PRECISE User's Guide
Version 1.0

—

CERFACS Technical Report
TR/PA/97/38

R. A. McCoy and V. Toumazou

Contents

Chapter 1

Introduction

PRECISE is a set of tools to perform numerical experiments exploring the robustness of computational schemes. Designed for developers and users of numerical software, it provides a means for studying the reliability and stability properties of both numerical algorithms and mathematical problems. Using PRECISE, one can determine the quality of computed solutions with respect to a choice of algorithm, finite precision arithmetic, data imprecision, or any other uncertainties. When used with a backward stable algorithm, one may investigate instabilities in the physical or mathematical problem itself, particularly near singularities. Conversely, when used with well-understood mathematical problems, one can study the reliability of a numerical algorithm and its implementation.

An important aspect of PRECISE is that it is based on analysis of *computed* results, those which are actually obtained by a computer implementation of a numerical algorithm applied to a specific problem. As such, a PRECISE analysis is complementary to the standard a priori error bounds usually obtained for numerical algorithms. Moreover, since a priori error bounds are generally over-estimates of worse-case error and often assume error-free implementations, the results obtained by PRECISE may be viewed as more pertinent to the problem at hand. For those who have ever obtained a numerical result and then wondered "is it correct?", PRECISE is designed to address the inherent subtleties in this question.

PRECISE, like any tool, requires both knowledge and experience for its effective use. For this purpose, Chapter 2 is included to provide background material on the ideas behind PRECISE. For those new to this type of analysis, this chapter provides an elementary introduction to the concepts necessary for experiments in numerical stability and reliability; references to the ample literature in this field are provided to allow further investigations into specific areas. For researchers already familiar to this type of numerical analysis, this chapter should serve as a reminder and reference during use of the PRECISE modules.

The two primary components of PRECISE, the module for backward error analysis and the module for spectral sensitivity analysis, are described in Chapters 3 and 4. The descriptions include further background material which is specific to each module and discussions of the specific software components. In addition, these chapters contained detailed examples which demonstrate how PRECISE can be used to study properties of numerical algorithms and mathematical problems.

In an effort to make these tools easy to use, the theoretical aspects behind PRECISE are only described here in an elementary fashion. This document concentrates on the *methodology* of performing a PRECISE analysis, along with interpreting the results. As such, much of the formal framework and technical machinery which are usually found in the literature on error analysis are not found here in this guide. This is not to imply that the theoretical discussions are not important here; on the contrary, it is intended that results obtained with PRECISE will provide further insight into numerical behaviour and both complement and enrich ideas presented in more formal settings. A good starting point for in depth discussions of the ideas behind PRECISE are provided in the recent book by Chaitin-Chatelin and Frayssé [5].

Chapter 2

PRECISE Concepts

PRECISE is composed of two distinct modules:

> Module 1 is a set of tools to perform a statistical backward error analysis. This module can be applied to any numerical problem which is amenable to a backward error analysis. It provides estimates for:
>
> - backward errors,
> - reliability and quality indices,
> - condition numbers at regular and singular points for both the algorithm/method and the mathematical problem,
> - distances to singularities,
> - order of Hölder singularities.
>
> Module 2 is a set of executable programs to perform a spectral sensitivity analysis. The programs can be used to produce visual representations of:
>
> - perturbed spectra,
> - spectral portraits and pseudospectra for matrices,
> - sensitivity portraits.

These modules are presented in Chapters 3 and 4. However, background concepts which are common to both modules are collected here in this chapter to provide a reference point for the use of the PRECISE tools. Therefore, this chapter should be read at least once by all users to gain an understanding of the ideas used in the descriptions of Modules 1 and 2 in subsequent chapters.

The first concept which is essential to the interpretation of results for both modules is the principal of backward error [20].

2.1 Backward Error

We assume a mathematical problem (P) of the form

$$F(x) = y \tag{2.1}$$

is to be solved for x. No other assumptions are made on F or y, F may be linear or non-linear, and y may be any dimension. An example problem might be a linear system to be solved such as $Ax = b$. Once a solution $\tilde{x}$ is obtained, naturally an important question is how well $\tilde{x}$ approximates the solution x; this is known as a forward error analysis.

The important feature of (P) which allows a backward error analysis is that there is a mapping back to the original data for a comparison of $F(\tilde{x})$ and y. The difference between $F(\tilde{x})$ and y is usually called the residual. The backward error involves a normalisation to take into account the weighting of both the original data and and the obtained solution. The normalisation is specific to the type of mathematical problem. In practice, since all quantities are actually approximated on a computer and F is (usually) evaluated by a numerical algorithm, the backward error too can only be approximated, although this distinction is usually ignored.

The importance of backward error estimates is that it makes it possible to know if the approximate solution $\tilde{x}$ solves the original problem (P). The next logical step, which is the principal of backward error analysis, is to consider the computed solution $\tilde{x}$ as the exact solution to a nearby problem $(\tilde{P})$. This simple idea has powerful implications. For one, it enables the use of analytical tools such as differentiability and perturbation theory in the analysis of numerical algorithms. Also, the details of the computer arithmetic become secondary; the errors in the computation are interpreted in terms of equivalent perturbations on the given problem, while the computed quantities are *exact* for the perturbed problem $(\tilde{P})$. Rounding errors are put on the same footing as errors in the original data. And when establishing error bounds, one can distinguish between contributions from the algorithm and contributions from the problem. Finally, a wide variety of *user-defined* perturbations may be used for a sensitivity analysis. As we shall see, the last item can be put to particularly productive purposes. In Section 2.2, we discuss perturbations in more detail.

The value of the backward error can be viewed as the minimal distance of (P) to the set of perturbed problems that are solved exactly by $\tilde{x}$. For problems which do not involve singularities, the relation between the forward error and the backward error is given by the inequality

$$\text{forward error} \leq K \times \text{backward error},$$

where the value K represents the conditioning of the problem. If the problem is well conditioned, that is, the value of K is not large, then the magnitude of the forward error can be reasonably well determined by the value of the backward error. In practice, for problems in which K is of magnitude 10^q and the magnitude of the backward error is 10^{-k}, then one expects the components of $\tilde{x}$ to agree with the solution x to roughly $k - q$ decimal digits. Backward error estimates, when available, are also highly recommended as stopping criteria for iterative algorithms [1, 2].

A backward error analysis is usually performed to understand the behaviour of a specific algorithm applied to a particular type of problem of the form (P). Due to finite precision arithmetic, a numerical algorithm is always assumed to introduce perturbations in the course of obtaining the approximate solution $\tilde{x}$. The ideal situation is to have the backward error the same magnitude as the machine precision $\mathbf{u}$; in this case, solutions to (P) and $(\tilde{P})$ are numerically indistinguishable. An algorithm which produces a backward error which corresponds to the magnitude of perturbations on (P) is known as *backward stable*. The backward stability of an algorithm is a qualitative property of the algorithm's sensitivity to perturbations in the data.

The backward error, therefore, is closely linked to the type of perturbations on the problem (P). Most discussions of backward error are based in terms of perturbations of the size of the machine precision $\mathbf{u}$, since these must always be considered with any numerical algorithm. Similarly, the backward stability of an algorithm is often expressed strictly in terms of machine precision. However, an important idea to keep in mind for the use of PRECISE is that, if an algorithm is indeed backward stable, it will be stable for a range of perturbations sizes and not only for those close to the machine precision. Thus, one may examine numerical stability by intentionally introducing, in a controlled way, perturbations on a problem and observing the resulting outcomes. One may observe how different types of perturbations affect the results, and can lead to the discovery of sensitive or unreliable features in the solution. In cases where the perturbations reveal no instabilities, then the analysis provides the user with further evidence to the fact that the computed solution can be considered accurate, stable, and reliable, i.e. a "good" solution.

Interestingly, different types of perturbations can produce markedly different behaviour for the same algorithm applied to the same problem. It is not only a question of the size of the perturbations; the manner in which they are applied to the problem under study can affect the outcomes. The term *perturbation model* is used to describe the set of perturbations which are chosen to produce the computed results. It is important to keep in mind that the quantitative measures of reliability and sensitivity are only defined with respect to a specific perturbation model. In other words, the values for backward error and condition numbers differ according to each perturbation model. As explained in Section 2.2, there are actually an infinite variety of perturbation models which can be applied to any problem.

2.2 Perturbation Models

Behind the use of all the PRECISE software is the use of structured perturbations on the numerical data. The notion behind a perturbation model is to mimic the effects of finite-precision arithmetic or data uncertainty to the point of producing observable, quantifiable behaviour. It is most interesting to observe the behaviour over a range of meaningful perturbation sizes and observe if the behaviour remains consistent over the entire range. If instabilities are found, then this reveals much about the numerical problem; if not, this provides strong evidence of numerical reliability.

The basic items which compose a perturbation model are the range of perturbation sizes, the type of metric for the data, the choice of data items to be perturbed, and the type of perturbations to be performed. Each of these items is explained below.

2.2.1 Perturbation Values

In general, a wide range of perturbation values should be used, spanning at least a few orders of magnitude. For data obtained from experimental measurements or truncated approximations, the perturbation range should include the order of the data uncertainty; one is looking to see if there is a qualitative change in the results for perturbations at this size. For the same reason, perturbations of the same order of magnitude as the machine precision $\mathbf{u}$ are also frequently used. For computers implementing IEEE floating-point arithmetic, the values for the machine precision are shown in Table 2.1. An example range of perturbation values, therefore, might be ten values taken uniformly from the range $[2^{-54}, 2^{-10}]$. The magnitude of the perturbation values is closely related to the type of perturbations applied to the data, which are discussed below.

	Machine Precision
Single Precision	$2^{-24} \approx 5.96 \cdot 10^{-8}$
Double Precision	$2^{-53} \approx 1.11 \cdot 10^{-16}$

Table 2.1: Machine precision values for IEEE standard floating-point arithmetic.

2.2.2 Perturbation Types

For numerical linear algebra algorithms, the two most common types of perturbation models involve either normwise or componentwise perturbations. As the name implies, in a normwise model the data components of (P) are perturbed by values $||F||$ and $||y||$, which correspond to the magnitude of the data with respect to the chosen metric. In a componentwise model the data components are perturbed according to each components individual values, $|F|$ and $|y|$, which correspond to the modulus values of the data. As an example, consider the case of studying a linear system $Ax = b$ with perturbations of size t. A normwise perturbation model would produce a set of perturbed systems of the form $(A + \Delta A)x = (b + \Delta b)$, where the perturbation matrices satisfy

$$\begin{aligned} ||\Delta A|| &\leq t||A||, \\ ||\Delta b|| &\leq t||b||. \end{aligned}$$

While in a componentwise perturbation model, the inequalities defining the perturbations are given by

$$\begin{aligned} |\Delta A| &\leq t|A|, \\ |\Delta b| &\leq t|b|, \end{aligned}$$

where the modulus inequalities are understood to be satisfied for each component. The componentwise model is seen to preserve the same structure of the data; in particular, for sparse matrices the same sparsity pattern is maintained for compenentwise perturbations, but not for normwise perturbations.

These distinctions have originally developed because of the type of theoretical analysis performed for certain algorithms. Inspection of the above inequalities shows that a normwise analysis is based on a more global treatment of the problem than a componentwise analysis. As such, analytical error bounds obtained through a normwise analysis may be more pessimistic in some cases than those obtained from a componentwise analysis; this is the case, for example, in the solution of linear systems by Gaussian elimination.

For the purposes of PRECISE, however, these distinctions are not so critical. One powerful aspect of PRECISE is that any type of perturbation model can be used to search for meaningful results; there is no restriction in this regard. It is useful, therefore, to generalise the inequalities defining the perturbation types. The normwise perturbations must satisfy

$$\begin{aligned} ||\Delta A|| &\leq t\alpha, \\ ||\Delta b|| &\leq t\beta \end{aligned}$$

for some quantities $\alpha \geq 0$ and $\beta \geq 0$ which can be adjusted according to the perturbation model. For instance, values of $\alpha \approx ||A||$ and $\beta = 0$ indicate the same normwise perturbation on the matrix A as before, but no perturbations on the right-hand-side b. Another example might be $\alpha = 1$ and $\beta = 1$, specifying perturbations which are fixed in size for both A and b. Any combination of values for α and β gives a valid perturbation model for a study by PRECISE, except of course the combination $\alpha = 0$ and $\beta = 0$.

Similarly, the componentwise perturbations are generalised by the inequalities

$$\begin{aligned} |\Delta A| &\leq tE, \\ |\Delta b| &\leq tf, \end{aligned}$$

where E is any matrix of the same order as A and f is a vector of the same size as b, with both E and f having non-negative components. For problems involving sparse matrices, both E and f are usually specified to have the same structural properties as A and b, respectively.

2.2.3 Data Metrics

This choice pertains to the choice of norm used to measure distances in the data space. For instance, in linear algebra contexts commonly used norms are the 2-norm, the 1-norm, the ∞-norm, or in some cases, the Frobenius norm. In Module 1, a metric is required to determine the estimators for the backward error. This choice is problem dependent, and depends also on the goals of the user. For normwise perturbation models, this choice obviously affects the actual magnitude of the perturbations on the components. To perform an error analysis, a suitable choice is often the ∞-norm, which gives a good upper bound on error quantities and in addition is inexpensive to compute, particularly when the data is distributed in a multi-processor environment.

2.2.4 Data to be Perturbed

One has the flexibility to perturb all or just a subset of the data components in the problem. For example, when studying a linear system $Ax = b$, one can perturb only the matrix A, only the right-hand-side b, or both A and b. One might also choose to perturb only a few elements of A, for instance, those which correspond to certain critical unknown variables. One usually starts with a simple approach, such as perturbing all data elements, and then trying other combinations for comparison.

2.2.5 Choosing a Perturbation Model

While there are an infinite number of possibilities for choosing a perturbation model, there are some clear guidelines to follow to make the choices quite straight-forward. Most often, when performing a stability analysis, a series of perturbation models will be used to gain further and further insight into the numerical behaviour of the problem and its solution. Therefore, one may start with almost with any model, and then refine the choices according to the phenomena which are observed. The choices are always problem dependent, but the following general principles can be used to guide the analysis.

Initially, one usually starts with a wide range of perturbation sizes which includes the machine precision or any level of data uncertainty. For example, if the numerical procedure calls for a parameter to define the error tolerance of the approximated solution, then the perturbation range should include this value. A special case of this situation is the specification of the stopping criteria for iterative solvers; for example, if the iterations are stopped using a criteria of 10^{-k}, then the perturbation interval should bracket this value. While the range of perturbation values can initially be quite large, the actual number of perturbation sizes taken from the range does not have to be large also. Five to ten perturbation sizes taken from the range $[\xi^{-k_{\min}}, \xi^{-k_{\max}}]$ should suffice for initial tests. As values for the stability indicators are observed, one may increase the number of perturbations taken from the range, and also adjust the values of $k_{\min}$ and $k_{\max}$ accordingly. Note that the user is free to specify any value for ξ, but a value of $\xi = 2$ allows a fine sampling of interval range.

The choices of perturbation types, data metrics, and the data to be perturbed are all very problem dependent. If the numerical algorithm under study is known to exhibit certain behaviour for a particular type of perturbation, then this most certainly should be used in a stability analysis. On the other hand, if no theoretical behaviour is predicted, then the user is free to make these choices based on other criteria. Notably, the manner in which the numerical procedure is implemented can be used to guide the choice for an initial perturbation model. In other words, the model which is most conveniently implemented may be used first. See also Section 3.2.1 for a discussion on implementation issues concerning perturbation models.

Finally, after an initial perturbation model has been selected and the results interpreted, the choice of subsequent perturbation models will depend on the observed results. There are in general three categories of behaviour which might be observed. If the results show no evidence of numerical instability, then this provides corroborative evidence of the reliability of the numerical procedure. In this case, one would like to apply more rigorous tests to the problem in the hope that the same positive results can be obtained. The more tests that are applied with positive outcomes, the stronger the validation of the numerical procedure. For this situation, the user is free to change the perturbation model in any number of ways, for instance by changing the perturbation sizes, the perturbation type, or the data metric. The process can continue in this way until the user is confident in the results.

The second category of results involves the situation when instabilities are observed. Thus, the subsequent perturbation models should be designed to investigate this condition. If instabilities are observed, further insight can be gained by using the same perturbation sizes and changing the type of perturbation or by limiting the data which is being perturbed. In such cases, it is not uncommon for different perturbation models to produce remarkably different behaviour in the numerical problem; the results are not contradictory, but actually provide a more complete characterisation of the numerical properties of the problem under study.

As explained in Section 3.1, the indicators may be inconclusive for some perturbation values. If the indicators show inconclusive results, this can either be the result of an improper perturbation model or an actual problem in the numerical procedure. In this case, the range of perturbation values should be changed to determine if the indicators reveal different results. Different perturbation types or data metrics may also be used. If a number of perturbation models are attempted without conclusive results, then most likely some anomaly in the numerical procedure is present. For example, a numerical procedure which returns a constant error value for a certain range of perturbation values would exhibit this behaviour. Thus, in such cases, the numerical procedure requires a more thorough inspection to determine the cause. See also Section 3.1 for a discussion on interpreting the stability indicators.

Chapter 3

Module 1 - Stability Analysis

The framework for analysing the stability properties of numerical problems and algorithms is presented in this chapter. This module is suitable for numerical problems which are amenable to a backward error analysis; that is, problems which can be expressed in the form

$$F(x) = y, \tag{3.1}$$

where x is to be solved from the known values F and y. Problems of this type include the determination of roots of polynomials, the solution of linear systems $Ax = b$, eigenproblems $Ax = \lambda x$, and generalised eigenproblems $Ax = \lambda Bx$.

The analysis procedure involves the generation of data for the calculation of statistical stability, reliability, and conditioning indicators. The problem (3.1) will be solved numerous times but each time with the data perturbed in a controlled manner. Because the perturbations are defined by the user, statistical indicators can be generated which reflect the stability properties of the numerical problem and the solution method. The general procedure to perform a backward error analysis requires the following steps:

1 Select and Implement a Perturbation Model

2 Generate Sample Data

3 Calculate Statistical Indicators

4 Interpret Indicators

As shall be shown later, steps (2) and (3) can be done separately in two steps or together in one step. The general procedure is often iterative; the indicators will be produced repeatedly for a number of different perturbation models, with each set of results providing a deeper characterisation of the numerical properties of the problem.

Assuming that one already has a working numerical routine for a particular problem, a backward error analysis requires a small amount of software development effort. In this respect there are two main tasks to perform. The first corresponds to Step 1 above, which is to implement the perturbation model, including an iteration to produce samples of perturbed solutions for each perturbation size. The second task, which corresponds to Step 3, is to process the samples and calculate values for the stability indicators. Neither task is difficult, and PRECISE subroutines are provided to aid the user in this endeavor.

In the next section, indicators which display the stability and reliability properties of the numerical problem are described. A discussion of how to interpret these indicators is also presented for linear systems and standard eigenvalue problems. Subroutines provided for a backward error analysis are described in Section 3.3; these routines are generic enough to be applicable to many problems, and they may also be used as a starting point and customised for particular cases. In addition, completed programs to perform a backward error analysis utilising certain popular numerical software packages such as the NAG Parallel Library, LAPACK, and SCALAPACK are also provided; these programs can also serve as examples for the use of PRECISE with other numerical software.

3.1 Reliability and Stability Indicators

The primary design of the backward error analysis is to produce meaningful indicators of reliability and stability. Since the indicators are produced by an *a posteriori* process, they provide pertinent information about the numerical problem and the suitability of the numerical algorithm used to solve it. The indicators provide estimates of backward error and conditioning which are clearly distinguishable with respect to the corresponding perturbation models. In addition, if the problem under study is influenced by singularities, improved sensitivity measures can be given in the form of Hölderian condition numbers. In this section, the meanings of the indicators are explained in a general formulation, and then more specifically for the cases of linear systems and eigenvalue problems. A discussion of how to interpret the indicators is given in Section 3.1.4.

3.1.1 General Formulation

The indicators are presented here in a general formulation to provide a generic structure which is applicable to any type of numerical problem which can be analysed via the backward error. The stability and reliability indicators are more precisely described for the cases of linear systems and standard eigenproblems in Sections 3.1.2 and 3.1.3, respectively. Methods for actually calculating the indicators are presented in Section 3.2.2.

The following notation is required to describe the indicators. Assume as before that the problem $F(x) = y$ is to be solved for x. Let x_θ be the solution computed by the numerical method G_θ; that is, $x_\theta = G_\theta(y)$. The subscript θ is used to denote the fact that, for the same type of problem, one may use different numerical schemes, algorithms, and implementations. For consistency, let F_θ represent the numerical representation of F. The numerical scheme that is under study can then be denoted by

$$y \overset{G_\theta}{\longmapsto} x_\theta \overset{F_\theta}{\longmapsto} \zeta_\theta = F_\theta(x_\theta).$$

Let z represent the data chosen to be perturbed from the original problem $F(x) = y$, let $\|\cdot\|$ be the data metric for the perturbation model, and let δ be a positive parameter which measures the size of the data perturbations: $\delta = \|\Delta z\|$. For a specific perturbation Δz, we have

$$z + \Delta z \overset{G_\theta}{\longmapsto} x_\theta + \Delta x_\theta \overset{F_\theta}{\longmapsto} \zeta_\theta + \Delta\zeta_\theta = y + \Delta w_\theta. \tag{3.2}$$

The value $\Delta\zeta_\theta$ measures the variation with respect to ζ_θ, and Δw_θ measures the variation with respect to y. This notation allows the following definitions for indicators of reliability and sensitivity. One should keep in mind that the indicators are always defined with respect to the given perturbation model.

Indicator $\mathcal{I}$ - Reliability and Numerical Stability

In the notation defined above, the reliability of considering G_θ as an approximation to F^{-1} is measured by

$$\mathcal{I}_{\theta\delta} = \sup_{||\Delta z||=\delta} \frac{1}{\delta} ||\Delta w_\theta||. \tag{3.3}$$

This indicator is of primary importance in that the validity of the other indicators is revealed by the qualitative behaviour of $\mathcal{I}$. The point addressed by this indicator pertains to the reliability of considering the numerical procedure G_θ as an inverse to the mapping F at x_θ. The idea is to observe if perturbations of size δ on the input data are consistently exhibited in the solutions obtained from the numerical procedure. Ideally, a numerical procedure that is backward stable will produce perturbations in the solution which correspond in size to perturbations in the input data.

Observe that the quantity Δw_θ is needed to determine this indicator. Since this value varies with respect to y, the weighting of the mapping $F(x_\theta + \Delta x_\theta)$ should be accounted for when producing values for $\mathcal{I}$. For this reason, if a known backward error formula is available for the type of problem being solved, this should be used to derive values for this indicator. An example of how this is done is shown in Section 3.1.2 for linear systems. In cases where no backward error formula exists, this value can be approximated by normalising with the value of y, but the results should be more carefully interpreted.

From this indicator, quantitative estimates of backward error values can be obtained, see Section 3.2.2. The remaining indicators are valid for perturbation values in which the numerical algorithm exhibits backward stability.

Indicator $\mathcal{K}$ - Stability and Conditioning

The sensitivity of F^{-1} at ζ_θ, with $x_\theta = F^{-1}(\zeta_\theta)$, is measured by

$$\mathcal{K}_\delta = \sup_{||\Delta z||=\delta} \frac{||\Delta x_\theta||}{||\Delta \zeta_\theta||}. \tag{3.4}$$

This indicator corresponds to conditioning values for the numerical problem. Technically, this sensitivity is only measured with respect to the specific perturbation model. The sensitivity of F^{-1} is estimated at ζ_θ and not at y because $x = F^{-1}(y)$ is unknown; this is meaningful only if F has similar behaviour around both x_θ and x, which pertains to the possibility of considering G_θ as an approximate inverse for F. This indicator gives a quantitative estimate of the condition number for the numerical problem.

