} = \mathbf{A}_{Jnq}(\mathbf{A}_{bNT}\mathbf{f}_n + \mathbf{A}_{bq}\mathbf{f}_q)$    (4.43)
7. $\mathbf{f}_{t,k+1} = \mathbf{A}_{bj}\mathbf{f}_{j,k+1} + \mathbf{A}_{bNT}\mathbf{f}_{n,k+1} + \mathbf{A}_{bq}\mathbf{f}_{q,k+1}$    (4.41)
8. $\mathbf{f}_{f,k+1} = (\lambda\mathbf{B}_f^T - \mathbf{A}_f^T) \underset{\text{LU}}{\backslash} (\mathbf{A}_{tf}^T\mathbf{f}_{t,k+1} + \mathbf{A}_{jf}^T\mathbf{f}_{j,k+1})$    (4.38)
9. $\mathbf{f}_{q,k+2} = (\mathbf{A}_{tq}^T + \mathbf{A}_{fqtf})\mathbf{f}_{t,k+1} + \mathbf{A}_{fqjf}\mathbf{f}_{j,k+1}$    (4.46)
10. go back to step 4 until convergence of $\mathbf{f}_f$,$\mathbf{f}_t$ and $\mathbf{f}_j$ is reached.
```

The steps 9 and 10 are necessary because the coupling between the equations is not strong enough to give good results after the first iteration. The reason lies in the large variety of matrix sizes involved. $\mathbf{A}_j$ is smaller than 100x100 and $\mathbf{A}_f$ on the other hand is more like 50'000x50'000. These uneven sizes are mandatory to overcome the loss of sparsity during sub-calculations but lead to a weak coupling between some of the equations.

Naturally, the results for the left eigenvector will never be better, than the quality of the eigenvalue taken from the right eigenvector calculation.

4.7 Regional Modes

The algorithm 4.6 is not specific to the calculation of the global mode. The control over the mode calculated lies in the starting guess $[\lambda_0, \mathbf{e}_0]$. Therefore it is possible to find regional oscillations with a corresponding starting guess, i.e. the eigenvector contains the information if an oscillation is global or regional in its shape, the starting guess for regional oscillations must therefore also have the shape of a regional oscillation.

The complex structure of $\mathbf{A}$ prohibits the convergence of the algorithm if the starting guess is not within a relatively close vicinity of the solution. Therefore it is necessary to calculate a starting guess for the regional neutronic eigenvector $\tilde{\mathbf{e}}_{n,0}$ from the higher harmonics of the static nodal equation A.36

$$\bar{\varphi}_{1n} = \frac{\sum_{m=1}^{6}\left(\mathbf{Y}_{1,nm} + \tilde{\beta}\frac{\nu_2\Sigma_{f2}}{\Sigma_{a2}}\mathbf{Y}_{2,nm}\right)\bar{\varphi}_{1m}}{\sum_{m=1}^{6}(\mathbf{X}_{1,nm} + \tilde{\beta}\frac{\nu_2\Sigma_{f2}}{\Sigma_{a2}}\mathbf{X}_{2,nm}) - \Sigma_{a1}(k_\infty - 1)} \equiv \sum_{m=1}^{6}\mathbf{A}_{nm}\bar{\varphi}_{1m} \qquad \text{(A.36)}$$

where n denotes the number of the node and m stands for the six spatial neighbors. The value k in the matrix $\mathbf{A}_{nm}$ is taken from the steady state core simulator of the NPP.

The starting guesses for the regional calculation are the dominating eigenvectors of $\mathbf{A}_{nm}$. These vectors represent the neutronic modes of the reactor. The eigenvectors of the relatively small matrix $\mathbf{A}_{nm}$ are calculated by the built in functions of MATLAB.

Applying these good starting guesses, it is possible to use algorithm 4.6 as in the global case. To make absolutely sure, that the algorithm converges to a regional solution, the matrices $\mathbf{A}_{jt}, \mathbf{A}_{j}, \mathbf{A}_{jf}, \mathbf{A}_{fj}, \mathbf{A}_{tj}$ and the vector $\mathbf{e}_j$ are assumed to be zero. Physically this means, that the regional oscillation may not be seen from the outside of the reactor, since the core flow is not oscillating.

The left eigenvector of the regional oscillation is calculated in the same manner as for the global mode. Actually, the calculation comes much cheaper in computer time, since, as stated before, a significant number of matrices may be assumed to be zero.