In situations where the solution is not smoothly dependent on the input data, the linear measure of sensitivity (3.4) does not properly characterise the numerical problem. Perturbations from finite-precision arithmetic or otherwise have a non-linear influence on the solution; they transform singular problems into a set of ill-conditioned regular problems [5]. Such cases include the computation of multiple eigenvalues or multiple roots of polynomials. For these cases, the linear sensitivity measure $\mathcal{K}$ will always indicate high conditioning values.

A more appropriate measure is given by Hölder sensitivities. For multiplicities of order m, the Hölder sensitivity of order $h = 1/m1$ is given by

$$\mathcal{K}_\delta^h = \sup_{\|\Delta z\|=\delta} \frac{\|\Delta x_\theta\|}{\|\Delta \zeta_\theta\|^h}. \tag{3.5}$$

This measure provides a characterisation of the singular problem as a whole, and not as a set of separate, ill-conditioned regular problems. Furthermore, since frequently the order of the multiplicity may be unknown in advance, or may be variable with respect to different perturbation models, this measure offers significant insight into the non-linearity of the problem.

Indicator $\mathcal{L}$ - Algorithm Sensitivity

The sensitivity of the algorithm G_θ with respect to the input data is given by $\mathcal{L}$. This indicator can be used as a measure of the suitability of the numerical method for the problem at hand. The sensitivity of G_θ at z is measured by

$$\mathcal{L}_{\theta\delta} = \sup_{\|\Delta z\|=\delta} \frac{1}{\delta}\|\Delta x_\theta\|. \tag{3.6}$$

This indicator, being a measure of sensitivity, is closely linked to the indicator $\mathcal{K}$. When the indicator $\mathcal{K}$ shows the numerical problem to be highly sensitive, then this indicator will also show high sensitivity from the variability of the solutions x_θ.

Indicator $\mathcal{E}$ - Forward Error

Estimates of forward error in the approximation x_θ can be made based on the unperturbed solution and the sensitivity indicators. The estimates are derived from the fact that the forward error is bounded by the product of the condition number and the backward error. For regular problems, the forward error is estimated by

$$\mathcal{E}_\delta = \mathcal{K}_\delta \times (\text{backward error}), \tag{3.7}$$

where the determination of the backward error is problem dependent. For singular problems, the forward error of a solution x_θ of multiplicity $m > 1$ is estimated using the Hölder sensitivity of order $h = 1/m$:

$$\mathcal{E}_\delta = \mathcal{K}_\delta^h \times (\text{backward error})^h, \tag{3.8}$$

For reference purposes, the indicators are summarised in Table 3.1. In the following sections, explicit formulae for these indicators are given for linear systems and standard eigenproblems. These two problem types are of particular interest because explicit formulations of the backward error have been developed for certain perturbation models; namely, those involving normwise or componentwise perturbations. The indicators above may be developed for other types of numerical problems as well. For example, backward error analysis has been studied for problem types such as linear least squares [13], generalised eigenproblems [12], and certain integration operators [14]. However, for these problems the theory is not as well-developed. In such cases, one may still apply the methods described here, although the biasing of the data should be taken into account when interpreting results.

Backward stability or reliability	$\mathcal{I}_{\theta\delta} = \max\limits_{\|\|\Delta z\|\|=\delta} \frac{\|\|\Delta w_\theta\|\|}{\delta}$	$\mathcal{I}_\theta = \lim\limits_{\delta\to 0} \mathcal{I}_{\theta\delta}$
Sensitivity of the numerical method	$\mathcal{L}_{\theta\delta} = \max\limits_{\|\|\Delta y\|\|=\delta} \frac{\|\|\Delta x_\theta\|\|}{\delta}$	$\mathcal{L}_\theta = \lim\limits_{\delta\to 0} \mathcal{L}_{\theta\delta}$
Conditioning of the problem	$\mathcal{K}_\delta = \max\limits_{\|\|\Delta z\|\|=\delta} \frac{\|\|\Delta x_\theta\|\|}{\|\|\Delta\zeta_\theta\|\|}$	$\mathcal{K} = \lim\limits_{\delta\to 0} \mathcal{K}_\delta$
Hölder Conditioning of the problem ($h = 1/m$)	$\mathcal{K}^h_\delta = \max\limits_{\|\|\Delta z\|\|=\delta} \frac{\|\|\Delta x_\theta\|\|}{\|\|\Delta\zeta_\theta\|\|^h}$	$\mathcal{K}^h = \lim\limits_{\delta\to 0} \mathcal{K}^h_\delta$
Forward Error Bound	$\mathcal{E}_\delta = \mathcal{K}_\delta \times$ (b. e.)	$\mathcal{E} = \lim\limits_{\delta\to 0} \mathcal{E}_\delta$
Forward Error Bound of multiplicity $m > 1$	$\mathcal{E}^h_\delta = \mathcal{K}^h_\delta \times (\text{b. e.})^h$	$\mathcal{E}^h = \lim\limits_{\delta\to 0} \mathcal{E}^h_\delta$

Table 3.1: General indicators of sensitivity and reliability for perturbation sizes δ. (b. e. denotes backward error).

3.1.2 Linear Systems

The indicators defined in Section 3.1.1 are defined here explicitly for the problem of linear systems $Ax = b$, where A is a square matrix, b is a vector, and the solution vector x is to be found by some numerical procedure G_θ. Recall that all the indicators are defined with respect to a given perturbation model, and so too is the backward error. Given in this section are indicator values for perturbation models using normwise and componentwise perturbation types (see Section 2.2).

We first recall equations for the backward error of linear systems. Let $\tilde{x}$ be a computed solution and $\tilde{r} = A\tilde{x} - b$ the associated residual vector. The *normwise* backward error associated with the approximate solution $\tilde{x}$ of the linear system $Ax = b$ is defined by

$$\begin{aligned} \eta^{\mathcal{N}} &= \inf\{\omega;\ \|\Delta A\| \le \omega\alpha, \|\Delta b\| \le \omega\beta \text{ and } (A+\Delta A)\tilde{x} = b + \Delta b\}, \\ &= \frac{\|\tilde{r}\|}{\alpha\|\tilde{x}\| + \beta} \qquad \text{(Rigal and Gaches (1967)).} \end{aligned}$$

The *componentwise* backward error associated with the approximate solution $\tilde{x}$ of the system $Ax = b$ is defined by:

$$\begin{aligned} \eta^{\mathcal{C}} &= \inf\left\{\omega;\ |\Delta A| \le \omega E;\ |\Delta b| \le \omega f \text{ and } (A+\Delta A)\tilde{x} = b + \Delta b\right\}, \\ &= \max_{1\le i\le n} \frac{|\tilde{r}_i|}{(E|\tilde{x}| + f)_i}, \qquad \text{(Oettli and Prager (1964)),} \end{aligned}$$

where $0/0$ is interpreted as 0 and $\xi/0$ is interpreted as ∞, in which case the backward error is not defined. Common parameter choices for the normwise model are $\alpha = ||A||$ and $\beta = ||b||$; while $E = |A|$ and $f = |b|$ are commonly used for the componentwise model.

To make use of the notation defined in Section 3.1.1, let x_θ represent the unperturbed solution of $Ax = b$ obtained by the method G_θ, and let ζ_θ be its image from the unperturbed map $\zeta_\theta = Ax_\theta$. Let z_t represent the perturbed input data for the problem produced by using perturbations of size t on the matrix A to obtain $A+\Delta A$ and/or perturbing the right-hand-side to obtain $b+\Delta b$. Let X_t represent the sample set obtained from solving $x_t = G_\theta(z_t)$ a number of times for the same perturbation size t; also, let Y_t represent the image of these samples, $Y_t = AX_t$. Let X_t have mean m_t and standard deviation σ_t; and let Y_t have mean ρ_t and standard deviation v_t. The analysis procedure then takes the form

$$\begin{array}{ccccc} & G_\theta & & A & \\ \left.\begin{array}{c} A+\Delta A \\ b+\Delta b \end{array}\right\} & \longmapsto & x_\theta + \Delta x_\theta & \longmapsto & \zeta_\theta + \Delta\zeta_\theta = b + \Delta w_\theta. \\ \downarrow & & \downarrow & & \downarrow \\ Z_t & & X_t & & Y_t \end{array} \tag{3.9}$$

For linear systems, the image sample Y_t needs to be normalised appropriately with respect to the right-hand-side b. That is, since the perturbations on the input data are to be compared in size to perturbations in the solutions, the data biasing $\rho_t - b$ must be accounted for when using the standard deviation for the reliability indicator $\mathtt{I}_{\theta\mathtt{t}}$. This is done by defining $(\hat{v}_t)_i = \left(v_{ti}^2 + |\rho_t - b|_i^2\right)^{1/2}$, for $i = 1, \ldots, n$, where n is the dimension of b. In this way, the value Δw_θ is estimated by $(\hat{v}_t)$, while the value $\Delta\zeta_\theta$ is estimated by v_t.

Using this procedure, then for each perturbation value t one can obtain the indicator values shown in Table 3.2.

	Normwise	Componentwise
$\mathtt{I}_{\theta t}$	$\dfrac{\lVert\hat{v}\rVert}{t(\alpha\lVert x_\theta\rVert + \beta)}$	$\dfrac{1}{t}\max\limits_{1\le i\le n}\dfrac{(\hat{v}_t)_i}{(E\lvert x_\theta\rvert + f)_i}$
$\mathtt{K}_t$	$\dfrac{\lVert\sigma_t\rVert(\alpha\lVert x_\theta\rVert + \beta)}{\lVert x_\theta\rVert\lVert v_t\rVert}$	$\dfrac{\lVert\sigma_t\rVert_\infty}{\lVert x_\theta\rVert_\infty}\dfrac{1}{\max\limits_{1\le i\le n}\dfrac{v_{ti}}{(E\lvert x_\theta\rvert + f)_i}}$
$\mathtt{L}_{\theta t}$	$\dfrac{\lVert\sigma_t\rVert}{t\lVert x_\theta\rVert}$	$\dfrac{\lVert\sigma_t\rVert_\infty}{t\lVert x_\theta\rVert_\infty}$
$\mathtt{E}_{\theta t}$	$\dfrac{\lVert\sigma_t\rVert}{\lVert x_\theta\rVert}\dfrac{\lVert Ax_\theta - b\rVert}{\lVert v_t\rVert}$	$\dfrac{\lVert\sigma_t\rVert_\infty}{\lVert x_\theta\rVert_\infty}\dfrac{\max\limits_{1\le i\le n}\dfrac{\lvert Ax_\theta - b\rvert_i}{(E\lvert x_\theta\rvert + f)_i}}{\max\limits_{1\le i\le n}\dfrac{v_{ti}}{(E\lvert x_\theta\rvert + f)_i}}$

Table 3.2: Statistical indicators for the analysis of linear systems $Ax = b$ using normwise or componentwise perturbation types. Common scalings for normwise perturbations are $\alpha = \lVert A\rVert$ and $\beta = \lVert b\rVert$, while for componentwise perturbations, $E = \lvert A\rvert$ and $f = \lvert b\rvert$.

3.1.3 Standard Eigenvalue Problems

In this section, explicit formulae are given for the indicators for standard eigenvalue problems $Ax = \lambda x$. The analysis is similar to that of linear systems where only A is perturbed. For this problem, we make use of the known formulae for the backward error for standard eigenproblems. Let $\tilde{x}$ and $\tilde{\lambda}$ be respectively an approximate eigenvector and an approximate eigenvalue of the matrix A. We define $r = A\tilde{x} - \tilde{\lambda}\tilde{x}$ as the associated residual vector.

The *normwise* backward error associated with $(\tilde{\lambda}, \tilde{x})$ is defined by

$$\begin{aligned}\eta^{\mathcal{N}}(\tilde{\lambda}, \tilde{x}) &= \min\{\omega; \|\Delta A\| \leq \omega\alpha, \text{ such that } (A+\Delta A)\tilde{x} = \tilde{\lambda}\tilde{x}\}, \\ &= \frac{\|r\|}{\alpha\|\tilde{x}\|}.\end{aligned}$$

We note that, if $x' = k\tilde{x}$ and $r' = Ax' - \lambda' x'$, then we have $\frac{\|r'\|}{\alpha\|x'\|} = \frac{\|r\|}{\alpha\|\tilde{x}\|}$. Consequently, the backward error does not depend on the normalization chosen for the approximate eigenvector. The *componentwise* backward error associated with the approximations $\tilde{\lambda}$ and $\tilde{x}$ is defined by:

$$\begin{aligned}\eta^{\mathcal{C}}(\tilde{\lambda}, \tilde{x}) &= \min\{\omega; |\Delta A| \leq \omega E, \text{ such that } (A+\Delta A)\tilde{x} = \tilde{\lambda}\tilde{x}\} \\ &= \max_{1\leq i\leq n} \frac{|r|_i}{(E|\tilde{x}|)_i}.\end{aligned}$$

For each size t of perturbations, we compute a sample of eigenvalues Λ_t and the corresponding eigenvectors X_t, along with a sample of residuals $Y_t = AX_t - \Lambda_t X_t$. Let the eigenvalue samples X_t have mean m_t and standard deviation σ_t, and let the residual samples Y_t have mean ρ_t and standard deviation v_t. The means m_t and ρ_t can be real or complex, while the standard deviations are always in $\mathbb{R}$. Note that since the residuals are centered about zero, there is no biasing of the samples Y_t.

The formulae for the indicators of standard eigenproblems are shown in Table 3.3 for normwise and componentwise perturbation types. If λ is an eigenvalue of multiplicity $m > 1$, then Hölder sensitivities should be used; so these are also listed.

3.1.4 Interpreting Indicators

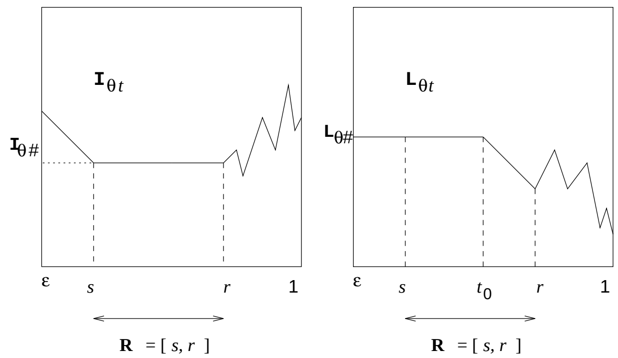


Figure 3.1: General behaviour of the statistical indicators $\mathtt{I}_{\theta t}$ and $\mathtt{L}_{\theta t}$

Reliability The generic behaviour of $t \longmapsto \mathtt{I}_{\theta t}$ and $t \longmapsto \mathtt{L}_{\theta t}$ is given in Figure 3.1. From left to right, t ranges from small to large perturbations. Observe that the map $t \longmapsto \mathtt{I}_{\theta t}$ is constant whenever the backward error at X_t is of order 1 in t. Therefore, this leads to the following key observation:

The existence of an interval where $\mathtt{I}_{\theta t}$ is *constant* in t implies that s is an approximation of the backward error for the given perturbation model, and the remaining indicators are valid on this constant interval.

	Normwise	Componentwise
$\mathtt{I}_{\theta t}$	$\dfrac{1}{t}\dfrac{\Vert(\sqrt{v_{ti}^2+\vert\rho_{ti}\vert^2})_i\Vert}{\alpha\Vert x_\theta\Vert}$	$\dfrac{1}{t}\max_i \dfrac{\sqrt{v_{ti}^2+\vert\rho_{ti}\vert^2}}{(E\vert x_\theta\vert)_i}$
$\mathtt{L}_{\theta t}$	$\dfrac{\sigma_t}{t\vert\lambda_\theta\vert}$	$\dfrac{\sigma_t}{t\vert\lambda_\theta\vert}$
$\mathtt{K}_t$	$\dfrac{\sigma_t}{\vert\lambda_\theta\vert}\dfrac{\alpha\Vert x_\theta\Vert}{\Vert v_t\Vert}$	$\dfrac{\sigma_t}{\vert\lambda_\theta\vert}\dfrac{1}{\max_i \dfrac{v_{ti}}{(E\vert x_\theta\vert)_i}}$
$\mathtt{E}_{\theta t}$	$\dfrac{\sigma_t}{\vert\lambda_\theta\vert}\dfrac{\Vert Ax_\theta-\lambda_\theta x_\theta\Vert}{\Vert v_t\Vert}$	$\dfrac{\sigma_t}{\vert\lambda_\theta\vert}\dfrac{\max_i \dfrac{\vert Ax_\theta-\lambda_\theta x_\theta\vert_i}{(E\vert x_\theta\vert)_i}}{\max_i \dfrac{v_{ti}}{(E\vert x_\theta\vert)_i}}$
$\mathtt{K}_t^h$	$\dfrac{\sigma_t}{\vert\lambda_\theta\vert}\left(\dfrac{\alpha\Vert x_\theta\Vert}{\Vert v_t\Vert}\right)^h$	$\dfrac{\sigma_t}{\vert\lambda_\theta\vert}\dfrac{1}{\max_i \left(\dfrac{v_{ti}}{(E\vert x_\theta\vert)_i}\right)^h}$
$\mathtt{E}_{\theta t}^h$	$\dfrac{\sigma_t}{\vert\lambda_\theta\vert}\left(\dfrac{\Vert Ax_\theta-\lambda_\theta x_\theta\Vert}{\Vert v_t\Vert}\right)^h$	$\dfrac{\sigma_t}{\vert\lambda_\theta\vert}\left(\dfrac{\max_i \dfrac{\vert Ax_\theta-\lambda_\theta x_\theta\vert_i}{(E\vert x_\theta\vert)_i}}{\max_i \dfrac{v_{ti}}{(E\vert x_\theta\vert)_i}}\right)^h$

Table 3.3: Statistical indicators for standard eigenproblems $Ax = \lambda x$ for normwise and componentwise perturbation models. The scaling value $\alpha = \Vert A\Vert$ is commonly used for a normwise analysis, while $E = \vert A\vert$ is commonly used for a componentwise analysis.

The value s in Figure 3.1 is the lowest perturbation value of t such that the approximate method G_θ, considered as a perturbation of F^{-1}, behaves in a way which is indistinguishable from that of F^{-1}. It therefore represents an estimate of the backward error with respect to the perturbation model which was used to produce the samples. That is, if x_θ is the solution to $F(x) = y$ computed by G_θ; then s is of the order of the backward error at x_θ associated with the model of perturbations. The value s is also the lower threshold of reliability for the estimations given by PRECISE.

On the constant interval, the perturbations in the input data are consistent with the resulting perturbations in the solutions. Hence, G_θ can indeed be considered an approximate inverse of F. We term the interval $\mathbf{R} = [s, r]$ on which $\mathtt{I}_{\theta t}$ is constant *the reliability interval* for G_θ to solve $F(x) = y$. Observe that $\mathtt{I}_{\theta t}$ should be of order 1 if the scaling values for $\|\hat{v}_t\|$ are chosen appropriately. However, the actual value of $\mathtt{I}_{\theta t}$ is usually not of critical importance, it need only be constant with respect to t for a reliable analysis.

If no reliability interval is observed, then one possibility is that the problem is too unstable for the attempted perturbation interval. Another possibility is that the algorithm G or implementation G_θ is not backward stable, *i.e.* $s \geq 1$. In any case, if no reliability interval is observed, then the remaining indicators do not provide any valid information. The condition for backward stability within finite precision is of course $s \leq \epsilon$, where ϵ is the working precision.

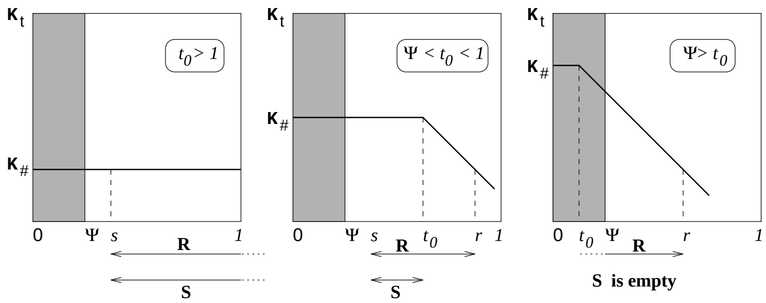


Figure 3.2: Machine precision Ψ and dangerous border t_0

Conditioning and Error Estimation The generic behaviour of $t \longmapsto \mathtt{K}_t$ is shown on Figure 3.2. The grey regions correspond to the interval $[0, \Psi]$ which are not accessible with finite precision computations. In general the indicator $\mathtt{K}_t$ is constant on an interval $\mathbf{S} = [s, t_0[$, with $t_0 \leq r$ which is included in the reliability interval $\mathbf{R}$, so $\mathbf{S} \subset \mathbf{R}$. We term the interval $\mathbf{S}$ the *regularity interval*, and the value t_0 for which $\mathtt{K}_t$ ceases to be constant the *dangerous border*. It is possible that $t_0 > 1$, so $\mathbf{S}$ cannot be distinguished from $\mathbf{R}$ (Figure 3.2, left). It is also possible that $t_0 < \Psi$ so that $\mathbf{S}$ is empty (Figure 3.2, right). If it exists, the dangerous border t_0 can be viewed as a representation of the order of the distance to a singularity relative to the model of perturbations.

If the regularity interval $\mathbf{S}$ is not empty, the constant value $\mathtt{K}_\sharp$ for $\mathtt{K}_t$ is of the order of the condition number for F^{-1}. In particular, condition number estimates with respect to the perturbation model for a computed solution x_θ are provided by the constant value of $\mathtt{K}_\sharp$ on the interval $\mathbf{S}$. This value can then be used to provide an estimated upper bound for the computed solution x_θ. The relative forward error is approximated by

$$\frac{\|x - x_\theta\|}{\|x\|} \sim \mathtt{K}_\sharp \times s. \tag{3.10}$$

In cases where the algorithm appears to be backward stable but the indicator $\mathtt{K}$ is increasing or decreasing with no clear sensitivity measure, the perturbation values are likely producing nonlinear behaviour on the solution. In such cases, the Hölder sensitivity measures $\mathtt{K}^{\mathtt{h}}$ are more appropriate. The order h may not be known in advance, and so the behaviour for a few different orders can be calculated. Interestingly, the appropriate order h may depend on the perturbation model, and can actually change for different perturbation sizes in an interval. Such cases have been observed for multiple, defective eigenvalues, in which sensitivity of the problem causes the numerical procedure to "see" the eigenvalue differently according to the perturbation sizes [9]. Thus, the Hölder sensitivity indicators $\mathtt{K}^{\mathtt{h}}$ provide an amazingly insightful characterisation of numerical problems near singularities.

Remarks The analysis of a numerical problem with Module 1 is frequently an iterative and interactive process. First, a perturbation model is defined and indicator values are obtained; subsequently, further perturbation models are tested to confirm or illuminate certain behaviours. The entire process results in an improved, more thorough understanding of both the mathematical problem and the numerical algorithm used for its solution.

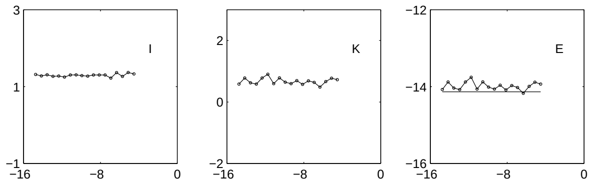
In summary, an analysis with this module requires the following procedure:

- Choose a perturbation model in which perturbations of size t are selected from an interval.
- Compute samples by performing the algorithm repeatedly for each perturbation value t.
- Plot the maps $t \longmapsto \mathtt{I}_{\theta t},\ \mathtt{K}_t$.
- Determine the reliability interval $\mathbf{R} = [s, r]$ and the regularity interval $\mathbf{S} = [s, t_0[$.
- For regular problems, $t_0 = r$ and $\mathbf{R} = \mathbf{S}$. Then if $t_0 > \Psi$ where Ψ is the working precision, the error estimation is $\mathtt{E} = \mathtt{K}_\sharp \times s$. The distance to singularity viewed by the computer is given by t_0. If $t_0 < \Psi$, the problem is too ill-conditioned for the working precision.

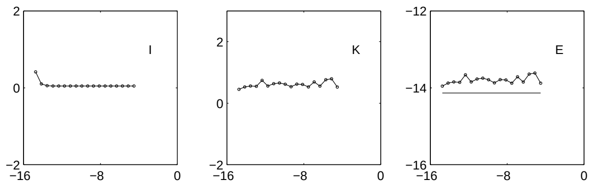
In the following section, a brief example of the analysis process is given.

3.1.5 Example Backward Error Analysis

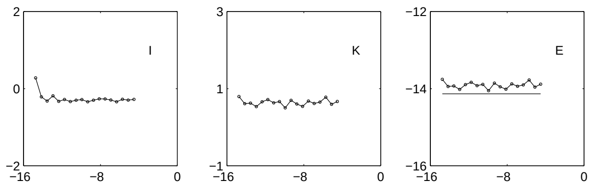
Using a GMRES algorithm, a system $Ax = b$ is studied, where A is a symmetric Hankel matrix of order $N = 100$ with elements $a_{ij} = 0.5/(N - i - j + 1.5)$. The restart parameter of GMRES was $m = 20$. To allow a comparison of the PRECISE estimated forward error, a solution $x = (1, 1, \ldots, 1)^T$ is chosen and used to form the right hand side b. Studies were performed by applying normwise and componentwise perturbations on each of A and b individually. For all cases, the perturbations ranged from $t = 2^{-49}$ to $t = 2^{-10}$, with 10 samples used for each perturbation size t. The computations were performed with double precision arithmetic 2^{-16} precision. A convergence criteria of $\dfrac{||r||}{||b||} \leq 10^{-14}$ was used, where $r = b - A\tilde{x}$. The statistical estimators obtained for each case are shown in Figure 3.3; the vertical and horizontal coordinates are shown using a logarithmic scale.



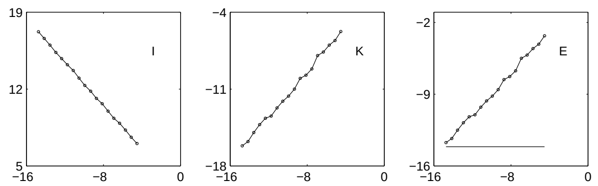
Normwise perturbations on A



Normwise perturbations on b



Componentwise perturbations on A



Componentwise perturbations on b

Figure 3.3: Behaviour of the statistical estimates $\mathtt{I}_{\theta t}, \mathtt{K}_t$, and forward error estimates $\mathtt{E}$ shown with actual relative forward error (straight solid lines).

One may first observe that one of the perturbation models, the case with componentwise perturbations on just the right-hand-side b, has no reliability interval, since the indicator $\mathtt{I}$ is non-constant throughout the range of perturbation sizes. This reveals that componentwise perturbations on the elements of b do not produce corresponding variations in the solution. For a backward error analysis, therefore, no information is revealed by this perturbation model. On the other hand, the remaining three models show reliability intervals of approximately $\mathbf{R} = [10^{-15}, 10^{-4}]$, with the upper bound reaching the end of the perturbation range. The backward error of the solutions for each case are therefore of the order 10^{-15}, which is very close to the stopping criteria used for the iterative solver, and also not far from the machine precision. Note that these three different perturbation models are, by definition, estimating the backward error with different normalisations. This is why the value of $\mathtt{I}$ is not the same in each case; it is of magnitude 10^1 for normwise perturbations on A, while it is of magnitude 10^0 for the other two cases with a non-empty reliability interval.

Since the reliability interval is non-empty for three of the cases, the remaining indicators for these cases are also revealing. The conditioning of the system is indicated by `K`, which is of roughly of order 10^1. Thus, the system is well conditioned. As can be seen in the estimates of the forward error, this gives a very good error bound. The indicator `E` shows values of 10^{-14}, which is in fact very close to the true forward error (recall that we started with a known solution, making this value available).

3.2 Implementation Issues

A backward error analysis requires a small amount of programming effort. The first task is to implement the perturbation model, including the iteration to produce samples of perturbed solutions for each perturbation size. A second task involves processing the samples to calculate the statistical indicators. The processing may be done simultaneously while the data is generated, or the samples may be written to a file and the indicators calculated in a separate phase. A final task is to visualise the stability indicators once they have been calculated. This task depends on the type of graphical software available to the user. These issues are examined in the following sections. For examples of completed implementations of backward error analysis routines using popular software libraries, see Section 3.4.

3.2.1 Implementing a Perturbation Model

A general discussion of perturbation models is given in Section 2.2. Here, some practical concerns for implementing a perturbation model are discussed. One should also be aware of these issues when deciding what perturbation model(s) to apply for a study with PRECISE; the choice of perturbation model may depend on how it is to be implemented.

The main idea is to use perturbation sizes t taken from a range of values $[\xi^{-k_{\max}}, \xi^{-k_{\min}}]$. For each perturbation size, a number of samples of perturbed solutions shall be produced. A starting point is how to specify the perturbation values. Since in general a wide range of values is required which spans orders of magnitude, a simple method is to specify the lower and upper exponents of the range. That is, using some base value such as $\xi = 2$ for instance, one need only specify the range $[\xi^{-k_{\max}}, \xi^{-k_{\min}}]$ by providing $k_{\min}$ and $k_{\max}$. Any base value ξ can be used. However, the particular choices of $\xi = 10$ or $\xi = 2$ are usually the most convenient; the former value allows perturbations of corresponding digits in the data components, while the latter value corresponds to bit-wise perturbations of the data. To store the perturbation values in an array, a counter k can be incremented from $k_{\min}$ and $k_{\max}$ to produce a uniform sample of perturbation values t. In a distributed parallel environment, each processor can perform this operation without communication, as long as $k_{\min}$ and $k_{\max}$ are constants known to all processors. Note that the particular order in which the perturbation sizes are used to produce indicators is usually not important; they can be increasing or decreasing in size during the indicator calculations as long as the correct order is presented during their visualisation.

A number of samples will be generated for each perturbation size t. The number of samples does not need to be large, usually between 5 to 15 samples gives very satisfactory results [5]. The number of samples should remain constant for all the different perturbation values taken from the perturbation range.

A random number generator is also required. All the routines in PRECISE use the routine named `RANDOM`, which can be modified according to system requirements (see Sec-

$\boxed{\texttt{solve } F(x) = y.} \Longrightarrow x, F(x)$

`for each perturbation size t:`

`for each sample:`

$\boxed{\texttt{perturb problem } \tilde{F}(x) = \tilde{y}.}$

$\boxed{\texttt{solve } \tilde{F}(\tilde{x}) = \tilde{y}.} \Longrightarrow \tilde{x}, F(\tilde{x})$

Figure 3.4: General scheme for generating data for a backward error analysis. The data used for the analysis is indicated by arrows ($\Longrightarrow$). Note that after each perturbed solution $\tilde{x}$ is obtained, the unperturbed mapping F (not $\tilde{F}$) is used to generate the image $F(\tilde{x})$.

tion 3.3). If perturbations are to be produced on distributed data in a parallel environment, care should be taken to ensure the sequences of pseudorandom numbers on each processor are independent; this can be done by using different seeds on each processor.

Another issue concerns perturbations of real or complex data; that is, data in $\mathbb{R}$ or $\mathbb{C}$. When randomly perturbing a real data value x by a perturbation size t, the result is either $x+t$ and $x-t$, each having equal probability. However, when randomly perturbing a complex data value, care should be taken since random number generators usually produce only real random (or more precisely, pseudorandom) numbers. To perturb a complex data value z by a perturbation size t, let r represent a random number between 0 and 1, then a valid perturbation is given by $z + w$, where $w = te^{i2\pi r}$.

In some cases, the perturbations may need to be adjusted to suit a particular data metric. For example, when performing a normwise analysis on matrices, care should be taken to ensure the inequalities defining normwise perturbations are satisfied. The type of norm used may require additional scaling of the perturbation values. In particular, using the $\|\cdot\|_\infty$ matrix norm, the perturbation on each component should be scaled according to the number of non-zero row elements. This is particularly important for large, sparse matrices, where one usually only chooses to perturb the non-zero elements since algorithms for these problems are designed specifically to manipulate only these elements.

A diagram of the general procedure for producing the sample data is given in Figure 3.4. First, the numerical problem under study is solved without introducing user-defined perturbations. Following this, an iteration is required, where a number of samples of perturbed problems are solved for each perturbation size. For each perturbed solution $\tilde{x}$ obtained, both $\tilde{x}$ and the value obtained by the reverse mapping $F(\tilde{x})$ are required for the analysis. Note that in general $F(\tilde{x})$ differs from $\tilde{F}(\tilde{x})$, and only the former quantity is needed. For example, if the problem under study is a linear system $Ax = b$ and the perturbed problem $\tilde{A}\tilde{x} = \tilde{b}$ is solved, then the quantities used for the analysis are $\tilde{x}$ and $A\tilde{x}$.

For most applications, it should be quite straight-forward to place the numerical solver in an iterative procedure as shown in Figure 3.4. In general, the iteration requires a separate copy of all the input data for the problem to be saved so that it will be available for each subsequent perturbed problem. If this is not feasible due to memory constraints, the initial data for the problem may be re-loaded from disk files for each perturbation. While this can be time-consuming, the issue of efficiency is not the primary concern during a backward error analysis. If the numerical solver has been designed to work in a parallel environment, one should ensure that the processors are synchronised throughout the iteration.

The example software discussed in Section 3.3 can be used to generate perturbations for most types of problems. In those examples, routines to produce perturbations on vectors can be used for matrices as well. Modifications to these routines to produce more complicated perturbation models can be easily performed. In addition, other perturbation techniques are possible for special cases. For instance, since iterative algorithms for large, sparse matrices usually access the matrix data only through matrix-vector multiplications, it may be worthwhile to perturb only the vector resulting from the matrix-vector product. As explained above, any approach to producing perturbations can produce valid results in the context of PRECISE.

3.2.2 Calculating Indicators

The data produced by the iteration shown in Figure 3.4 can be saved to a file or used immediately to calculate the statistical indicators. The choice depends on memory restrictions as well as convenience. The indicators for a particular perturbation size can be calculated as soon as the specified number of samples are completed; however, the sample data X_t and the corresponding image values Y_t must be held until all the samples for that perturbation size are available. If this requires a large amount of memory, the best solution may be to write the samples X_t and Y_t to a temporary disk file. In a parallel environment, it also may be easier to use a temporary disk file rather than add additional communication facilities to the numerical solver. In general, if the numerical solver has a somewhat involved interface for such items as a parallel environment or reverse communication, then it may be more convenient to write the data to a file and calculate the stability indicators in a separate phase. Otherwise, the indicators can be calculated immediately as soon as the samples are available. Examples of both of these techniques are given in Section 3.4.

The routines to accumulate the samples and calculate indicators for linear systems (`BESYMN, BESYST`) and standard eigenproblems (`BESEMN, BESEST`) are described in Section 3.3. The general procedure for the calculation of the indicators is described here so that the techniques can be adapted to other numerical problems.

For a fixed perturbation size t, we assume that the samples X and Y are held in storage, where X_i and Y_i denote the i^{th} sample. The sample values X_i and Y_i can be scalar or vector valued, depending on the numerical problem. For example, in the case of standard eigenproblems, the term X_i denotes the i^{th} eigenvalue using perturbations of size t, and Y_i denotes the corresponding i^{th} residual value.

The first step is to compute the means m_t and ρ_t of X and Y, respectively. If the samples contain vectors, then the means should be computed for each component, resulting in a vector of the same dimension. The metric defined by the perturbation model is used to evaluate the vectors to obtain the indicator values.

After the means are determined, then the standard deviations σ_t and v_t can be calculated. Note that this involves a second pass through the sample data; the two-pass formula used for N data elements x with mean $\bar{x}$ and standard deviation σ_x is:

$$\sigma_x = \left[\frac{1}{N-1}\left(\sum_j^N (x_j-\bar{x})^2 - \frac{1}{N}\Big[\sum_j^N (x_j-\bar{x})\Big]^2\right)\right]^{1/2} \tag{3.11}$$

We recommend using the two-pass method to calculate standard deviations, as rounding errors in a one-pass method can give results which are far from accurate [16]. Note that in complex arithmetic, conjugate values should be used for the multiplications. For vectors, the standard deviations σ_t and v_t are computed for each component according to Equation 3.11.

Once the standard deviations are computed, the indicators may be calculated based on the appropriate equations for the problem type and perturbation model.

3.2.3 Visualising Indicators

The calculated indicator values are usually written to file and plots produced with separate visualisation software. Any software which can produce two-dimensional X-Y plots can be used. The perturbation values serve as the abscissa for all the plots of $\mathtt{I_t}, \mathtt{K_t}, \mathtt{L_t}, \mathtt{E_t}, \mathtt{K_t^h}$ and $\mathtt{E_t^h}$. Since the values usually span many orders of magnitude, a logarithmic scale should be used for both axes; care is then needed in cases where the calculated values are close to zero.

To expediate the interpretation of the plots, it is usually best to start by examining only the plots of $\mathtt{I_t}, \mathtt{K_t}$, and perhaps also $\mathtt{E_t}$. If investigation of these plots reveals the necessity, plots of the other indicators can then be generated.

Example MATLAB routines to generate plots of the indicators can be found in Appendix B.

3.3 Module 1 Subroutines

Subroutines to perform a backward error analysis are described in this section. For linear systems and standard eigenvalue problems, these subroutines can be used without modification; for other types of problems, they may also be modified to suit more particular cases. The example implementations with existing software libraries described in Section 3.4 can also serve as useful starting references. The user should feel at liberty to examine the subroutines closely and make modifications as necessary. As long as the methodology of performing a backward error analysis is followed, the subroutines can be altered to suit any type of numerical problem.

There are two categories of subroutines. The first is used to implement the perturbation model for the problem under study. The perturbation subroutines are applicable to data stored in arrays; as such they should be generic enough to be used for any type of numerical problem. The subroutines in the second category are applicable to linear systems and standard eigenproblems. These subroutines calculate the indicators as described in Sections 3.1.2 and 3.1.3.

It is assumed the user has a numerical problem to be solved and has already implemented a numerical procedure to solve it. As explained in Section 3.2.1, the numerical procedure will be placed in an iteration in which the perturbation routines are used to produce multiple solution samples for each perturbation size. The samples $\mathtt{X_t}$ and $\mathtt{Y_t}$ are input parameters for the subroutines in the second category. The second category subroutines are organized so that they can also be called inside the iteration producing the solution samples. That is, the solution samples can be passed to these subroutines one at a time as they are generated, and the indicators will be calculated as soon as the required number of samples are reached.

The structure of the second category routines, therefore, allows great flexibility in how the sample data is manipulated. One technique is to write the samples $\mathtt{X_t}$ and $\mathtt{Y_t}$ to a temporary file and subsequently read them in a separate program which uses only subroutines from the second category. This technique is perhaps the simplest method from an implementation point of view. Another technique is to use the second category routines directly in the same iteration as the numerical procedure, making the process more efficient. Examples of these two techniques are given in Section 3.4.

The subroutines are applicable to both sequential and parallel numerical solvers. In the case of parallel solvers, the use of a temporary file to hold the samples $\mathtt{X_t}$ and $\mathtt{Y_t}$ is the most convenient technique. However, an example of how to hold the samples in storage and calculate the indicators during the iteration can be found in Section 3.4 for the NAG Parallel Library.

3.3.1 Subroutines for Perturbations Models

Subroutines to implement the user-defined perturbation model are described in this section. In particular, subroutines are specified to produce both normwise and componentwise perturbation types. The specification of a subroutine to generate random numbers is also provided, as well as a subroutine used to evaluate the data metric. These subroutines can be used as a starting point for the development of other perturbation models. The subroutines for perturbation models are summarised in Table 3.4.

Note that in these routines, there is a significant difference between the perturbations on real versus complex data. In the real case, perturbations of size t on a data value x will result in one of the values $x \pm t$, each with equal probability. In the complex case, the perturbed value will be of the form $x + te^{i2\pi r}$, where r is a uniform random variable from the interval $[0, 1]$.

Subroutine	Description
`BEDNPTB`	Normwise perturbation types on `DOUBLE PRECISION` data.
`BEDCPTB`	Componentwise perturbation types on `DOUBLE PRECISION` data.
`BEZNPTB`	Normwise perturbation types on `DOUBLE COMPLEX` data.
`BEZCPTB`	Componentwise perturbation types on `DOUBLE COMPLEX` data.
`RANDOM`	Generates random numbers in the interval $[0, 1]$.
`BENORM`	Calculates the norm of a vector as defined by the perturbation model.

Table 3.4: Subroutines for perturbation models.

BEDNPTB

Description

The routine BEDNPTB performs normwise perturbations on data components stored in DOUBLE PRECISION arrays. The arrays are treated as vectors with the increment specified as a parameter; this is similar to the mechanism for BLAS routines. To perform perturbations on matrices stored in two-dimensional arrays, this routine may be used to perturb each matrix column. Each data component v is randomly perturbed by the value p to one of the values $v+p$ or $v-p$, each with equal probability. This routine calls the external function RANDOM to generate the random numbers.

Specification

```
      SUBROUTINE BEDNPTB(N,V,INCV,PVAL)
      INTEGER N,INCV
      DOUBLE PRECISION V(*)
      DOUBLE PRECISION PVAL
C     ... External Functions ...
      DOUBLE PRECISION FUNCTION RANDOM
```

Arguments

N - INTEGER (Input)
The number of data components in V to perturb.

V - DOUBLE PRECISION (*) (Input/Output)
Array of size at least N*INCV containing the elements to be perturbed. The data values will be modified to contain the new perturbed values.

INCV - INTEGER (Input)
The increment for the loop counter.

PVAL - DOUBLE PRECISION (Input)
The perturbation value p. If this value is zero, then the routine returns immediately, and the other parameters are not referenced.

BEDCPTB

Description

The routine BEDCPTB performs componentwise perturbations on data components stored in DOUBLE PRECISION arrays. The arrays are treated as vectors with the increment specified as a parameter; this is similar to the mechanism for BLAS routines. To perform perturbations on matrices stored in two-dimensional arrays, this routine may be used to perturb each matrix column. Each data component v is randomly perturbed by the value p to one of the values $v + p|v|$ or $v - p|v|$, each with equal probability. This routine calls the external function RANDOM to generate the random numbers.

Specification

```
      SUBROUTINE BEDCPTB(N,V,INCV,PVAL)
      INTEGER N,INCV
      DOUBLE PRECISION V(*)
      DOUBLE PRECISION PVAL
C     ... External Functions ...
      DOUBLE PRECISION FUNCTION RANDOM
```

Arguments

N - INTEGER (Input)
The number of data components in V to perturb.

V - DOUBLE PRECISION (*) (Input/Output)
Array of size at least N*INCV containing the elements to be perturbed. The data values will be modified to contain the new perturbed values.

INCV - INTEGER (Input)
The increment for the loop counter.

PVAL - DOUBLE PRECISION (Input)
The perturbation value p. If this value is zero, then the routine returns immediately, and the other parameters are not referenced.

BEZNPTB

Description

The routine BEZNPTB performs normwise perturbations on data components stored in DOUBLE COMPLEX arrays. The arrays are treated as vectors with the increment specified as a parameter; this is similar to the mechanism for BLAS routines. To perform perturbations on matrices stored in two-dimensional arrays, this routine may be used to perturb each matrix column. Each data component v is randomly perturbed by the value p to the value $v + w$, where $w = pe^{i2\pi r}$ with r a uniform random variable from the interval $[0, 1]$. This routine calls the external function RANDOM to generate the random numbers.

Specification

```
      SUBROUTINE BEZNPTB(N,V,INCV,PVAL)
      INTEGER N,INCV
      DOUBLE COMPLEX V(*)
      DOUBLE COMPLEX PVAL
C     ... External Functions ...
      DOUBLE PRECISION FUNCTION RANDOM
```

Arguments

N - INTEGER (Input)
: The number of data components in V to perturb.

V - DOUBLE COMPLEX (*) (Input/Output)
: Array of size at least N*INCV containing the elements to be perturbed. The data values will be modified to contain the new perturbed values.

INCV - INTEGER (Input)
: The increment for the loop counter.

PVAL - DOUBLE COMPLEX (Input)
: The perturbation value p. If this value is zero, then the routine returns immediately, and the other parameters are not referenced.

BEZCPTB

Description

The routine BEZCPTB performs normwise perturbations on data components stored in DOUBLE COMPLEX arrays. The arrays are treated as vectors with the increment specified as a parameter; this is similar to the mechanism for BLAS routines. To perform perturbations on matrices stored in two-dimensional arrays, this routine may be used to perturb each matrix column. Each data component v is randomly perturbed by the value p to the value $v+w|v|$, where $w = pe^{i2\pi r}$ with r a uniform random variable from the interval $[0,1]$. This routine calls the external function RANDOM to generate the random numbers.

Specification

```
      SUBROUTINE BEZCPTB(N,V,INCV,PVAL)
      INTEGER N,INCV
      DOUBLE COMPLEX V(*)
      DOUBLE COMPLEX PVAL
C     ... External Functions ...
      DOUBLE PRECISION FUNCTION RANDOM
```

Arguments

N - INTEGER (Input)
: The number of data components in V to perturb.

V - DOUBLE COMPLEX (*) (Input/Output)
: Array of size at least N*INCV containing the elements to be perturbed. The data values will be modified to contain the new perturbed values.

INCV - INTEGER (Input)
: The increment for the loop counter.

PVAL - DOUBLE COMPLEX (Input)
: The perturbation value p. If this value is zero, then the routine returns immediately, and the other parameters are not referenced.

RANDOM

Description

The function `RANDOM` returns a `DOUBLE PRECISION` random value r in the interval $[0, 1]$ with a uniform distribution.

This subroutine is provided as an interface to the pseudorandom number generator. Because there is no standard random number generator defined in FORTRAN, this subroutine is system dependent and must be modified to suit the particular requirements of the system.

In practice, a system-supplied pseudorandom number generator may not exhibit optimal distribution qualities; whenever possible, a separate, well-constructed generator is recommended.

Specification

```
DOUBLE PRECISION FUNCTION RANDOM (SEED)
INTEGER SEED
```

Arguments

`SEED` - `INTEGER` (Input)

Seed value. On entry, if `SEED` is zero then the next value r is returned from the pseudorandom number generator. If `SEED` is not zero on entry, then this value is used as a seed to reset the pseudorandom number generator.

BENORM

Description

The function `BENORM` is used to evaluate a vector `X` and return the norm value as determined by the data metric of the perturbation model. The vector `X` is assumed to be from the sample data $\mathtt{X_t}$ or $\mathtt{Y_t}$ as described in Section 3.2.2; the array is assumed to contain contiguous components and therefore no increment parameter is used.

The subroutine has been supplied to allow a common interface for the data metric. The subroutine provided here computes and returns $\|X\|_\infty$.

A similar routine can be defined for `DOUBLE COMPLEX` vectors.

Specification

```
DOUBLE PRECISION FUNCTION BENORM(N,X)
INTEGER N
DOUBLE PRECISION X(*)
```

Arguments

`N` - `INTEGER` (Input)
Number of elements in the vector `X`.

`X` - `DOUBLE PRECISION (*)` (Input)
The function returns the norm value of the `N` elements in the array `X`. `X` must have at least `N` elements.

3.3.2 Subroutines for Linear Systems

Subroutines to collect the samples and calculate the indicators for linear system problems $Ax = b$ are described in this section. These routines are designed to be used iteratively in that they are called repeatedly, once for each sample solution which is produced from solving a perturbed linear system problem. The routines never access the matrix A, but will require the right-hand-side b, along with the vectors $\tilde{x}$ and $\tilde{y} = A\tilde{x}$ where $\tilde{x}$ is the solution obtained by the numerical procedure for the perturbed or unperturbed problems.

The general procedure for using these routines is depicted in Figure 3.5. The variable `NXFLAG` is used to provide a reverse communication interface; the unperturbed solution will be solved once, and perturbed solutions will be solved numerous times. The indicators $\mathtt{I_t}, \mathtt{K_t}, \mathtt{L_t}$ and $\mathtt{E_t}$ are calculated by `BEDSYST`, which is called by `BEDSYMN` when the required number of samples has been obtained for the perturbation size t. The subroutine `BEDSYFO` is used to write the indicator values to a file.

Subroutines for both `DOUBLE PRECISION` and `DOUBLE COMPLEX` data are provided. The subroutines for the two data types are similar in form, and so only the specifications for the `DOUBLE PRECISION` routines are given here.

These routines are not designed for parallel architectures, but may be modified to manipulate data which is distributed in a multi-processor environment; for an example of this, see the example implementation with the NAG Parallel Library in Section 3.4.

Subroutine	Description
`BEDSYMN`	Main driver to collect samples for `DOUBLE PRECISION` data.
`BEDSYST`	Calculate indicators.
`BEDSYFO`	Write out all indicator values to a file.
`BEZSYMN`	Main driver to collect samples for `DOUBLE COMPLEX` data.
`BEZSYST`	Calculate indicators.
`BEZSYFO`	Write out all indicator values to a file.

Table 3.5: Subroutines for linear systems. The names of double precision routines start with `BEDSYxx`, while double complex routines are named `BEZSYxx`.

```
     numptb  ← number of perturbation sizes
     numsamp ← number of sample sizes
     ptbs(i) ← perturbation sizes t

10   continue

     CALL BEDSYMN(..)

     if (NXFLAG.EQ.1) then

          ... solve unperturbed problem Ax = b
              (or read x from a file).

          ... y  ←  A x
              (or read y from a file).

          goto 10
     else if (NXFLAG.EQ.2) then

          ... solve a perturbed problem A x̃ = b
              (or read x̃ from a file).

          ... ỹ  ←  A x̃
              (or read ỹ from a file).

          goto 10
     else if (NXFLAG.EQ.3) then

          CALL BEDSYFO(..)
          ... stop

     end if
```

Figure 3.5: General procedure for using the linear system subroutines BEDSYMN and BEDSYFO. The subroutines BEZSYMN and BEZSYFO are used similarly.

BEDSYMN

Description

BEDSYMN is the main driver to collect samples and calculate indicators for linear system problems $Ax = b$ with DOUBLE PRECISION data. This subroutine is called repeatedly, each time with a single solution of the perturbed or unperturbed linear system. The number of perturbation sizes and the number of samples for each perturbation size must be determined prior to the first call to BEDSYMN. In addition, all the perturbation sizes are assumed to be stored in an array.

The routine is first called to initialize internal values. The second call is used to store the solution to the unperturbed problem. Subsequent calls are used to store the sample solutions to the perturbed problems. As soon as the number of samples is reached for a fixed perturbation size, the indicators are calculated and the workspace is reinitialized to store samples for the next perturbation size. In this way, only storage for the number of samples for *one* perturbation size is needed. To aid the user, a reverse-communication flag NXFLAG is provided to ensure the iteration is performed correctly. See Figure 3.5.

For componentwise models, observe that the vector value $\mathtt{E}|\mathtt{x}|+\mathtt{f}$ is required to calculate the indicators, where x is the solution to the unperturbed problem. If $\alpha = 0$, then only f is required, while if $\beta = 0$ only $E|x|$ is required. Thus, when the unperturbed problem is solved for a componentwise model, the user should subsequently calculate:

$$\begin{cases} \mathtt{E}|\mathtt{x}|+\mathtt{f} & \text{if } \alpha > 0 \text{ and } \beta > 0, \\ \mathtt{E}|\mathtt{x}| & \text{if } \alpha > 0 \text{ and } \beta = 0, \\ \mathtt{f} & \text{if } \alpha = 0 \text{ and } \beta > 0, \end{cases}$$

where E and f are determined by the perturbation model, the usual cases being $E = |A|$ and $f = |b|$. The resulting vector value should then be placed in the first N components of the workspace array.

Specification

```
      SUBROUTINE BEDSYMN(PFLAG,N,NUMPTB,NUMSAMP,ALPHA,BETA,X,Y,B,
     +                   XCALNORM,RESNORM,RWORK,LRW,KNTPTB,KNTSAMP,
     +                   PTBS,STATS,NEWPSZ,NXFLAG,INFO)
      INTEGER PFLAG,N,NUMPTB,NUMSAMP,LRW,KNTPTB,KNTSAMP,
     +        NEWPSZ,NXFLAG,INFO
      DOUBLE PRECISION ALPHA,BETA,XCALNORM,RESNORM
      DOUBLE PRECISION PTBS(*),RWORK(*),X(*),Y(*),B(*),STATS(4,*)
```

Arguments

PFLAG - INTEGER (Input)

Specifies what type of indicators are to be calculated. A value of 1 specifies normwise-type indicators are to be calculated, and a value of 2 specifies componentwise-type indicators. If PFLAG=2 is specified, then by default the componentwise perturbations use the values $E = |A|$ (if ALPHA > 0) and $f = |b|$ (if BETA > 0).

N - INTEGER (Input)

Specifies the order of the matrix A.

NUMPTB - INTEGER (Input)
Specifies the number of perturbation values t which are held in PTBS.

NUMSAMP - INTEGER (Input)
Specifies the number of samples used to calculate the indicators for each perturbation size. Recommended values for NUMSAMP are between 5 to 15. Large numbers of samples will require both more computational effort and more workspace.

ALPHA - DOUBLE PRECISION (Input)
Specifies the scaling used to produce the perturbations. For normwise-type models, the perturbations of size t will satisfy $\|\Delta A\| \leq t\alpha$. ALPHA must be greater or equal to zero. For both normwise and componentwise models, a value of zero indicates no perturbations on the matrix A.

BETA - DOUBLE PRECISION (Input)
Specifies the scaling used to produce the perturbations. For normwise-type models, the perturbations of size t will satisfy $\|\Delta b\| \leq t\beta$. BETA must be greater or equal to zero. For both normwise and componentwise models, a value of zero indicates no perturbations on the right-hand-side b.

X - DOUBLE PRECISION (*) (Input/Workspace)
Array of length at least N. If NXFLAG=1, then this array should hold on input the unperturbed solution x obtained from the numerical linear system solver. If NXFLAG=2, then this array should hold on input the perturbed solution $\tilde{x}$ obtained from the numerical linear system solver. When calculating indicators, this array will also be used as workspace.

Y - DOUBLE PRECISION (*) (Input/Workspace)
Array of length at least N. If NXFLAG=1, then this array should hold on input the result $y = Ax$, where x is the unperturbed solution obtained from the numerical linear system solver. If NXFLAG=2, then this array should hold on input the result $\tilde{y} = A\tilde{x}$, where $\tilde{x}$ is the perturbed solution obtained from the numerical linear system solver. Note that if the perturbed problem involved a perturbed matrix $A+\Delta A$, only A (and not $A + \Delta A$) should be used to calculate y. When calculating indicators, this array will also be used as workspace.

B - DOUBLE PRECISION (*) (Input)
Array of length at least N. On entry, contains the right-hand-side b of the linear system.

XCALNORM - DOUBLE PRECISION (Output)
Norm of the unperturbed solution x. On entry with NXFLAG=1 and PFLAG=1, this parameter is set to the value $\|X\|$ using the function BENORM. On entry with NXFLAG=1 and PFLAG=2, this parameter is set to the value $\max|X(I)|, I = 1,..,N$. This value should remain unchanged between subsequent calls.

RESNORM - DOUBLE PRECISION (Output)
Norm of the unperturbed image y. On entry with NXFLAG=1 and PFLAG=1, this parameter is set to the value $\|Y\|$ using the function BENORM. On entry with NXFLAG=1 and PFLAG=2, this parameter is set to the value $\max|Y(I)|, I = 1,..,N$. This value should remain unchanged between subsequent calls.

`RWORK` - `DOUBLE PRECISION (*)` (Input/Workspace)
Workspace of length at least `LRW` for holding samples and producing indicators.

For componentwise computations (`PFLAG=2`), if the solution `X` to the unperturbed problem has been requested (`NXFLAG=1`), then one of the following values must be computed and placed in the first `N` locations of `RWORK` before calling the routine again:

$$\begin{cases} \mathtt{E}|\mathtt{X}| + \mathtt{f} & \text{if } \alpha > 0 \text{ and } \beta > 0, \\ \mathtt{E}|\mathtt{X}| & \text{if } \alpha > 0 \text{ and } \beta = 0, \\ \mathtt{f} & \text{if } \alpha = 0 \text{ and } \beta > 0, \end{cases}$$

where E and f are determined by the perturbation model, the usual cases being $E = |A|$ and $f = |b|$.

`LRW` - `INTEGER` (Input)
Length of the workspace array `RWORK`. It must have a value at least `1+N*(2*NUMSAMP+1)`.

`KNTPTB` - `INTEGER` (Output)
For informational purposes, this parameter indicates which perturbation value in `PTBS` is currently being processed. This value must remain unchanged between subsequent calls.

`KNTSAMP` - `INTEGER` (Output)
For informational purposes, this parameter indicates which sample is needed for the current perturbation size. This value must remain unchanged between subsequent calls.

`PTBS` - `DOUBLE PRECISION (*)` (Input)
Contains the perturbation sizes used to produce the perturbed samples. The array size must be at least `NUMPTB`.

`STATS` - `DOUBLE PRECISION (4,*)` (Output)
When `NXFLAG=3`, this array holds the indicator values for the corresponding perturbation sizes in `PTBS`. The values $\mathtt{I_t}, \mathtt{K_t}, \mathtt{L_t}$ and $\mathtt{E_t}$ are held respectively in `PTBS(1,*)`, `PTBS(2,*)`, `PTBS(3,*)` and `PTBS(4,*)`. The subroutine `BEDSYFO` may be used to write these values and related information to a file.

`NEWPSZ` - `INTEGER` (Output)
For informational purposes, when `NXFLAG=2` this parameter indicates that the first sample for a perturbation value is requested. That is, `KNTPTB` has been incremented and `KNTSAMP` has been reset to 1. This may be used to monitor the progress of the calculation.

`NXFLAG` - `INTEGER` (Input/Output)
Reverse communication flag. On initial entry, this parameter must have a value of zero. On subsequent entries, this parameter should remain unchanged from the previous call.

On exit, a value of 1 specifies that the solution to the unperturbed problem is to be placed in `X`, and its image is to be placed in `Y`. Then the subroutine should be called again.

On exit, a value of 2 specifies that the solution to the perturbed problem is to be placed in `X`, and its image is to be placed in `Y`. Note that to obtain the image value `Y`, the product of `X` and the *unperturbed* matrix A should be used. Then the subroutine should be called again.

Conceivably, since the number of perturbations and the number of samples are known in advance, fixed `FOR`-loops may be used instead to perform the iteration. This option is available to the user, but the indicator flags should be used to monitor the progress of the iteration.

`INFO` - `INTEGER` (Input/Output)
Error flag. On initial entry, this value must be zero. On exit, a value of zero indicates no errors have occurred. A value other than zero indicates an error; an error message will be printed by the subroutine, and the iteration should be stopped.

BEDSYST

Description

BEDSYST is used to processes the sample data and calculate the indicators $\mathtt{I_t}, \mathtt{K_t}, \mathtt{L_t}$ and $\mathtt{E_t}$ for linear system problems $Ax = b$ in DOUBLE PRECISION. This subroutine is called once for each perturbation value t. This subroutine is called directly from BEDSYMN; the specification is provided here for those developing analysis routines for other types of numerical problems. See Section 3.2.2 for a discussion of how the indicators are calculated.

Specification

```
      SUBROUTINE BEDSYST(PFLAG,B,N,NUMSAMP,PTBVAL,KNTPTB,X,Y,
     +                   ALPHA,BETA,XCALNORM,RESNORM,
     +                   XSAMPS,YSAMPS,REFS,STATS)
      INTEGER PFLAG,N,NUMSAMP,KNTPTB
      DOUBLE PRECISION ALPHA,BETA,XCALNORM,RESNORM,PTBVAL
      DOUBLE PRECISION B(*),X(*),Y(*),XSAMPS(*),YSAMPS(*),REFS(*)
      DOUBLE PRECISION STATS(4,*)
```

Arguments

PFLAG - INTEGER (Input)
: Specifies what type of indicators are to be calculated. A value of 1 specifies normwise-type indicators are to be calculated, and a value of 2 specifies componentwise-type indicators. If PFLAG=2 is specified, then by default the componentwise perturbations use the values $E = |A|$ (if ALPHA > 0) and $f = |b|$ (if BETA > 0).

B - DOUBLE PRECISION (*) (Input)
: Array of length at least N. On entry, contains the right-hand-side b of the linear system.

N - INTEGER (Input)
: Specifies the order of the matrix A.

NUMSAMP - INTEGER (Input)
: Specifies the number of samples held in XSAMPS and YSAMPS.

PTBVAL - DOUBLE PRECISION (Input)
: Specifies the perturbation value t which was used to produce the samples.

KNTPTB - INTEGER (Input)
: Specifies which perturbation value in PTBS is currently being processed. The indicator values will be stored in the corresponding locations of the STATS array.

X - DOUBLE PRECISION (*) (Workspace)
: Used as local workspace.

Y - DOUBLE PRECISION (*) (Workspace)
: Used as local workspace.

ALPHA - DOUBLE PRECISION (Input)
    Specifies the scaling used to produce the perturbations.

BETA - DOUBLE PRECISION (Input)
    Specifies the scaling used to produce the perturbations.

XCALNORM - DOUBLE PRECISION (Input)
    Norm of the unperturbed solution x.

RESNORM - DOUBLE PRECISION (Input)
    Norm of the unperturbed image y.

XSAMPS - DOUBLE PRECISION (*) (Input/Workspace)
    Holds the values of the sample solutions $\mathtt{X_t}$ obtained from solving the perturbed problems. The array is also used as workspace during the calculation of the indicators.

YSAMPS - DOUBLE PRECISION (*) (Input/Workspace)
    Holds the values of the sample images $\mathtt{Y_t}$ obtained from solving the perturbed problems. The array is also used as workspace during the calculation of the indicators.

REFS - DOUBLE PRECISION (*) (Input)
    Holds the reciprical values of one of the scaling factors $E|x| + f$, $E|x|$, or f. These values are only referenced for componentwise indicators (PFLAG=2).

STATS - DOUBLE PRECISION (4,*) (Output)
    Holds the computed indicator values. The values $\mathtt{I_t}, \mathtt{K_t}, \mathtt{L_t}$ and $\mathtt{E_t}$ are stored respectively in PTBS(1,KNTPTB), PTBS(2,KNTPTB), PTBS(3,KNTPTB) and PTBS(4,KNTPTB).

BEDSYFO

Description

BEDSYFO is used to write out to a file all information obtained from a backward analysis in DOUBLE PRECISION. In addition to the indicator values I_t, K_t, L_t and E_t, some values useful for the interpretation of results are also written.

The format of the output file is:

```
PTYPE
ALPHA
BETA
XCALNORM
RESNORM
ERRNORM
NUMPTB
t_1          I_1          L_1          K_1          E_1
t_2          I_2          L_2          K_2          E_2
  ⋮
t_NUMPTB     I_NUMPTB     L_NUMPTB     K_NUMPTB     E_NUMPTB
```

The values t_i are the perturbation values used to produce the samples, while the values I_i, L_i, K_i, and E_i are the corresponding indicator values. Note that the order of the perturbation values is not specified, so t_1 may or may not be greater than t_{NUMPTB}; the visualisation tool should ensure that the plots are generated in the correct order. See Appendix B for an example of a Matlab source file which can be used to generate plots from the output file.

Specification

```
      SUBROUTINE BEDSYFO(OFILE,PTYPE,NUMPTB,PTBS,STATS,ALPHA,BETA,
     &                   XCALNORM,RESNORM,ERRNORM)
      CHARACTER*72 OFILE
      INTEGER PTYPE,NUMPTB
      DOUBLE PRECISION ALPHA,BETA,ERRNORM,XCALNORM,RESNORM
      DOUBLE PRECISION PTBS(*),STATS(4,*)
```

Arguments

OFILE - CHARACTER*72 (Input)
The name of the output file to store the results.

PFLAG - INTEGER (Input)
Specifies what type of indicators were calculated. A value of 1 specifies normwise-type indicators, and a value of 2 specifies componentwise-type indicators.

`NUMPTB` - `INTEGER` (Input)
Specifies the number of perturbation values t which are held in `PTBS`.

`PTBS` - `DOUBLE PRECISION (*)` (Input)
Contains the perturbation sizes used to produce the perturbed samples. The array size must be at least `NUMPTB`.

`STATS` - `DOUBLE PRECISION (4,*)` (Input)
This array holds the indicator values for the corresponding perturbation sizes in PTBS. The values $\mathtt{I_t}, \mathtt{K_t}, \mathtt{L_t}$ and $\mathtt{E_t}$ are held respectively in `PTBS(1,*)`, `PTBS(2,*)`, `PTBS(3,*)` and `PTBS(4,*)`.

`ALPHA` - `DOUBLE PRECISION` (Input)
Specifies the scaling used to produce the perturbations.

`BETA` - `DOUBLE PRECISION` (Input)
Specifies the scaling used to produce the perturbations.

`XCALNORM` - `DOUBLE PRECISION` (Input)
Norm of the unperturbed solution x. If `PFLAG=1`, this value is $||x||$, calculated using the function `BENORM`. If `PFLAG=2`, this value is $\max|x(i)|, i = 1, .., N$.

`RESNORM` - `DOUBLE PRECISION` (Input)
Norm of the unperturbed image y. If `PFLAG=1`, this value is $||y||$, calculated using the function `BENORM`. If `PFLAG=2`, this value is $\max|y(i)|, i = 1, .., N$.

`ERRNORM` - `DOUBLE PRECISION` (Input)
Norm of the error between the known true solution x_k and the unperturbed solution x_c obtained by the numerical procedure. This value can be calculated by the user if the true solution x_k to the analytical problem is known; it provides a means to compare the actual forward error with the forward error bound $\mathtt{E_t}$ determined by PRECISE. If x_k is not known, set this value to zero. if x_k is known and `PFLAG=1`, set this value to $||x_k - x_c||$. if x_k is known and `PFLAG=2`, set this value to $\max|x_k(i) - x_c(i)|, i = 1, .., N$.

3.3.3 Subroutines for Eigenproblems

Subroutines to collect the samples and calculate the indicators for standard eigenproblems $Ax = \lambda x$ are described in this section. These routines are designed to be used iteratively in that they are called repeatedly, once for each sample solution which is produced from solving a perturbed eigenproblem. The routines never access the matrix A, but will require the values $\tilde{\lambda}$, $\tilde{x}$, and $\tilde{y} = A\tilde{x}$, where $\tilde{\lambda}$ and $\tilde{x}$ are the eigenvalue and eigenvector, respectively, obtained by the numerical procedure for the perturbed or unperturbed problems. Note that the analysis is performed for only a single eigenvalue and its corresponding eigenvector.

The general procedure for using these routines is depicted in Figure 3.6. The variable `NXFLAG` is used to provide a reverse communication interface; the unperturbed solution will be solved once, and perturbed solutions will be solved numerous times. The indicators $\mathtt{I_t}, \mathtt{K_t}, \mathtt{L_t}$, and $\mathtt{E_t}$ are calculated by `BEDSEST`, which is called by `BEDSEMN` when the required number of samples has been obtained for the perturbation size t. A flag may be specified to indicate that the eigenvalue has multiplicity $m > 1$, in which case the Hölder indicators $\mathtt{K_t^h}$ and $\mathtt{E_t^h}$ for $h = 1/m$ are also calculated. The subroutine `BEDSEFO` is used to write the indicator values to a file.

Subroutines for both `DOUBLE PRECISION` and `DOUBLE COMPLEX` data are provided. Since unsymmetric matrices in $\mathbb{R}^{n \times n}$ may have complex eigenvalues, the subroutines for the two data types differ in form. Therefore, the specifications for both versions are provided.

These routines are not designed for parallel architectures, but may be modified to manipulate data which is distributed in a multi-processor environment; for an example of this, see the example implementation with the NAG Parallel Library in Section 3.4.

Subroutine	Description
`BEDSEMN`	Main driver to collect samples for `DOUBLE PRECISION` data.
`BEDSEST`	Calculate indicators.
`BEDSEFO`	Write out all indicator values to a file.
`BEZSEMN`	Main driver to collect samples for `DOUBLE COMPLEX` data.
`BEZSEST`	Calculate indicators.
`BEZSEFO`	Write out all indicator values to a file.

Table 3.6: Subroutines for standard eigenproblems. The names of double precision routines start with `BEDSExx`, while double complex routines are named `BEZSExx`.

```
     numptb  ← number of perturbation sizes
     numsamp ← number of sample sizes
     ptbs(i) ← perturbation sizes t

10   continue

     CALL BEDSEMN(..)

     if (NXFLAG.EQ.1) then

          ... solve unperturbed problem A x = λ x
              (or read λ and x from a file).

          ... y ← A x
              (or read y from a file).

          goto 10
     else if (NXFLAG.EQ.2) then

          ... solve a perturbed problem A x̃ = λ̃ x̃
              (or read λ̃ and x̃ from a file).

          ... ỹ ← A x̃
              (or read ỹ from a file).

          goto 10
     else if (NXFLAG.EQ.3) then

          CALL BEDSEFO(..)
          ... stop

     end if
```

Figure 3.6: General procedure for using the standard eigenproblem subroutines BEDSEMN and BEDSEFO. The routines BEZSEMN and BEZSEFO are used similarly.

BEDSEMN

Description

BEDSEMN is the main driver to collect samples and calculate indicators for standard eigenproblems $Ax = \lambda x$ with DOUBLE PRECISION matrices. The eigenvalue λ may be complex. This subroutine is called repeatedly, each time with a single solution of the perturbed or unperturbed linear system. The number of perturbation sizes and the number of samples for each perturbation size must be determined prior to the first call to BEDSEMN. In addition, all the perturbation sizes are assumed to be stored in an array.

The routine is first called to initialize internal values. The second call is used to store the solution to the unperturbed problem. Subsequent calls are used to store the sample solutions to the perturbed problems. As soon as the number of samples is reached for a fixed perturbation size, the indicators are calculated and the workspace is reinitialized to store samples for the next perturbation size. In this way, only storage for the number of samples for *one* perturbation size is needed. To aid the user, a reverse-communication flag NXFLAG is provided to ensure the iteration is performed correctly. See Figure 3.6.

For componentwise models, observe that the vector value E|x| is required to calculate the indicators, where x is the eigenvector computed for the unperturbed problem. The matrix E is determined by the perturbation model, a commonly used value is $E = |A|$. Thus, when the unperturbed problem is solved for a componentwise model, the user should subsequently calculate E|x| and the resulting vector value should then be placed in the first N components of the workspace array.

Specification

```
 SUBROUTINE BEDSEMN(PFLAG,KASCENT,N,NUMPTB,NUMSAMP,ALPHA,
+                   ER,EI,XR,XI,YR,YI,RWORK,LRW,
+                   KNTPTB,KNTSAMP,PTBS,STATS,NEWPSZ,NXFLAG,INFO)
 INTEGER PFLAG,KASCENT,N,NUMPTB,NUMSAMP,NEWPSZ,LRW,
+        KNTPTB,KNTSAMP,LRW,NEWPSZ,NXFLAG,INFO
 DOUBLE PRECISION ALPHA,ER,EI
 DOUBLE PRECISION PTBS(*),RWORK(*),STATS(6,*)
 DOUBLE PRECISION XR(*),XI(*),YR(*),YI(*)
```

Arguments

PFLAG - INTEGER (Input)
  Specifies what type of indicators are to be calculated. A value of 1 specifies normwise-type indicators are to be calculated, and a value of 2 specifies componentwise-type indicators. If PFLAG=2 is specified, then by default the componentwise perturbations use the values $E = |A|$.

KASCENT - INTEGER (Input)
  Specifies the ascent value of the eigenvalue. A value greater than 1 indicates that the eigenvalue sensitivity should be evaluated by the Hölder quantities $\mathtt{K}_\mathtt{t}^\mathtt{h}$ and $\mathtt{E}_\mathtt{t}^\mathtt{h}$, where $h = 1/$KASCENT. A value of 1 indicates the eigenvalue is simple, in which case the quantities $\mathtt{K}_\mathtt{t}^\mathtt{h}$ and $\mathtt{E}_\mathtt{t}^\mathtt{h}$ are not calculated.

`N` - `INTEGER` (Input)
Specifies the order of the matrix A.

`NUMPTB` - `INTEGER` (Input)
Specifies the number of perturbation values t which are held in `PTBS`.

`NUMSAMP` - `INTEGER` (Input)
Specifies the number of samples used to calculate the indicators for each perturbation size. Recommended values for NUMSAMP are between 5 to 15. Large numbers of samples will require both more computational effort and more workspace.

`ALPHA` - `DOUBLE PRECISION` (Input)
Specifies the scaling used to produce the perturbations. For normwise-type models, the perturbations of size t will satisfy $\|\Delta A\| \leq t\alpha$. `ALPHA` must be greater than zero.

`ER` - `DOUBLE PRECISION` (Input)
If `NXFLAG=1`, then this parameter should hold on input the real part of the eigenvalue λ obtained from solving the *unperturbed* problem. If `NXFLAG=2`, then this parameter should hold on input the real part of the eigenvalue $\tilde{\lambda}$ obtained from solving the *perturbed* problem.

`EI` - `DOUBLE PRECISION` (Input)
If `NXFLAG=1`, then this parameter should hold on input the imaginary part of the eigenvalue λ obtained from solving the *unperturbed* problem. If `NXFLAG=2`, then this parameter should hold on input the imaginary part of the eigenvalue $\tilde{\lambda}$ obtained from solving the *perturbed* problem.

`XR` - `DOUBLE PRECISION (*)` (Input/Workspace)
Array of length at least N. If `NXFLAG=1`, then this array should hold on input the real part of the unperturbed eigenvector x obtained from solving the *unperturbed* problem. If `NXFLAG=2`, then this array should hold on input the real part of the eigenvector $\tilde{x}$ obtained from solving the *perturbed* problem. When calculating indicators, this array will also be used as workspace.

`XI` - `DOUBLE PRECISION (*)` (Input/Workspace)
Array of length at least N. If `NXFLAG=1`, then this array should hold on input the imaginary part of the unperturbed eigenvector x obtained from solving the *unperturbed* problem. If `NXFLAG=2`, then this array should hold on input the imaginary part of the eigenvector $\tilde{x}$ obtained from solving the *perturbed* problem. When calculating indicators, this array will also be used as workspace.

`YR` - `DOUBLE PRECISION (*)` (Input/Workspace)
Array of length at least N. If `NXFLAG=1`, then this array should hold on input the result $y_r = Ax_r$, where x_r is `XR`, the real part of the eigenvector from solving the *unperturbed* solution. If `NXFLAG=2`, then this array should hold on input the result $y_r = A\tilde{x}_r$, where $\tilde{x}_r$ is `XR`, the real part of the eigenvector from solving the *perturbed* solution. When calculating indicators, this array will also be used as workspace.

`YI` - `DOUBLE PRECISION (*)` (Input/Workspace)
Array of length at least N. If `NXFLAG=1`, then this array should hold on input the result $y_i = Ax_i$, where x_i is `XI`, the real part of the eigenvector from solving the *unperturbed*

solution. If NXFLAG=2, then this array should hold on input the result $y_i = A\tilde{x}_i$, where $\tilde{x}_i$ is XI, the imaginary part of the eigenvector from solving the *perturbed* solution. When calculating indicators, this array will also be used as workspace.

RWORK - DOUBLE PRECISION (*) (Input/Workspace)
Workspace of length at least LRW for holding samples and producing indicators.

For componentwise computations (PFLAG=2), if the solution to the unperturbed problem has been requested (NXFLAG=1), then the value E|x| should be computed and placed in the first N locations of RWORK before calling the routine again. The value E = |A| is usually used for a componentwise analysis.

LRW - INTEGER (Input)
Length of the workspace array RWORK. It must have a value at least 5+2*N*(NUMSAMP+1).

KNTPTB - INTEGER (Output)
For informational purposes, this parameter indicates which perturbation value in PTBS is currently being processed. This value must remain unchanged between subsequent calls.

KNTSAMP - INTEGER (Output)
For informational purposes, this parameter indicates which sample is needed for the current perturbation size. This value must remain unchanged between subsequent calls.

PTBS - DOUBLE PRECISION (*) (Input)
Contains the perturbation sizes used to produce the perturbed samples. The array size must be at least NUMPTB.

STATS - DOUBLE PRECISION (6,*) (Output)
When NXFLAG=3, this array holds the indicator values for the corresponding perturbation sizes in PTBS. The values $\mathtt{I_t}, \mathtt{K_t}, \mathtt{L_t}$ and $\mathtt{E_t}$ are held respectively in PTBS(1,*), PTBS(2,*), PTBS(3,*) and PTBS(4,*). If KASCENT > 1, then $\mathtt{K_t^h}$ and $\mathtt{E_t^h}$ are held in the respective positions PTBS(5,*) and PTBS(6,*). The subroutine BEDSEFO may be used to write these values and related information to a file.

NEWPSZ - INTEGER (Output)
For informational purposes, when NXFLAG=2 this parameter indicates that the first sample for a perturbation value is requested. That is, KNTPTB has been incremented and KNTSAMP has been reset to 1. This may be used to monitor the progress of the calculation.

NXFLAG - INTEGER (Input/Output)
Reverse communication flag. On initial entry, this parameter must have a value of zero. On subsequent entries, this parameter should remain unchanged from the previous call.

On exit, a value of 1 specifies that the solution to the unperturbed problem is to be placed in parameters ER,EI,XR,XI, and its image is to be placed in the parameters in YR,YI. Then the subroutine should be called again.

On exit, a value of 2 specifies that the solution to the perturbed problem is to be placed in ER,EI,XR,XI, and its image is to be placed in YR,YI. Note that to obtain the image values `YR,YI`, the product of `XR,XI` and the *unperturbed* matrix A should be used. Then the subroutine should be called again.

Conceivably, since the number of perturbations and the number of samples are known in advance, fixed `FOR`-loops may be used instead to perform the iteration. This option is available to the user, but the indicator flags should be used to monitor the progress of the iteration.

`INFO` - `INTEGER` (Input/Output)
Error flag. On initial entry, this value must be zero. On exit, a value of zero indicates no errors have occurred. A value other than zero indicates an error; an error message will be printed by the subroutine, and the iteration should be stopped.

BEDSEST

Description

BEDSEST is used to processes the sample data and calculate the indicators $\mathtt{I_t}, \mathtt{K_t}, \mathtt{L_t}$ and $\mathtt{E_t}$ for standard eigenvalue problems $Ax = \lambda x$ in DOUBLE PRECISION arithmetic. The Hölder indicators $\mathtt{K_t^h}$ and $\mathtt{E_t^h}$ are also calculated in the case of a multiple eigenvalue of order $m = 1/h$. This subroutine is called once for each perturbation value t. This subroutine is called directly from BEDSEMN; the specification is provided here for those developing analysis routines for other types of numerical problems. See Section 3.2.2 for a discussion of how the indicators are calculated.

Specification

```
      SUBROUTINE BEDSEST(PFLAG,KASCENT,N,NUMSAMP,PTBVAL,KNTPTB,
     +                   ER,EI,XR,XI,YR,YI,ALPHA,XCALNORM,RESNORM,
     +                   AREF,XSAMPS,YSAMPS,REFS,STATS)
      INTEGER PFLAG,KASCENT,N,NUMSAMP,KNTPTB
      DOUBLE PRECISION PTBVAL,ER,EI,ALPHA,XCALNORM,RESNORM,AREF
      DOUBLE PRECISION XR(*),XI(*),YR(*),YI(*)
      DOUBLE PRECISION XSAMPS(*),YSAMPS(*),REFS(*),STATS(6,*)
```

Arguments

PFLAG - INTEGER (Input)
: Specifies what type of indicators are to be calculated. A value of 1 specifies normwise-type indicators are to be calculated, and a value of 2 specifies componentwise-type indicators. If PFLAG=2 is specified, then by default the componentwise perturbations use the values $E = |A|$.

KASCENT - INTEGER (Input)
: Specifies the ascent value of the eigenvalue. A value greater than 1 indicates that the eigenvalue sensitivity should be evaluated by the Hölder quantities $\mathtt{K_t^h}$ and $\mathtt{E_t^h}$, where $h = 1/$KASCENT. A value of 1 indicates the eigenvalue is simple, in which case the quantities $\mathtt{K_t^h}$ and $\mathtt{E_t^h}$ are not calculated.

N - INTEGER (Input)
: Specifies the order of the matrix A.

NUMSAMP - INTEGER (Input)
: Specifies the number of samples held in XSAMPS and YSAMPS.

PTBVAL - DOUBLE PRECISION (Input)
: Specifies the perturbation value t which was used to produce the samples.

KNTPTB - INTEGER (Input)
: Specifies which perturbation value in PTBS is currently being processed. The indicator values will be stored in the corresponding locations of the STATS array.

ER - DOUBLE PRECISION (Input)
: The real part of the eigenvalue λ obtained from solving the *unperturbed* problem.

`EI` - `DOUBLE PRECISION` (Input)
The imaginary part of the eigenvalue λ obtained from solving the *unperturbed* problem.

`XR` - `DOUBLE PRECISION (*)` (Workspace)
Array of length at least N.

`XI` - `DOUBLE PRECISION (*)` (Workspace)
Array of length at least N.

`YR` - `DOUBLE PRECISION (*)` (Workspace)
Array of length at least N.

`YI` - `DOUBLE PRECISION (*)` (Workspace)
Array of length at least N.

`ALPHA` - `DOUBLE PRECISION` (Input)
Specifies the scaling used to produce the perturbations.

`XCALNORM` - `DOUBLE PRECISION` (Input)
Modulus of the computed eigenvalue, i.e. $|\lambda|$.

`RESNORM` - `DOUBLE PRECISION` (Input)
Norm of the residual, i.e. $\|Ax - \lambda x\|$.

`AREF` - `DOUBLE PRECISION` (Input)
For normwise indicators, this value should hold $\alpha \times \|x\|$, where x is the eigenvector obtained from the *unperturbed* solution. For componentwise indicators, this parameter should have value 1.

`XSAMPS` - `DOUBLE PRECISION (*)` (Input/Workspace)
Holds the values of the sample solutions $\tilde{\lambda}$ obtained from solving the perturbed problems with perturbation size t. The real and imaginary parts of $\tilde{\lambda}$ are held in consecutive locations, with the real part first. The array is also used as workspace during the calculation of the indicators.

`YSAMPS` - `DOUBLE PRECISION (*)` (Input/Workspace)
Holds the values of the samples $A\tilde{x} - \tilde{\lambda}\tilde{x}$ obtained from solving the perturbed problems with perturbation size t. The real and imaginary parts are held in consecutive locations, with the real part first. The array is also used as workspace during the calculation of the indicators.

`REFS` - `DOUBLE PRECISION (*)` (Input)
Holds the reciprical values of the scaling factors $E|x|$. These values are only referenced for componentwise indicators (`PFLAG=2`).

`STATS` - `DOUBLE PRECISION (6,*)` (Output)
Holds the computed indicator values. The values $\mathtt{I_t}, \mathtt{K_t}, \mathtt{L_t}$ and $\mathtt{E_t}$ are stored respectively in `PTBS(1,KNTPTB)`, `PTBS(2,KNTPTB)`, `PTBS(3,KNTPTB)` and `PTBS(4,KNTPTB)`. If `KASCENT` > 1, then $\mathtt{K_t^h}$ and $\mathtt{E_t^h}$ are stored in the respective positions `PTBS(5,*)` and `PTBS(6,*)`.

BEDSEFO

Description

BEDSEFO is used to write out to a file all information obtained from a backward analysis in DOUBLE PRECISION. In addition to the indicator values $\mathtt{I_t}, \mathtt{K_t}, \mathtt{L_t}, \mathtt{E_t}, \mathtt{K_t^h}$, and $\mathtt{E_t^h}$, some values useful for the interpretation of results are also written.

The format of the output file is:

```
PTYPE
ALPHA
KASCENT
XCALNORM
RESNORM
ERRNORM
NUMPTB
```

t_1	I_1	L_1	K_1	E_1	K_1^h	E_1^h
t_2	I_2	L_2	K_2	E_2	K_2^h	E_2^h
$\vdots$						
$t_{\mathtt{NUMPTB}}$	$I_{\mathtt{NUMPTB}}$	$L_{\mathtt{NUMPTB}}$	$K_{\mathtt{NUMPTB}}$	$E_{\mathtt{NUMPTB}}$	$K^h_{\mathtt{NUMPTB}}$	$E^h_{\mathtt{NUMPTB}}$

The values t_i are the perturbation values used to produce the samples, while the values $\mathtt{I_i}, \mathtt{L_i}, \mathtt{K_i}, \mathtt{E_i}, \mathtt{K_i^h}$ and $\mathtt{E_i^h}$ are the corresponding indicator values. Note that the order of the perturbation values is not specified, so t_1 may or may not be greater than t_{NUMPTB}; the visualisation tool should ensure that the plots are generated in the correct order. See Appendix B for an example of a Matlab source file which can be used to generate plots from the output file.

Specification

```
      SUBROUTINE BEDSEFO(OFILE,PTYPE,NUMPTB,PTBS,STATS,ALPHA,KASCENT,
     &                   RWORK,ERRNORM)
      CHARACTER*72 OFILE
      INTEGER PTYPE,NUMPTB,KASCENT
      DOUBLE PRECISION ALPHA,ERRNORM
      DOUBLE PRECISION RWORK(*),PTBS(*),STATS(6,*)
```

Arguments

OFILE - CHARACTER*72 (Input)
The name of the output file to store the results.

PTYPE - INTEGER (Input)
Specifies what type of indicators were calculated. A value of 1 specifies normwise-type indicators, and a value of 2 specifies componentwise-type indicators.

`NUMPTB` - `INTEGER` (Input)
Specifies the number of perturbation values t which are held in `PTBS`.

`PTBS` - `DOUBLE PRECISION (*)` (Input)
Contains the perturbation sizes used to produce the perturbed samples. The array size must be at least `NUMPTB`.

`STATS` - `DOUBLE PRECISION (6,*)` (Input)
This array holds the indicator values for the corresponding perturbation sizes in `PTBS`. The values $\mathtt{I_t}, \mathtt{K_t}, \mathtt{L_t}, \mathtt{E_t}, \mathtt{K_i^h}$ and $\mathtt{E_i^h}$ are held respectively in `PTBS(1,*)`, `PTBS(2,*)`, `PTBS(3,*)`, `PTBS(4,*)`, `PTBS(5,*)`, and `PTBS(6,*)`.

`ALPHA` - `DOUBLE PRECISION` (Input)
Specifies the scaling used to produce the perturbations.

`KASCENT` - `INTEGER` (Input)
Specifies the ascent value of the eigenvalue.

`RWORK` - `DOUBLE PRECISION (*)` (Input)
Array holding the values of `RESNORM` and `XCALNORM`. The value in `RWORK(1)` must hold an integer value giving the locations of the two values. If `loc = int(RWORK(1)`, then the value of `RESNORM` is found at `RWORK(loc)`, and the value of `XCALNORM` is found at `RWORK(loc+4)`.

`ERRNORM` - `DOUBLE PRECISION` (Input)
Norm of the error between the known true solution λ_k and the unperturbed solution λ_c obtained by the numerical procedure. This value can be calculated by the user if the true solution λ_k to the analytical problem is known; it provides a means to compare the actual forward error with the forward error bound $\mathtt{E_t}$ determined by PRECISE. If λ_k is not known, set this value to zero. if λ_k is known set this value to $|\lambda_k - \lambda_c|$.

BEZSEMN

Description

BEZSEMN is the main driver to collect samples and calculate indicators for standard eigenproblems $Ax = \lambda x$ with DOUBLE COMPLEX matrices. This subroutine is called repeatedly, each time with a single solution of the perturbed or unperturbed linear system. The number of perturbation sizes and the number of samples for each perturbation size must be determined prior to the first call to BEZSEMN. In addition, all the perturbation sizes are assumed to be stored in an array.

The routine is first called to initialize internal values. The second call is used to store the solution to the unperturbed problem. Subsequent calls are used to store the sample solutions to the perturbed problems. As soon as the number of samples is reached for a fixed perturbation size, the indicators are calculated and the workspace is reinitialized to store samples for the next perturbation size. In this way, only storage for the number of samples for *one* perturbation size is needed. To aid the user, a reverse-communication flag NXFLAG is provided to ensure the iteration is performed correctly. See Figure 3.6.

For componentwise models, observe that the vector value E|x| is required to calculate the indicators, where x is the eigenvector computed for the unperturbed problem. The matrix E is determined by the perturbation model, a commonly used value is $E = |A|$. Thus, when the unperturbed problem is solved for a componentwise model, the user should subsequently calculate E|x| and the resulting vector value should then be placed in the first N components of the workspace array.

Specification

```
 SUBROUTINE BEZSEMN(PFLAG,KASCENT,N,NUMPTB,NUMSAMP,ALPHA,
+                   EIG,X,Y,CWORK,LRW,KNTPTB,KNTSAMP,
+                   PTBS,STATS,NEWPSZ,NXFLAG,INFO)
 INTEGER PFLAG,KASCENT,N,NUMPTB,NUMSAMP,NEWPSZ,LRW,
+        KNTPTB,KNTSAMP,LRW,NEWPSZ,NXFLAG,INFO
 DOUBLE PRECISION ALPHA
 DOUBLE PRECISION STATS(6,*)
 DOUBLE COMPLEX   EIG
 DOUBLE COMPLEX   PTBS(*),X(*),Y(*),CWORK(*)
```

Arguments

PFLAG - INTEGER (Input)

Specifies what type of indicators are to be calculated. A value of 1 specifies normwise-type indicators are to be calculated, and a value of 2 specifies componentwise-type indicators. If PFLAG=2 is specified, then by default the componentwise perturbations use the values $E = |A|$.

KASCENT - INTEGER (Input)

Specifies the ascent value of the eigenvalue. A value greater than 1 indicates that the eigenvalue sensitivity should be evaluated by the Hölder quantities K_t^h and E_t^h, where $h = 1/$KASCENT. A value of 1 indicates the eigenvalue is simple, in which case the quantities K_t^h and E_t^h are not calculated.

`N` - `INTEGER` (Input)
Specifies the order of the matrix A.

`NUMPTB` - `INTEGER` (Input)
Specifies the number of perturbation values t which are held in `PTBS`.

`NUMSAMP` - `INTEGER` (Input)
Specifies the number of samples used to calculate the indicators for each perturbation size. Recommended values for NUMSAMP are between 5 to 15. Large numbers of samples will require both more computational effort and more workspace.

`ALPHA` - `DOUBLE PRECISION` (Input)
Specifies the scaling used to produce the perturbations. For normwise-type models, the perturbations of size t will satisfy $\|\Delta A\| \leq t\alpha$. `ALPHA` must be greater than zero.

`EIG` - `DOUBLE COMPLEX` (Input)
If `NXFLAG=1`, then this parameter should hold on input the the eigenvalue λ obtained from solving the *unperturbed* problem. If `NXFLAG=2`, then this parameter should hold on input the eigenvalue $\tilde{\lambda}$ obtained from solving the *perturbed* problem.

`X` - `DOUBLE COMPLEX (*)` (Input/Workspace)
Array of length at least N. If `NXFLAG=1`, then this array should hold on input the unperturbed eigenvector x obtained from solving the *unperturbed* problem. If `NXFLAG=2`, then this array should hold on input the the eigenvector $\tilde{x}$ obtained from solving the *perturbed* problem. When calculating indicators, this array will also be used as workspace.

`Y` - `DOUBLE COMPLEX (*)` (Input/Workspace)
Array of length at least N. If `NXFLAG=1`, then this array should hold on input the result $y = Ax$, where x is `X`, the eigenvector from solving the *unperturbed* solution. If `NXFLAG=2`, then this array should hold on input the result $y = A\tilde{x}$, where $\tilde{x}$ is `X`, the eigenvector from solving the *perturbed* solution. When calculating indicators, this array will also be used as workspace.

`CWORK` - `DOUBLE COMPLEX (*)` (Input/Workspace)
Workspace of length at least `LRW` for holding samples and producing indicators.

For componentwise computations (`PFLAG=2`), if the solution to the unperturbed problem has been requested (`NXFLAG=1`), then the value `E|x|` should be computed and placed in the first `N` locations of `CWORK` before calling the routine again. The value `E` = `|A|` is usually used for a componentwise analysis.

`LRW` - `INTEGER` (Input)
Length of the workspace array `CWORK`. It must have a value at least `2*N + N*NUMSAMP)`.

`KNTPTB` - `INTEGER` (Output)
For informational purposes, this parameter indicates which perturbation value in `PTBS` is currently being processed. This value must remain unchanged between subsequent calls.

`KNTSAMP` - `INTEGER` (Output)
For informational purposes, this parameter indicates which sample is needed for the

current perturbation size. This value must remain unchanged between subsequent calls.

`PTBS` - `DOUBLE COMPLEX (*)` (Input)
Contains the perturbation sizes used to produce the perturbed samples. The array size must be at least `NUMPTB`.

`STATS` - `DOUBLE PRECISION (6,*)` (Output)
When `NXFLAG=3`, this array holds the indicator values for the corresponding perturbation sizes in `PTBS`. The values $\mathtt{I_t}, \mathtt{K_t}, \mathtt{L_t}$ and $\mathtt{E_t}$ are held respectively in `PTBS(1,*)`, `PTBS(2,*)`, `PTBS(3,*)` and `PTBS(4,*)`. If `KASCENT` $>$ `1`, then $\mathtt{K_t^h}$ and $\mathtt{E_t^h}$ are held in the respective positions `PTBS(5,*)` and `PTBS(6,*)`. The subroutine `BEZSEFO` may be used to write these values and related information to a file.

`NEWPSZ` - `INTEGER` (Output)
For informational purposes, when `NXFLAG=2` this parameter indicates that the first sample for a perturbation value is requested. That is, `KNTPTB` has been incremented and `KNTSAMP` has been reset to 1. This may be used to monitor the progress of the calculation.

`NXFLAG` - `INTEGER` (Input/Output)
Reverse communication flag. On initial entry, this parameter must have a value of zero. On subsequent entries, this parameter should remain unchanged from the previous call.

On exit, a value of 1 specifies that the solution to the unperturbed problem is to be placed in parameters `ER,EI,XR,XI`, and its image is to be placed in the parameters in `YR,YI`. Then the subroutine should be called again.

On exit, a value of 2 specifies that the solution to the perturbed problem is to be placed in `ER,EI,XR,XI`, and its image is to be placed in `YR,YI`. Note that to obtain the image values `YR,YI`, the product of `XR,XI` and the *unperturbed* matrix A should be used. Then the subroutine should be called again.

Conceivably, since the number of perturbations and the number of samples are known in advance, fixed `FOR`-loops may be used instead to perform the iteration. This option is available to the user, but the indicator flags should be used to monitor the progress of the iteration.

`INFO` - `INTEGER` (Input/Output)
Error flag. On initial entry, this value must be zero. On exit, a value of zero indicates no errors have occurred. A value other than zero indicates an error; an error message will be printed by the subroutine, and the iteration should be stopped.

BEZSEST

Description

BEZSEST is used to processes the sample data and calculate the indicators $\mathtt{I_t}, \mathtt{K_t}, \mathtt{L_t}$ and $\mathtt{E_t}$ for standard eigenvalue problems $Ax = \lambda x$ in DOUBLE COMPLEX arithmetic. The Hölder indicators $\mathtt{K_t^h}$ and $\mathtt{E_t^h}$ are also calculated in the case of a multiple eigenvalue of order $m = 1/h$. This subroutine is called once for each perturbation value t. This subroutine is called directly from BEZSEMN; the specification is provided here for those developing analysis routines for other types of numerical problems. See Section 3.2.2 for a discussion of how the indicators are calculated.

Specification

```
      SUBROUTINE BEZSEST(PFLAG,KASCENT,N,NUMSAMP,CPTBVAL,KNTPTB,
     +                   EIG,X,Y,ALPHA,XCALNORM,RESNORM,AREF,
     +                   XSAMPS,YSAMPS,REFS,STATS)
      INTEGER PFLAG,KASCENT,N,NUMSAMP,KNTPTB
      DOUBLE PRECISION ALPHA,XCALNORM,RESNORM,AREF
      DOUBLE PRECISION STATS(6,*)
      DOUBLE COMPLEX   CPTBVAL,EIG
      DOUBLE COMPLEX   X(*),Y(*),XSAMPS(*),YSAMPS(*),REFS(*)
```

Arguments

PFLAG - INTEGER (Input)
: Specifies what type of indicators are to be calculated. A value of 1 specifies normwise-type indicators are to be calculated, and a value of 2 specifies componentwise-type indicators. If PFLAG=2 is specified, then by default the componentwise perturbations use the values $E = |A|$.

KASCENT - INTEGER (Input)
: Specifies the ascent value of the eigenvalue. A value greater than 1 indicates that the eigenvalue sensitivity should be evaluated by the Hölder quantities $\mathtt{K_t^h}$ and $\mathtt{E_t^h}$, where $h = 1/\mathtt{KASCENT}$. A value of 1 indicates the eigenvalue is simple, in which case the quantities $\mathtt{K_t^h}$ and $\mathtt{E_t^h}$ are not calculated.

N - INTEGER (Input)
: Specifies the order of the matrix A.

NUMSAMP - INTEGER (Input)
: Specifies the number of samples held in XSAMPS and YSAMPS.

CPTBVAL - DOUBLE COMPLEX (Input)
: Specifies the perturbation value t which was used to produce the samples.

KNTPTB - INTEGER (Input)
: Specifies which perturbation value in PTBS is currently being processed. The indicator values will be stored in the corresponding locations of the STATS array.

EIG - DOUBLE COMPLEX (Input)
The eigenvalue λ obtained from solving the *unperturbed* problem.

X - DOUBLE COMPLEX (*) (Workspace)
Array of length at least N.

Y - DOUBLE COMPLEX (*) (Workspace)
Array of length at least N.

ALPHA - DOUBLE PRECISION (Input)
Specifies the scaling used to produce the perturbations.

XCALNORM - DOUBLE PRECISION (Input)
Modulus of the computed eigenvalue, i.e. $|\lambda|$.

RESNORM - DOUBLE PRECISION (Input)
Norm of the residual, i.e. $\|Ax - \lambda x\|$.

AREF - DOUBLE PRECISION (Input)
For normwise indicators, this value should hold $\alpha \times \|x\|$, where x is the eigenvector obtained from the *unperturbed* solution. For componentwise indicators, this parameter should have value 1.

XSAMPS - DOUBLE COMPLEX (*) (Input/Workspace)
Holds the values of the sample solutions $\tilde{\lambda}$ obtained from solving the perturbed problems with perturbation size t. The array is also used as workspace during the calculation of the indicators.

YSAMPS - DOUBLE COMPLEX (*) (Input/Workspace)
Holds the values of the samples $A\tilde{x} - \tilde{\lambda}\tilde{x}$ obtained from solving the perturbed problems with perturbation size t. The array is also used as workspace during the calculation of the indicators.

REFS - DOUBLE COMPLEX (*) (Input)
Holds the reciprical values of the scaling factors $E|x|$. These values are only referenced for componentwise indicators (PFLAG=2).

STATS - DOUBLE PRECISION (6,*) (Output)
Holds the computed indicator values. The values $\mathtt{I_t}, \mathtt{K_t}, \mathtt{L_t}$ and $\mathtt{E_t}$ are stored respectively in PTBS(1,KNTPTB), PTBS(2,KNTPTB), PTBS(3,KNTPTB) and PTBS(4,KNTPTB). If KASCENT $>$ 1, then $\mathtt{K_t^h}$ and $\mathtt{E_t^h}$ are stored in the respective positions PTBS(5,*) and PTBS(6,*).

BEZSEFO

Description

BEZSEFO is used to write out to a file all information obtained from a backward analysis in DOUBLE COMPLEX arithmetic. In addition to the indicator values $\mathtt{I_t}, \mathtt{K_t}, \mathtt{L_t}, \mathtt{E_t}, \mathtt{K_t^h}$, and $\mathtt{E_t^h}$, some values useful for the interpretation of results are also written.

The format of the output file is:

```
PTYPE
ALPHA
KASCENT
XCALNORM
RESNORM
ERRNORM
NUMPTB
```

$\lvert t_1\rvert$	I_1	L_1	K_1	E_1	K_1^h	E_1^h
$\lvert t_2\rvert$	I_2	L_2	K_2	E_2	K_2^h	E_2^h
$\vdots$						
$\lvert t_{\mathtt{NUMPTB}}\rvert$	$I_{\mathtt{NUMPTB}}$	$L_{\mathtt{NUMPTB}}$	$K_{\mathtt{NUMPTB}}$	$E_{\mathtt{NUMPTB}}$	$K^h_{\mathtt{NUMPTB}}$	$E^h_{\mathtt{NUMPTB}}$

The values t_i are the perturbation values used to produce the samples, while the values $\mathtt{I_i}, \mathtt{L_i}, \mathtt{K_i}, \mathtt{E_i}, \mathtt{K_i^h}$ and $\mathtt{E_i^h}$ are the corresponding indicator values. The modulus of the perturbation values $|t_i|$ are written to the file. Note that the order of the perturbation values is not specified, so $|t_1|$ may or may not be greater than $|t_{\mathrm{NUMPTB}}|$; the visualisation tool should ensure that the plots are generated in the correct order. See Appendix B for an example of a Matlab source file which can be used to generate plots from the output file.

Specification

```
      SUBROUTINE BEDSEFO(OFILE,PTYPE,NUMPTB,PTBS,STATS,ALPHA,KASCENT,
     &                   CWORK,ERRNORM)
      CHARACTER*72 OFILE
      INTEGER PTYPE,NUMPTB,KASCENT
      DOUBLE PRECISION ALPHA,ERRNORM
      DOUBLE PRECISION STATS(6,*)
      DOUBLE COMPLEX   CWORK(*),PTBS(*)
```

Arguments

OFILE - CHARACTER*72 (Input)
The name of the output file to store the results.

PTYPE - INTEGER (Input)
Specifies what type of indicators were calculated. A value of 1 specifies normwise-type indicators, and a value of 2 specifies componentwise-type indicators.

`NUMPTB` - `INTEGER` (Input)
Specifies the number of perturbation values t which are held in `PTBS`.

`PTBS` - `DOUBLE COMPLEX (*)` (Input)
Contains the perturbation sizes t used to produce the perturbed samples. The array size must be at least `NUMPTB`. The modulus of the perturbation values $|t|$ will be written to the file.

`STATS` - `DOUBLE PRECISION (6,*)` (Input)
This array holds the indicator values for the corresponding perturbation sizes in `PTBS`. The values $\mathtt{I_t}, \mathtt{K_t}, \mathtt{L_t}, \mathtt{E_t}, \mathtt{K_i^h}$ and $\mathtt{E_i^h}$ are held respectively in `PTBS(1,*)`, `PTBS(2,*)`, `PTBS(3,*)`, `PTBS(4,*)`, `PTBS(5,*)`, and `PTBS(6,*)`.

`ALPHA` - `DOUBLE PRECISION` (Input)
Specifies the scaling used to produce the perturbations.

`KASCENT` - `INTEGER` (Input)
Specifies the ascent value of the eigenvalue.

`CWORK` - `DOUBLE COMPLEX (*)` (Input)
Array holding the values of `RESNORM` and `XCALNORM`. The value in real part of `CWORK(1)` must hold an integer value giving the locations of the two values. If `loc = int(CWORK(1)`, then the value of `RESNORM` is found at `CWORK(loc)`, and the value of `XCALNORM` is found at `CWORK(loc+3)`.

`ERRNORM` - `DOUBLE PRECISION` (Input)
Norm of the error between the known true solution λ_k and the unperturbed solution λ_c obtained by the numerical procedure. This value can be calculated by the user if the true solution λ_k to the analytical problem is known; it provides a means to compare the actual forward error with the forward error bound $\mathtt{E_t}$ determined by PRECISE. If λ_k is not known, set this value to zero. if λ_k is known set this value to $|\lambda_k - \lambda_c|$.

3.4 Implementations with Software Libraries

Section 3.3 describes routines which are provided to assist the user in performing a backward error analysis with any appropriate numerical solver. In this section, examples of completed implementations to perform a backward error analysis are described. The implementations utilise numerical solvers from well-known software packages, including LAPACK and the NAG Parallel Library.

The example drivers are listed in Table 3.7 for linear system solvers, and in Table 3.8 for eigensolvers. Common to all the examples is the methodology outlined in Figure 3.4; the sample data X_t and Y_t is generated in an iterative procedure for a particular perturbation model. Some of the example drivers use the subroutines described in Section 3.3. However, in some cases the subroutines have been modified to suit the numerical solver. A brief discussion regarding the implementations is provided in the following sections. The user should consult the source code of the example drivers for further details.

3.4.1 Direct Sequential Solvers

From an software development point of view, direct sequential solvers are conveniently implemented for a backward error analysis. The example implementations using solvers from LAPACK are in this category. These drivers use the subroutines from Section 3.3 without modification; the interface to these solvers is simple enough so that the samples can be processed while the perturbed solutions are generated.

3.4.2 Direct Parallel Solvers

The direct solvers from the NAG Parallel Library and SCALAPACK are in this category. For these drivers, some modifications were made to the subroutines so that the error analysis procedure could be performed directly. First, the perturbation subroutines were modified so that they could perform perturbations on data which is distributed across multiple processors. The subroutines `BEDNPTB` and `BEDCPTB` were changed to utilise information regarding the data distribution. In addition, for the NAG Parallel Library, the more sophisticated random number generator was utilised, with care taken to ensure the random number sequences on each processor were independent.

Modifications were also made to the subroutines which collect the samples and calculate the indicators. Since the vector data is distributed, the routine `BEDSYMN` was modified to collect the samples while still maintaining the data distribution. Also, the routine to calculate the indicators `BEDSYSE` was modified to process the distributed data samples. This last modification included the addition of communication calls using BLACS subroutines.

The example drivers of this section demonstrate that the procedure of generating the perturbation samples and calculating the indicators can be performed simultaneously in a parallel environment. The backward analysis process is performed in one program, the same way as with the sequential drivers. This involved some additional software development, but resulted in an easy to use process for performing a backward error analysis. An optional technique would be to write the solutions to a file as they are generated, and process the data in a subsequent step; this technique is used for the case of iterative solvers.

Example Routine	Software Library	Description
`exbedsyrd`	-	Calculates indicators taking $\mathtt{X_t}$ samples from a file while generating the $\mathtt{Y_t}$ samples in double precision.
`exbezsyrd`	-	Calculates indicators taking $\mathtt{X_t}$ samples from a file while generating the $\mathtt{Y_t}$ samples in double complex precision.
`exdgesv`	LAPACK	Direct solver using LU factorisation with partial pivoting for real general matrices.
`exdsysv`	LAPACK	Direct solver using factorisation for real symmetric indefinite matrices.
`exdgtsv`	LAPACK	Direct solver for real general tridiagonal matrices.
`exdgttr`	LAPACK	Direct solver for real general tridiagonal matrices.
`exzgesv`	LAPACK	Direct solver using LU factorisation with partial pivoting for complex general matrices.
`exzhesv`	LAPACK	Direct solver using factorisation for complex Hermitian indefinite matrices.
`exzsysv`	LAPACK	Direct solver using factorisation for complex symmetric indefinite matrices.
`exzgtsv`	LAPACK	Direct solver for complex general tridiagonal matrices.
`exf04eb`	NAG Parallel	Black Box solver for real general simultaneous linear equations.
`exf04ec`	NAG Parallel	Black Box solver for complex general simultaneous linear equations.
`exf04fb`	NAG Parallel	Black Box solver for real symmetric positive-definite simultaneous linear equations.
`exf04fc`	NAG Parallel	Black Box solver for complex Hermitian positive-definite simultaneous linear equations.
`exf07ae`	NAG Parallel	Direct solver using LU factorisation with partial pivoting for real general matrices.
`exf07as`	NAG Parallel	Direct solver using LU factorisation with partial pivoting for complex general matrices.
`exf07fe`	NAG Parallel	Direct solver using Cholesky factorisation for real symmetric matrices.
`exf07fs`	NAG Parallel	Direct solver using Cholesky factorisation for complex Hermitian matrices.
`exf11b`	NAG Parallel	Iterative solver for real general simultaneous linear equations with sparse matrices using RGMRES.
`exf11d`	NAG Parallel	Black Box iterative solver for real general simultaneous linear equations with sparse matrices preconditioned with incomplete LU factorisation.
`exf11g`	NAG Parallel	Iterative solver for real symmetric sparse matrices using the Conjugate Gradient method or SYMMLQ.
`expdgesv`	SCALAPACK	Direct solver using LU factorisation with partial pivoting for real general matrices.
`expimcg`	PIM	Iterative solver for real, symmetric sparse matrices using the Conjugate Gradient method.

Table 3.7: Example backward error analysis routines for linear system solvers.

Example Routine	Software Library	Description
exbedserd	-	Calculates indicators taking the eigenvalues and eigenvectors from a file while computing the residuals in double precision.
exbezserd	-	Calculates indicators taking the eigenvalues and eigenvectors from a file while computing the residuals in double complex precision.
exdgeev	LAPACK	Direct solver using Hessenberg reduction and Schur factorisation for real unsymmetric matrices.
exdstev	LAPACK	Direct solver for real tridiagonal matrices.
exdsyev	LAPACK	Direct solver using tridiagonalization for real symmetric matrices.
exzgeev	LAPACK	Direct solver using Hessenberg reduction and Schur factorisation for complex non-Hermitian matrices.
exzheev	LAPACK	Direct solver using tridiagonalization for complex Hermitian matrices.
exf02fq	NAG Parallel	Solver using the one-sided Jacobi method for real symmetric matrices.
exf02fr	NAG Parallel	Solver using the one-sided Jacobi method for complex Hermitian matrices.

Table 3.8: Example backward error analysis routines for eigensolvers.

3.4.3 Iterative Solvers

The example drivers utilising iterative solvers from the NAG Parallel library and PIM use a two-step technique to perform the backward error analysis. The first step is to produce the perturbed solutions, saving the results in a temporary file. The example drivers for the iterative solvers are designed to perform this task. The second step is to process the sample data and calculate the indicator values; this can be done using one of the utilities `exbedsyrd` or `exbezsyrd` for linear systems, or `exbedserd` or `exbezserd` for eigenproblems.

This two-step process offers an easy implementation path for both sequential and parallel solvers. Since any solver usually has a facility to write the solution to a file, this facility may be utilised to provide sample data for the analysis. The utilities `exbedsyrd, exbezsyrd, exbedserd` or `exbezserd` may need to be slightly modified to suit the matrix format (dense, sparse, etc.), see the comments within the source code for details.

Chapter 4

Module 2 - Spectral Sensitivity Analysis

Eigenvalues of non-symmetric matrices play an essential role in physical problems. These eigenvalues may be used for their physical meaning (vibration modes, stability region, ...) or for their numerical meaning (convergence of numerical schemes, ...). However, as a matrix gets far from normal[1], its spectrum becomes more and more unstable. For such matrices, iterative methods can become backward unstable (if not implemented carefully), see [4, 5]; and even stable algorithms such as QR give inaccurate approximations for eigenvalues due to the huge spectral ill-conditioning, see [5, 7]. Moreover, there is still no clear method to determine a condition number for clustered eigenvalues, only upper bounds exist in the literature, see [6, 7]. In this context, a computed eigenvalue may loose its meaning if no information is known about its stability. The trend today is to consider an eigenvalue not by itself but also its topological neighbourhood. This approach leads to understanding of the influence of perturbations on the spectrum of nonnormal matrices, see [19].
Module 2 of PRECISE provides tools of choice for understanding the influence of perturbations on eigenvalues, namely *Perturbed Spectra* plots (denoted by **PTS**) and *Spectral Portraits* (denoted by **SPP**).

The **PTS** tool is useful when the scientist wants to study the effect of a **given type of perturbation** on the spectrum of a **real** matrix. Moreover, it can help in the determination of the **Jordan form** of a matrix.

On the other hand, the use of the **SPP** tool is not restricted to real matrices, it can also be applied to matrix pencils. Spectral Portraits display, for **complex normwise perturbations**, the set of complex points that can, for a given size of perturbation, **behave as an eigenvalue** during computation (eigenproblem, iterative method,...).

These tools are both based on the concepts of *pseudoeigenvalue* and *pseudospectrum* that are presented in the next section.

4.1 Perturbations and eigenvalue problems

In this section, the concepts of pseudoeigenvalue and pseudospectrum for matrices and for matrix pencils are defined.

Let A be any $n \times n$ complex or real matrix. By definition z is an ε-pseudoeigenvalue of A if z is an eigenvalue of $A + \Delta A$ with ΔA satisfying :

[1] A matrix A is said to be normal if $A^*A = AA^*$, otherwise it is said to be nonnormal.

- $\|\Delta A\| \leq \varepsilon \|A\|$, if the perturbations under consideration are normwise type,
- $|\Delta A| \leq \varepsilon |A|$, for componentwise perturbations.

In these formulations, the perturbations considered are relative but similar relations can be given for absolute perturbations as we will show in the case of matrix pencils.

The definitions of normwise and componentwise pseudospectrum derive directly from the definition of pseudoeigenvalue.

- The normwise ε-pseudospectrum of A is the set of **all** normwise ε-pseudoeigenvalues of A and is defined as

$$\Sigma_\varepsilon^{\mathcal{N}}(A) = \{z \in \mathbb{C}; \exists u \in \mathbb{C}^n, \exists \Delta A \text{ with } \|\Delta A\| \leq \varepsilon \|A\| \text{ such that } (A + \Delta A)u = zu\},$$

- the componentwise ε-pseudospectrum of A is the set of **all** the componentwise ε-pseudoeigenvalues of A and is defined as

$$\Sigma_\varepsilon^{\mathcal{C}}(A) = \{z \in \mathbb{C}; \exists u \in \mathbb{C}^n, \exists \Delta A \text{ with } |\Delta A| \leq \varepsilon |A| \text{ such that } (A + \Delta A)u = zu\}.$$

The larger the ε-pseudospectra in $\mathbb{C}$, the more unstable the eigenvalues.

These concepts also apply in the case of matrix pencils.

Let A and B be any $n \times n$ complex or real matrices. By definition z is an ε-pseudoeigenvalue of (A, B) if z is an eigenvalue of $(A + \Delta A, B + \Delta B)$ with ΔA and ΔB satisfying :

- $\|\Delta A\| \leq \varepsilon\alpha$ and $\|\Delta B\| \leq \varepsilon\beta$ for normwise perturbations,
- $|\Delta A| \leq \varepsilon E$ and $|\Delta B| \leq \varepsilon F$ for componentwise perturbations.

The use of α and β allows the user to consider absolute ($\alpha = \beta = 1$) or relative ($\alpha = \|A\|$, $\beta = \|B\|$) perturbations as well as other choices for these parameters that should be induced by the physical problem.
E and F are real nonnegative matrices that are chosen by the user to specify the perturbation model to be used. In the relative case, for example, E and F are such that $E = |A|$ and $F = |B|$.

We can now define the ε-pseudospectrum of a matrix pencil.

- The normwise ε-pseudospectrum of (A, B) is the set of **all** the normwise ε-pseudoeigenvalues of (A, B) and is defined as

$$\Sigma_\varepsilon^{\mathcal{N}}(A, B) = \{z \in \mathbb{C}; \exists u \in \mathbb{C}^n, \exists \Delta A \text{ with } \|\Delta A\| \leq \varepsilon\alpha \text{ and } \Delta B \text{ with } \|\Delta B\| \leq \varepsilon\beta \text{ such that } (A + \Delta A)u = z(B + \Delta B)u\},$$

- the componentwise ε-pseudospectrum of (A, B) is the set of **all** the componentwise ε-pseudoeigenvalues of (A, B) and is defined as

$$\Sigma_\varepsilon^{\mathcal{C}}(A, B) = \{z \in \mathbb{C}; \exists u \in \mathbb{C}^n, \exists \Delta A \text{ with } |\Delta A| \leq \varepsilon E \text{ and } \Delta B \text{ with } |\Delta B| \leq \varepsilon F \text{ such that } (A + \Delta A)u = z(B + \Delta B)u\}.$$

We now describe the two PRECISE tools dedicated to the study of spectral instabilities, namely the **PTS** and **SPP** tools.

4.2 The PTS tool for Perturbed Spectra

4.2.1 Description

The target application of the **PTS** tool is the study of general **real matrices**. At this time, this tool is restricted to real matrices since the particular structure of their perturbed spectra (see Section 4.2.3) does not occur with complex matrices.

Perturbed Spectra plots are useful to study the effect of perturbations on the computed eigenvalues of a matrix submitted to a specified type of perturbation. The generation of Perturbed Spectra plots is performed in three main steps :

1. Using the user-specified perturbation model (types of perturbations, range of values, etc.), the routine generates samples of perturbed matrices.
2. The routine computes by a *backward stable algorithm* the eigenvalues of the perturbed matrices.
3. A graphical interface is then used to visualise the computed eigenvalues.

4.2.2 Use of the software

Executables

The six character names of the executables depend on the type of routine to be used. The general syntax is `ptsddX` :

- `pts` for PerTurbed Spectra,
- `d` because the matrices currently treated are stored using Dense storage format,
- `d` for Direct eigensolver, namely QR,
- `X=s` for Sequential program, `X=p` for Parallel program.

Two versions of the software are available : `ptsdds` is a sequential program while `ptsddp` is a SPMD based parallel version. Both versions treat matrices stored in dense format[2].

Parameter file

The parameters used in the **PTS** programs are read from the file `PTS_PARAMETER`. In this section, we describe the `PTS_PARAMETER` file used for the example given in Section 4.2.3, page 71 : this file is shown on Figure 4.1.

We now describe each parameter.

• `Name of the matrix` : character*72.
The user has to specify the name of the matrix.

• `Size of the matrix` : integer.
This parameter gives the size of the matrix to be studied. It has to be less than or equal to the parameter `NMAX` that is machine dependant and limits the leading dimension of the matrices that can be treated by the program[3].

[2] See appendix A for details about storage formats.
[3] See the installation notes provided with the software.

```
----- Name of the matrix -------------------------------------------
Matrice_test/ROSE
----- Size of the matrix -------------------------------------------
10
----- Type of perturbation 1 for normwise, 0 for componentwise -----
1
----- Minimal size of perturbation(1.e-16 for example) -------------
1.e-16
----- Maximal size of perturbation(1.e-8 for example) --------------
1.e-10
----- Number of steps (20 for examples) ----------------------------
8
----- Number of samples per step -----------------------------------
12
----- Type of scaling  1 for relative, 0 for absolute --------------
1
----- Number of slaves ---------------------------------------------
3
```

Figure 4.1: Parameter file for the **PTS** tool

- `Type of perturbation` : integer.

The user has to choose between two types of perturbations : 1 is used for normwise perturbations, 0 for componentwise.

- `Minimal size of perturbation` : real.

This parameter determines the minimal size of perturbations. Denoted by `EPSMIN`, it has to be **positive** and should be greater than or equal to the machine precision.

- `Maximal size of perturbation` : real.

This parameter determines the maximal size of perturbations. Denoted by `EPSMAX`, it must be greater than `EPSMIN`.

- `Number of steps` : integer.

Denoted by `NBPERTU`, this parameter determines the number of perturbation sizes that will be taken uniformly from perturbation range specified by `EPSMIN` and `EPSMAX`. For an optimal use of the **PTS** tool, `NBPERTU` should be between 10 and 20.
Note that a logarithmic scale is used to discretise the range of perturbations. For example, if we choose to go from 10^{-10} to 10^{-16} using 8 steps, the following intermediary sizes of perturbations will be used : 10^{-16}, $7.19\,10^{-16}$, $5.17\,10^{-15}$, $3.72\,10^{-14}$, $2.68\,10^{-13}$, $1.93\,10^{-12}$, $1.38\,10^{-11}$, 10^{-10}.

- `Number of samples per step` : integer.

This parameter, denoted by `NBITER`, gives the number of samples generated for **each** size of perturbation. Usually, $5 \leq$ `NBITER` ≤ 15 provides the user with a meaningful graphical representation of Perturbed Spectra.

- `Type of scaling` : integer.

This parameter specifies if the perturbations used are absolute. There are two possibilities : 1 for relative, 0 for absolute. For example, using normwise perturbations, a perturbation ΔA of size ε on a matrix A will be such that $\|\Delta A\| \leq \varepsilon \|A\|$ using relative scaling, while it will be such that $||\Delta A|| \leq \varepsilon$ with absolute scaling.

- `Number of slaves` : integer.

The user specifies the number of subtasks to spawn. This parameter is obviously used only by the parallel code `ptsddp`.

Output file

The executables produce an output file called `RESULTS`. This file contains data in the following format :

- `N` : Integer
- `NBPERTU` : Integer
- `NBITER` : Integer

$$\left.\begin{array}{ll} \bullet\ \text{Real part} & \text{Imaginary part} \\ \quad\vdots & \quad\vdots \\ \quad\vdots & \quad\vdots \end{array}\right\}\ \texttt{N}\text{ eigenvalues of }\texttt{A}$$

$$\left.\begin{array}{l} \bullet\ \texttt{PERTU}\text{(size of perturbation) : Real} \\ \left.\begin{array}{ll} \bullet\ \text{Real part} & \text{Imaginary part} \\ \quad\vdots & \quad\vdots \\ \quad\vdots & \quad\vdots \end{array}\right\}\ \texttt{N}\text{ eigenvalues of }\texttt{A}+\Delta\texttt{A} \end{array}\right\}\ \text{Repeated }\texttt{NBPERTU}\times\texttt{NBITER}\text{ times}$$

The value of `PERTU` and the eigenvalues of the corresponding perturbed matrices will occur `NBPERTU`×`NBITER` times.

Figure 4.2 shows the RESULTS file obtained with the matrix La Rose and normwise perturbations as shown in example 71.

An example of a MATLAB graphical interface can be found in Appendix B.

Steps for the user

The use of the **PTS** tool is composed of four main steps.

- **Step 1.** Edit the parameter file, `PTS_PARAMETER`.
- **Step 2.** Ensure the matrix is in the proper dense storage format. For the parallel code, ensure that `pvm` is running, and that the `hostlist` file is defined.
- **Step 3.** Run the command `ptsdds` or `ptsddp`.
- **Step 4.** Use a graphical tool to process the output file and view the perturbed spectra.

4.2.3 Example

For this example, we have used the `ptsddp` program on the matrix named La Rose. It is the

```
10
8
12
 0.9998457181762015E+00      0.0000000000000000E+00
 0.1000077140848154E+01      0.1337239311600716E-03
 0.1000077140848154E+01     -0.1337239311600716E-03
 0.1999548595480349E+01      0.7805181504504507E-03
 0.1999548595480349E+01     -0.7805181504504507E-03
 0.2000902814048370E+01      0.0000000000000000E+00
 0.2998870741195100E+01      0.0000000000000000E+00
 0.3000564626821804E+01      0.9767322701590892E-03
 0.3000564626821804E+01     -0.9767322701590892E-03
 0.4000000000279702E+01      0.0000000000000000E+00
 0.3022093903239938E-11
 0.9979632681242349E+00      0.0000000000000000E+00
 0.1001018237277036E+01      0.1780863799432818E-02
 0.1001018237277036E+01     -0.1780863799432818E-02
 0.1992246311260903E+01      0.1292420724305985E-01
 0.1992246311260903E+01     -0.1292420724305985E-01
 0.2015543527244482E+01      0.0000000000000000E+00
 0.2974618209149055E+01      0.0000000000000000E+00
 0.3012670471306856E+01      0.2158227725424235E-01
 0.3012670471306856E+01     -0.2158227725424235E-01
 0.4000004955810760E+01      0.0000000000000000E+00
 0.3022093903239938E-11
 0.9992307476959306E+00      0.1322084547826867E-02
 0.9992307476959306E+00     -0.1322084547826867E-02
 0.1001538640335503E+01      0.0000000000000000E+00
 0.1981706730849792E+01      0.0000000000000000E+00
 0.2009101335471569E+01      0.1680374363368810E-01
 0.2009101335471569E+01     -0.1680374363368810E-01
 0.2981578584109721E+01      0.3255634810466539E-01
 0.2981578584109721E+01     -0.3255634810466539E-01
 0.3036953731685986E+01      0.0000000000000000E+00
 0.3999979562568226E+01      0.0000000000000000E+00
          .                             .
          .                             .
          .                             .
          .                             .
          .                             .
```

Figure 4.2: Output file for Perturbed Spectra

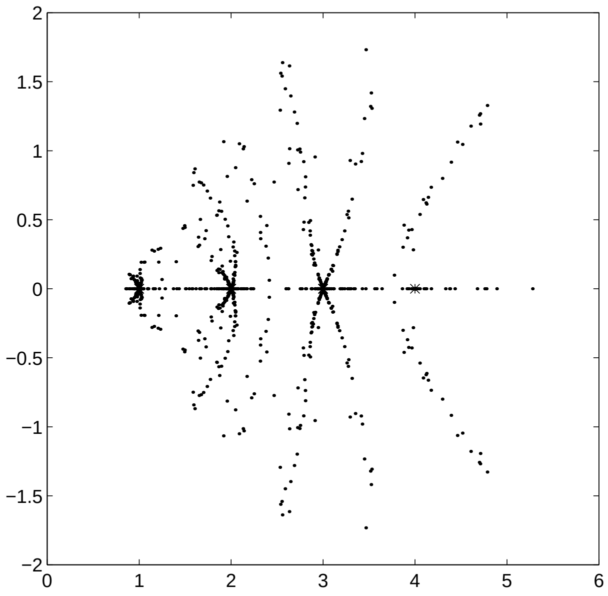


Figure 4.3: Normwise Perturbed Spectra of La Rose matrix

companion matrix associated with the polynomial $P(x) = (x-1)^3(x-2)^3(x-3)^3(x-4)$, that is,

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ -864 & 4968 & -12492 & 18086 & -16703 & 10290 & -4287 & 1194 & -213 & 22 \end{pmatrix}$$

so that its spectrum has one simple eigenvalue, equal to 4, and three defective eigenvalues of multiplicity $3 : 1, 2$ and 3 respectively.

For Figure 4.3, the specifications were the ones given in the file `PTS_PARAMETER` described in the previous section. We have used 10 sizes of normwise relative perturbations from 10^{-16} to 10^{-10} with 10 samples for each size of perturbation. The output file `RESULTS` has been treated with MATLAB and the resulting picture is given by Figure 4.3. We can observe the appearance of two curves of *spurious eigenvalues*, crossing the real axis around 2.3 and 3.8 : they result from large perturbations for which the theory of linear perturbations does not apply.

On the other hand, for Figure 4.4, we have used the same interval of perturbations on the same matrix, but instead we used componentwise perturbations. Moreover, Figure 4.4 is a close-up around the eigenvalue 2 that displays only the computed eigenvalues corresponding to small sizes (close to machine precision) of perturbations for which the theory of linear perturbations still applies. We can observe here a 6-branch star. From the star structure and the number of branches, we can infer that the eigenvalue 2 is an eigenvalue of index 3 in

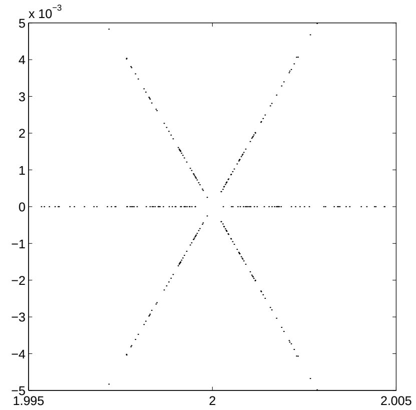


Figure 4.4: Zoom on the componentwise Perturbed Spectra of La Rose matrix

the Jordan decomposition of the matrix. More information about the qualitative analysis a Perturbed Spectra and the determination of the Jordan form of the matrix can be found in [8].

4.3 The SPP tool for Spectral Portraits

If no information is available regarding the spectrum, or if you want to use complex perturbations or treat a complex matrix or matrix pair (even real), the **PTS** tool would not be the right one; it only shows the sensitivity of the eigenvalues with respect to the particular eigensolver used to generate the eigenvalues. Therefore, if you want to study the set of complex numbers that can behave as eigenvalues during a computation, a Perturbed Spectra portrait would not provide the user with significant information. The use of the **SPP** tool allows the user to overcome some of these difficulties.

4.3.1 Description

A Spectral Portrait shows the effects of *normwise complex perturbations*, which is the most general case of perturbation, on the spectrum of a matrix or a matrix pencil. Before listing the main steps of Spectral Portrait computation, we will introduce some definitions and show how they are related with the concepts of normwise ε-pseudospectrum introduced in Section 4.1.

It has been shown (see [11, 19]) that for a matrix pencil

$$\Sigma_\varepsilon^{\mathcal{N}}(A,B) = \{z \in \mathbb{C};\ \|(A - zB)^{-1}\| \geq \frac{1}{\varepsilon(\alpha + |z|\beta)}\} \ .$$

The contour line $\{z \in \mathbb{C};\ \|(A-zB)^{-1}\| = \dfrac{1}{\varepsilon(\alpha+|z|\beta)}\}$ is then the border of the ε-pseudospectrum, and for matrices, the same formula holds with $\beta = 0$ and $B = I$.

For a nonnormal matrix pencil, $\|(A - zB)^{-1}\|$ can be large even at points z far from eigenvalues. This appears clearly on the map $z \mapsto \log_{10}(\|(A - zB)^{-1}\|(\alpha + |z|\beta))$, which is called the Spectral Portrait of the matrix pencil[4] (A, B).

For matrix pencils, the main steps for the Spectral Portrait generation consist of :

1. computing α and β, i.e. $\|A\|$ and $\|B\|$ in the case of relative perturbations,
2. choosing a region of the complex plane on which the portrait will be computed and discretising this region,
3. computing for each point z of the discretisation

$$\phi(z) = \|(A - zB)^{-1}\|(\alpha + |z|\beta) , \tag{4.1}$$

4. using a graphical visualisation tool to view the Portrait.

The use of the **SPP** programs requires the specification of a region in the complex plane which must be pre-determined. For cases where little is known a priori about the spectral properties of a matrix, an accurate and efficient means to compute the outer boundaries of the pseudospectra can be used : in [3], it is shown that the field of values is an effective way to determine the extremities of the pseudospectrum of interest. This facility is explained in Section 4.3.2.

4.3.2 Use of the software

Executables

Eight different versions of the software are available, each applicable to a specific storage format, type of eigenvalue problem, and machine type. The six character names of the respective executables indicate the type of problem and the data to be treated. The general syntax is `sppXYZ`.
The three first characters are always the same, `spp` for SPectral Portrait. The characters `XYZ` are dependent on the problem :

- `X` depends on the storage format[5] of the matrices, `X=d` for Dense and `X=s` for Sparse.
- `Y` deals with the type of problem, `Y=g` for Generalized (matrix pencil), `Y=s` for Standard (matrix).
- finally, `Z` concerns the type of code to be used, `Z=s` for Sequential code, `Z=p` for Parallel code.

For example, `sppdsp` is the parallel code dedicated to the computation of the portrait of a matrix stored in dense format, while `sppsgs` is the sequential code to deal with sparse matrix pencils.
Table 4.1 lists all the executables that are available.

[4] For the matrix A, we set $B = I$ and $\beta = 0$.
[5] See Appendix A for details about storage formats.

Storage format	Type of eigenproblem	Type of code	Name of the executable
Dense	Standard	Sequential	sppdss
Dense	Standard	Parallel	sppdsp
Dense	Generalized	Sequential	sppdgs
Dense	Generalized	Parallel	sppdgp
Sparse	Standard	Sequential	sppsss
Sparse	Standard	Parallel	sppssp
Sparse	Generalized	Sequential	sppsgs
Sparse	Generalized	Parallel	sppsgp

Table 4.1: Executables for Spectral Portrait computation

Parameter files

The parameters used during spectral portrait computation are read by the program from the file `SPP_PARAMETERS.GEN` for the study of a matrix pencil, or from the file `SPP_PARAMETERS.STA` for a standard matrix.

In this section we describe how to set the parameters by illustration of two examples. The first example deals with the spectral portrait of a standard matrix and is related to the matrix studied in Section 4.3.3 page 85. Figure 4.12 was obtained using the file `SPP_PARAMETERS.STA` given in Figure 4.8.

```
------ Name of the matrix file ------------------------------------------
Matrice_test/Electro
------ Size of the matrix ----------------------------------------------
 10
------ 1 for real matrix, 0 for complex ---------------------------------
 1
------ X min, X max, Y min, Y max ---------------------------------------
 -0.1  1.1  0.0 1.0
------ X grid, Y grid ---------------------------------------------------
 128   128
------ Name of the output file ------------------------------------------
spp_results
------ (parallel only) PVM spawn - 0=hostlist file, 1=mpp, 2=no spawn ---
0
------ (parallel only) Number of tasks to spawn -------------------------
2
------ (parallel only) Message sizes (1-100) ----------------------------
10
------ end of parameter list --------------------------------------------
```

Figure 4.5: Parameter file for standard Spectral Portrait

We now describe each parameter.

• `Name of the matrix` : character*72.
The user has to specify the name of the matrix.

• `Size of the matrix` : integer.
This parameter gives the size of the matrix to be studied. It must be less than or equal to the parameter `NMAX` used during compilation of the program. The maximum size of `NMAX` is machine dependent.
Moreover, it should be noted that for sparse matrices, the maximum number of non-zeros is determined by the value of `NZMAX` used during compilation. If the matrix has significant fill-in during the LU factorisation of $(A - zI)$, the parameters `NMAX` and/or `NZMAX` may need to be adjusted and the program re-compiled.

• `1 for real matrix, 0 for complex` : integer.
This parameter specifies the type of matrix to be studied. Two possibilities only : `1` or `0`.

• `Xmin, Xmax, Ymin, Ymax` : real, real ,real, real.
These parameters determine the region of the complex plane to be studied.
It should be noted that, if the user does not know, *a priori*, the region to be studied, he may set `Xmin=Xmax` and `Ymin=Ymax`. In this case, a region enclosing the spectrum will be determined automatically[6] using a field of values computation, see [3]. In the same way, setting `Xmin=Xmax` (resp. `Ymin=Ymax`) will lead to an automatic determination of `Xmin` and `Xmax` (resp. `Ymin` and `Ymax`).

• `Xgrid, Ygrid` : integer, integer.
The region previously specified will be discretised using the mesh set by `Xgrid` and `Ygrid`. In general, the study of a matrix with Spectral Portraits is performed by generating more than one portrait. As such, the first computation should be done with coarse grid values (about 30) of `X grid` and `Y grid`. These parameters can then be increased for a more refined portrait after the determination of a region of interest.

• `Name of the output file` : character*72.
By default, the output file is `spp_results` but the user has the possibility to specify another name.

The three last parameters only concern the parallel routines (`sppdsp, sppssp`). They do not have to be specified with the sequential versions (`sppdss, sppsss`).

• `PVM spawn - 0=hostlist file, 1=mpp, 2=no spawn` :integer.
This parameter specifies the PVM spawn mechanism. It must be set to 0 for *spawn tasks one at a time on machines from* `hostlist`, 1 for *spawn all tasks at once* and 2 for *no spawn required*.

• `Number of tasks to spawn` : integer.
This parameter specifies the number of subtasks to spawn. Note that if P processors are used, then it is assumed that the first processor will perform the spawning of the other tasks. Therefore, this value should be $P - 1$.

• `Message sizes` : integer.
This last parameter determines the number of points of the grid given to each task in a single communication. Lower values of this parameter require more communication, but result in better load balancing.

Finally, regarding the region to be studied, it should be noted that for a real matrix (or matrix pencil) the spectrum is symmetric with respect to the real axis. Therefore, if the user wants to study the region $[0.0, 5.0] \times [-3.0, 3.0]$, the computation can be restricted to the region $[0.0, 5.0] \times [0.0, 3.0]$.

Figures 4.6 and 4.7 give examples of the use of the parameters. Figure 4.6 illustrate

[6] In this case, the user should specify a coarse grid, `Xgrid=Ygrid=30` for example, and the first portrait obtained should be used to determine a more appropriate region.

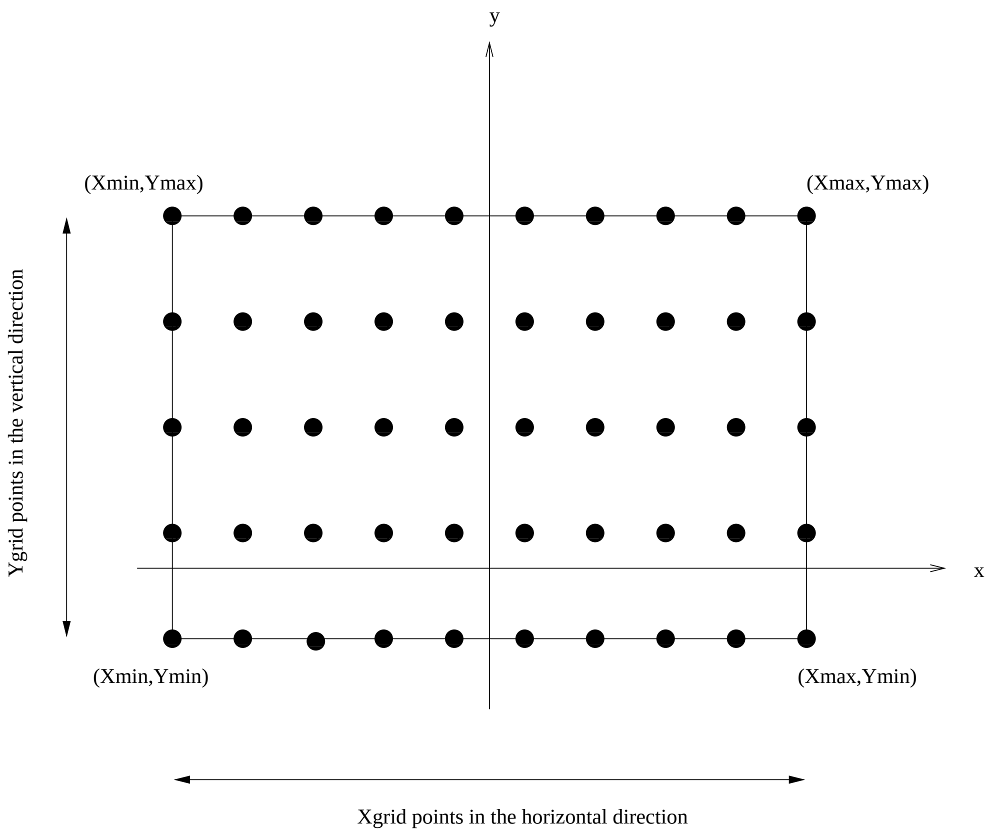


Figure 4.6: Discretisation of the region

how the region to be studied is defined through `Xmin, Xmax, Ymin, Ymax` and discretised using `Xgrid` and `Ygrid`.

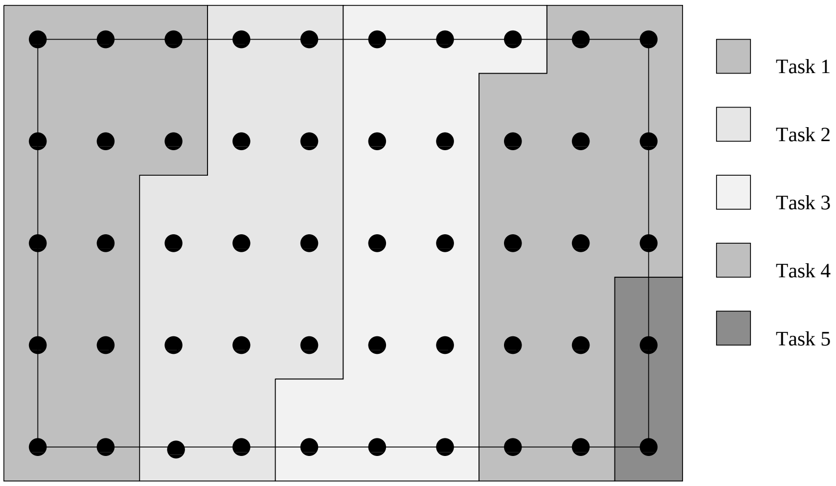


Figure 4.7: Representation of the subtasks to be spawned

On the other hand, Figure 4.7 shows how the region is decomposed to be treated by the parallel programs. It should be noticed that the parallel platform used can be a network of heterogeneous workstations. A task consists of the treatment of a subset of points of the grid; the computational time can vary considerably between two different points of the grid. Therefore the parallel management module takes into account the load unbalance that may originate from the variability in the load and in the computational power of the various nodes, or in the computational time required for each task. The granularity is controlled through the number of points per task (`Message sizes`). At the beginning, each node receives a task to treat. As soon as it has finished its computation, the result is stored and the node receives another task if there is one remaining. On the example of Figures 4.6 and 4.7, the region is discretised by 50 points and each task consists of the treatment of 12 points (`Message size`). It should be noticed that the last subtask contains only 2 points.

The second example concerns the spectral portrait of a matrix pencil shown in Figure 4.16 which was obtained using the file `SPP_PARAMETERS.GEN` shown in Figure 4.8.

This parameter file only differs from the one previously described in the parameters related to α and β that appear in Equation 4.1. We now describe them.

• `Scaling :  1 for relative, 0 for absolute` : integer.
This paramater allows the user to indicate if α (resp. β) is proportional to $||A||$ (resp. $||B||$), option `Scaling=1`, or not, option `Scaling=0`. When using relative scaling, α (resp. β) is expressed as $\alpha = a||A||$ (resp. $\beta = b||B||$) and the program will determine the values of the norms. They need not be specified. Otherwise if `Scaling=0`, $\alpha = a$ and $\beta = b$. As described below, a and b can be fixed by the user.

• `a and b fixed below` : integer.
This parameters provides the user with the possibility to set the values of a and b. A value of `1` indicates that a and b are specified as parameters. A value of `0` sets by default a and b to `1`. Therefore if `Scaling=1`, α (resp. β) is equal to $||A||$ (resp. $||B||$) and if `Scaling=0` α and β are set to `1`.

• `a, b` : real, real.
If the user has set the previous parameter to 1, he has to set the values of a and b. When

```
------ Name of the matrix file A ---------------------------------------
Matrice_test/ROSE
------ Name of the matrix file B ---------------------------------------
Matrice_test/DiagInv
------ Size of the matrix ----------------------------------------------
 10
------ 1 for real matrix, 0 for complex --------------------------------
 1
------ X min, X max, Y min, Y max --------------------------------------
 -20.0  100.0  -60.0  60.0
------ X grid, Y grid --------------------------------------------------
 64   64
------ Name of the output file -----------------------------------------
spp_results
------ Scaling : 1 for relative, 0 for absolute ------------------------
0
------ a and b fixed below? 1 for yes, 0 for no ------------------------
1
------ a, b ------------------------------------------------------------
1.0d0  1.0d-3
------ (parallel only) PVM spawn - 0=hostlist file, 1=mpp, 2=no spawn ---
0
------ (parallel only) Number of siblings to spawn ----------------------
2
------ (parallel only) Message sizes (1-100) ----------------------------
10
------ end of parameter list -------------------------------------------
```

Figure 4.8: Parameter file for generalized Spectral Portrait

`Scaling=1` the user actually specifies the values of a and b such that $\alpha = a\|A\|$ and $\beta = b\|B\|$. A commented example is given for Figure 4.16, page 88.

Table 4.2 shows with examples all the cases that can be treated with respect to the three parameters descibed before.

`Scaling`	`a and b fixed?`	`a, b`	Expression of $\phi(z) = \|(A - zB)^{-1}\|(\alpha + \|z\|\beta)$
`1`	`0`	By default `a=b=1` $\alpha = \|A\|,\ \beta = \|B\|$	$\|(A - zB)^{-1}\|(\|A\| + \|z\|\|B\|)$
`1`	`1`	`a` and `b` user defined $\alpha = a\|A\|,\ \beta = b\|B\|$	$\|(A - zB)^{-1}\|(a\|A\| + \|z\|b\|B\|)$
`0`	`0`	By default `a=b=1` $\alpha = 1,\ \beta = 1$	$\|(A - zB)^{-1}\|(1 + \|z\|)$
`0`	`1`	`a` and `b` user defined $\alpha = a, \beta = b$	$\|(A - zB)^{-1}\|(a + \|z\|b)$

Table 4.2: Value of the parameters for all the possible cases

The three last options concern the parallel version of the software (`sspdgp, sppsgp`) and do not need to be set in the case of the sequential versions (`sppdgs, sppsgs`).

• `PVM spawn - 0=hostlist file, 1=mpp, 2=no spawn` :integer.
This parameter specifies the PVM spawn mechanism. It must be set to 0 for *spawn tasks one at a time on machines from* `hostlist`, 1 for *spawn all tasks at once* and 2 for *no spawn required.*

• `Number of tasks to spawn` : integer.
This parameter specifies the number of subtasks to spawn. Note that if P processors are used, then it is assumed that the first processor will perform the spawning of the other tasks. Therefore, this value should be $P - 1$.

• `Message sizes` : integer.
This last parameter determines the number of points of the grid given to each task in a single communication. Lower values of this parameter require more communication, but result in better load balancing.

Output file

The executables produce an output file, called `spp_results` by default. This file contains data in the format :

- `FilenameA` : character, name of the matrix A.
- `FilenameB` : character, name of the matrix B.

• `Xgrid` : Integer • `Xmin` : Real • `Xmax` : Real

• `Ygrid` : Integer • `Ymin` : Real • `Ymax` : Real

• `I` : Integer • `J` : Integer • `Res` : Real

⋮ ⋮ ⋮

The line " `I J Res`" will occur `Xgrid`×`Ygrid` times, i.e., for each point z of the discretisation. `I` (resp. `J`) is the position of z in the real (resp. imaginary) direction and `Res` is the value of $\phi(z)$.
For example, '`10 17 0.5346E+01`' means $z = (x(10), y(17))$ and $\phi(z) = 0.5346E + 01$ with $x(10) = \mathtt{Xmin} + (10-1) \times Xstep$ where $Xstep = (\mathtt{Xmax} - \mathtt{Xmin})/(\mathtt{Xgrid} - 1)$ and similarly for $y(17)$.

The example given in Figure 4.9 is the beginning of the `spp_results` file obtained from a computation applied on the Rose Matrix using the region $[0.0, 5.0] \times [-2.5, 2.5]$ discretised by a 30×30 grid. An example of a MATLAB graphical interface can be found in Appendix A.

Steps for the user

The use of the Spectral Portrait computation tool is composed of four main steps :

- **Step 1.** Edit the parameter file, `SPP_PARAMETER.GEN` or `SPP_PARAMETER.STA`.
- **Step 2.** Ensure the matrices are in the proper storage format.
- **Step 3.** Use one of the commands listed in Table 4.1.
- **Step 4.** Use a graphical tool to treat the output file and view the spectral portrait.

4.3.3 Examples of use and interpretation

Spectral instability

For a normal matrix, which does not exhibit spectral instability (see [5, 19]), the ε-pseudospectrum (where ε is the machine precision) comprises the union of the closed balls of radius $\varepsilon\alpha$ about each eigenvalue, but for a nonnormal matrix the ε-pseudospectrum can be much larger than the union of these $\varepsilon\alpha$-balls.

The following example deals with the Wilkinson matrix (see [15]) denoted by W. The matrix is defined by :

$$W(i,i) = i \ , \ W(i,i+1) = n \text{ for } i = 1, ..., 49 \text{ and } W(n,n) = n \ .$$

Here, we have used $n = 50$. The spectrum of W is the set of integers from 1 to 50. If we compute the spectrum of W using the QR algorithm (which is backward stable), the computed eigenvalues match the exact ones. Let us consider now, the matrix Q^*WQ where Q is a unitary matrix($Q^*Q = I$). Q^*WQ and W have exactly the same spectrum. However, if we compute the eigenvalues of Q^*WQ with QR, the results are completely different. The eigenvalues, see Figure 4.10, are located in an ellipse around the exact spectrum. The spectral portrait of W, see Figure 4.11, enlightens this phenomenon. The computed eigenvalues of Q^*WQ lie on the border or inside the 10^{-16}-pseudospectrum of W. The value 10^{-16} corresponds to the double machine precision used during the computations with QR. For the computer, **all** the complex points inside this 10^{-16}-pseudospectrum are eigenvalues of the same numerical quality.

```
/home/toumazou/Matrice_test/ROSE
 30   .000000E+00   .500000E+01
 30  -.250000E+01   .250000E+01
  1   1   .691750E+01
  2   1   .704526E+01
  3   1   .718446E+01
  4   1   .733349E+01
  5   1   .749025E+01
  6   1   .765232E+01
  7   1   .781708E+01
  8   1   .798183E+01
  9   1   .814399E+01
 10   1   .830111E+01
 11   1   .845104E+01
 12   1   .859190E+01
 13   1   .872213E+01
 14   1   .884054E+01
 15   1   .894623E+01
 16   1   .903863E+01
 17   1   .911746E+01
 18   1   .918272E+01
 19   1   .923465E+01
 20   1   .927372E+01
  1   2   .685372E+01
  2   2   .699209E+01
  3   2   .714445E+01
  4   2   .730884E+01
  5   2   .748268E+01
  6   2   .766296E+01
  7   2   .784642E+01
  8   2   .802979E+01
  9   2   .820993E+01
 10   2   .838396E+01
  .   .       .
  .   .       .
  .   .       .
  .   .       .
```

Figure 4.9: Output file for Spectral Portrait

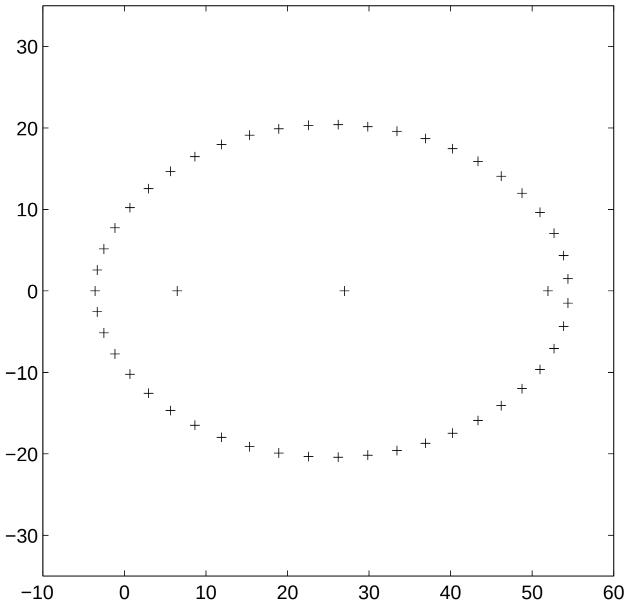


Figure 4.10: Computed eigenvalues of Q^*WQ

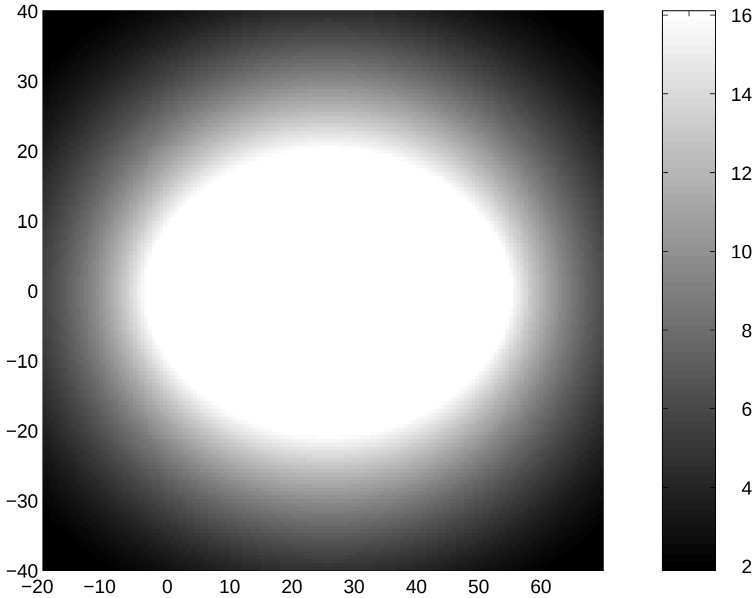


Figure 4.11: Spectral Portrait of W

Convergence of iterative scheme

In this example, the matrix A of size 288×288 comes from an electromagnetism problem that deals with the diffraction of a transverse-magnetic wave by a periodic $2D$ structure. This example has been submitted by the Electromagnetism Project at CERFACS [10].
The electric field x_k at time $k\Delta t$ (where Δt is the time step), is computed using an iterative scheme $x_k = Ax_{k-1} + b_{k-1}$ where b_k tends to 0. Due to the experimental assumptions, x_k should tend to 0 as k increases. Mathematically, the scheme converges to 0 provided that the spectral radius of A is smaller than 1. This fact holds in exact arithmetic. But in finite precision the scheme diverges. Figure 4.12 shows that the pseudospectrum spreads out of the unit ball for relatively small matrix perturbations. Figure 4.13 shows a close-up of the spectral portrait in the neighbourhood of the point $z = 1$: the contour line which is displayed is the border of the ε-pseudospectrum of A for $\varepsilon = 10^{-16}$. Due to finite precision arithmetic, the iteration matrix is not exactly A but a perturbed matrix $A + \Delta A$. And indeed for relative perturbations of the order of the machine double precision ($\varepsilon = 10^{-16}$), the ε-pseudospectrum includes complex points of modulus larger than 1: these points can be eigenvalues of the iteration matrix in finite precision. This explains the possibility for the iterative scheme to diverge in finite precision.

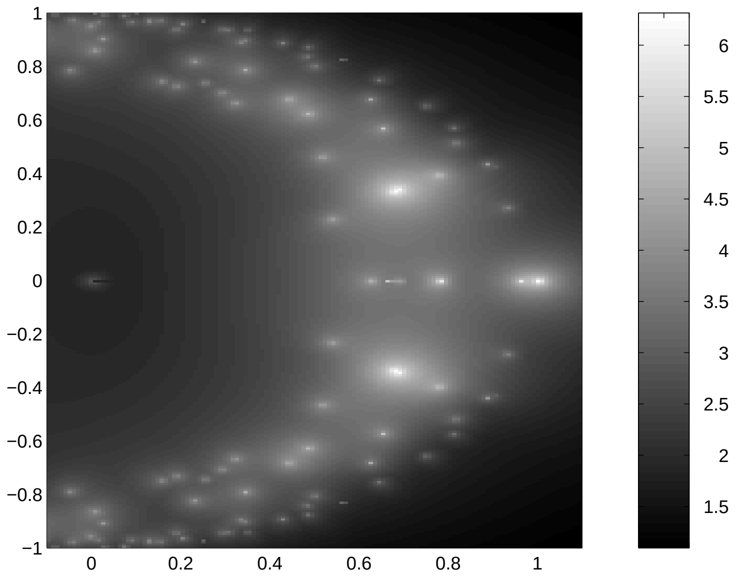


Figure 4.12: Spectral portrait of the Electromagnetism matrix

Case of matrix pencil

The use of the **SPP** tool is not restricted to the assessment of the numerical quality in finite precision computations. Generally, the matrices A and B are obtained at the final stage of the treatment of a physical problem in numerical simulation. They are many reasons why the entries of A and B can be subject to perturbations. They can simulate physical perturbations (such as deformations), to take into account data uncertainty (limited accuracy of physical measurements), or also deal with numerical perturbations such as the truncation error of an approximation scheme, discretisation of derivatives, iterative

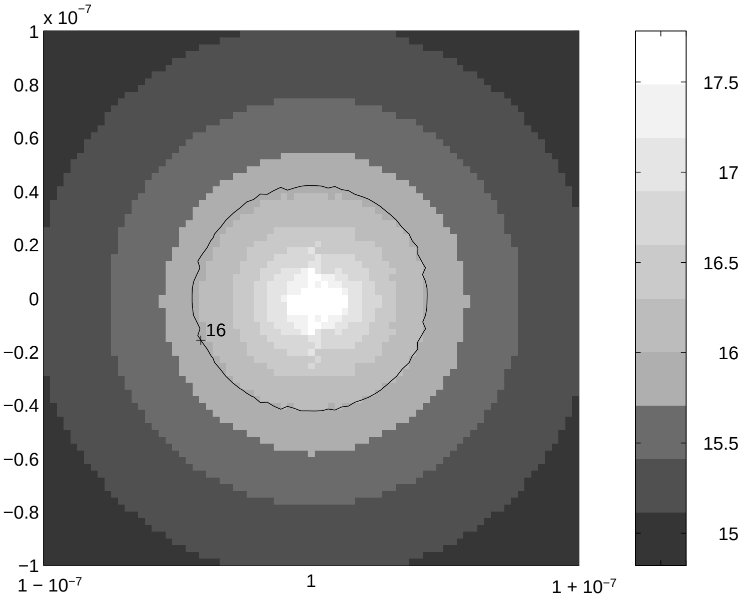


Figure 4.13: Electromagnetism matrix, close-up around the point $z = 1$

methods, rounding errors, etc. In such cases, the perturbations are known independently on A and B and it would not be appropriate to work on $B^{-1}A$ (this might even be impossible if B were singular). On the following academic example, we illustrate the difference between the spectral portraits of (A, B) and $B^{-1}A$.

Let A be the companion matrix of order 10 associated with the monic polynomial $p(x) = (x-1)^3(x-2)^3(x-3)^3(x-4) = x^{10} + \sum_{i=0}^{9} a_i x^i$. The ten coefficients are $a_0 = 864$, $a_1 = -4968$, $a_2 = 12492$, $a_3 = -18086$, $a_4 = 16703$, $a_5 = -10290$, $a_6 = 4287$, $a_7 = -1194$, $a_8 = 213$ and $a_9 = -22$. Let B be the diagonal matrix whose i^{th} entry equals $1/i$.

Figure 4.14 gives the generalized spectral portrait for the matrix pair (A, B), and for $\alpha = \|A\|$ and $\beta = \|B\|$. The mesh size is 128×128. The choice of a scale in shades of grey clearly shows the contours of the ε-pseudospectra for values of ε from 10^{-5} to 10^{-11}. Consider the region enclosed by the border associated with $\varepsilon = 10^{-9}$: this region is the set of all eigenvalues of all the matrix pairs $(A + \Delta A, B + \Delta B)$ for $\|\Delta A\| \leq 10^{-9}\|A\|$, $\|\Delta B\| \leq 10^{-9}\|B\|$.
Another interpretation is the following. Suppose that the matrices A and B are known to the normwise relative precision 10^{-7} for instance. Then, the region enclosed by the border associated with $\varepsilon = 10^{-7}$ is the set of all possible eigenvalues for the matrix pair. In our case, this region is very large because the matrix A is a nonnormal matrix with defective eigenvalues.

Figure 4.15 gives the classical spectral portrait of the matrix $B^{-1}A$. We can see that the ε-pseudospectra are rather different than in the generalized spectral portrait : they tend to be larger. This portrait can be useful for analyzing the quality of a computed eigenvalue $\tilde{\lambda}$ of the matrix pair (A, B) obtained with an algorithm using (implicitly or explicitly) the matrix $B^{-1}A$. If we compute the backward error $\eta(\tilde{\lambda})$ associated with $\tilde{\lambda}$, the spread of the pseudospectrum of level $-\log_{10}\eta(\tilde{\lambda})$ gives information about the accuracy of the computed

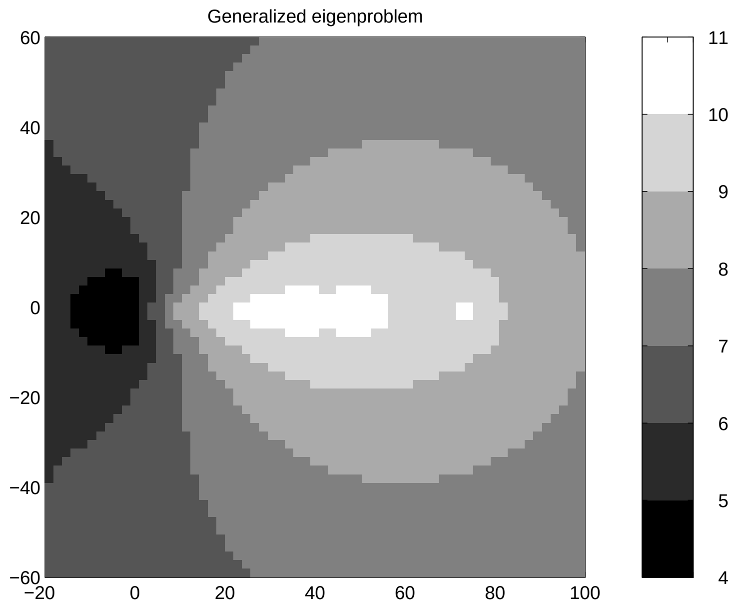


Figure 4.14: Generalized spectral portrait for (A, B), $\alpha = \|A\|$ and $\beta = \|B\|$

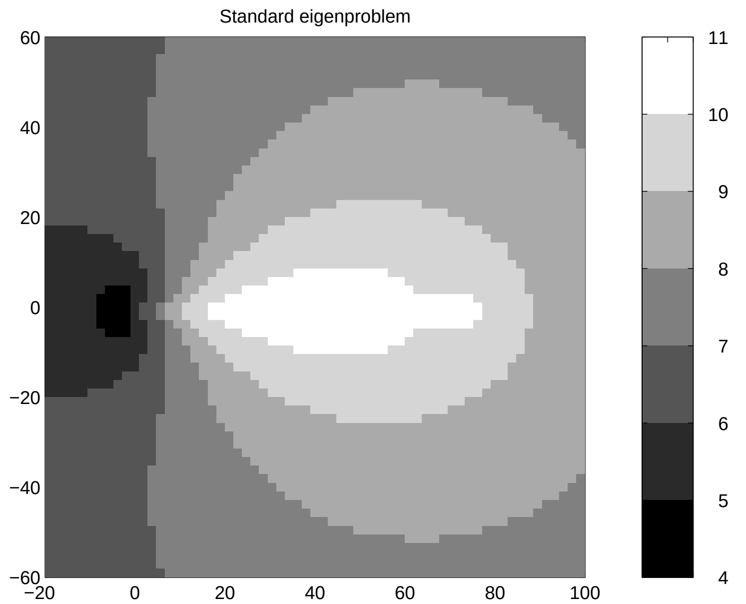


Figure 4.15: Spectral portrait for $B^{-1}A$, $\alpha = \|B^{-1}A\|$

eigenvalue $\tilde{\lambda}$ for $B^{-1}A$.

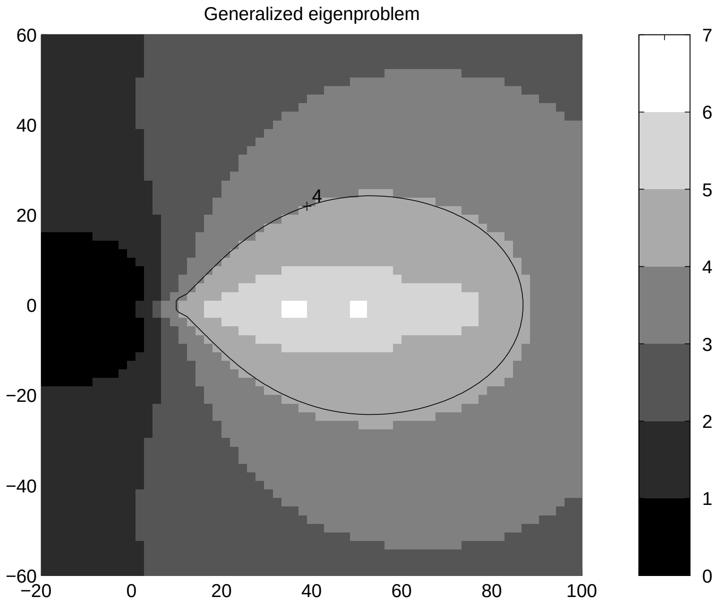


Figure 4.16: Spectral portrait for (A, B), $\alpha = 1$ and $\beta = 10^{-3}$

Suppose now that the matrix A is known within an absolute tolerance of 10^{-4} and B within an absolute tolerance of 10^{-7}. We want to visualize all the possible eigenvalues for all the pairs representing (A, B) within the given tolerance:

$$\begin{cases} \|\Delta A\| \leq 10^{-4} = \alpha\varepsilon, \\ \|\Delta B\| \leq 10^{-7} = \beta\varepsilon. \end{cases}$$

We can choose $\alpha = 1$, $\beta = 10^{-3}$ and $\varepsilon = 10^{-4}$: this corresponds to the spectral portrait given in Figure 4.16. The contour line of level $-\log_{10}\varepsilon = 4$ encloses the desired set. It is displayed on Figure 4.16. Of course, there are an infinite number of possible choices for the triple $(\alpha, \beta, \varepsilon)$. All the triples such that $\alpha\varepsilon = 10^{-4}$ and $\beta\varepsilon = 10^{-7}$ are admissible and would give the same information. It is remarkable that *one* curve (the border of the ε-pseudospectrum) encloses all the eigenvalues for pairs of matrices resulting from *independent* perturbations on A and B. This makes the spectral portrait particularly easy to interpret and a tool of choice for a sensitivity analysis.

Appendix A

Storage formats for input matrices

As was explained in the previous sections, two types of storage format are used in PRECISE Module2, namely the **Dense** and the **Sparse** that we will now describe.

A.1 Dense storage format

The matrix $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$ should be stored column by column as follows :
a
c
b
d

A.2 Sparse storage format

The storage format used for sparse matrices is the Compressed Sparse Row (CSR, see [17]). The data structure consists of three arrays.

- A real array A containing the values a_{ij} stored row by row, from row 1 to N where N is the size of A. The length of A is NNZ where NNZ is the number of nonzeros entries of A.

- An integer array JA containing the column indices of the elements a_{ij} as stored in the array A. The length og JA is NNZ.

- An integer array IA containing the pointers to the beginning of each row in the arrays A and JA. The length of IA is $N + 1$.

The matrixfile should contain N, NNZ and the three arrays on **one column**, as described here :
N
NNZ
$IA(1)$
$\vdots$
$IA(N + 1)$
$JA(1)$

$\vdots$
$JA(NNZ)$
$A(1)$
$\vdots$
$A(NNZ).$

Appendix B

MATLAB Graphics Routines

Enclosed are example graphics routines to generate the plots described in this document. The routines make use of the graphics capabilities of MATLAB [18].

B.1 Module 1 - Linear Systems

Here is an example graphics routine to generate plots of the statistical indicators for linear systems. Note that additional information saved in the files is also displayed. The program may be modified to enable interactive specification of the backward error.

```
%
% Graphics routine for PRECISE Module 1.
% Statistical indicators for linear systems.
%
filename='fout.dat';
fid=fopen(filename);
if (fid <= 0)
   filename
   error('cannot open file');
end;

fout = fscanf(fid,'%lf',[1 7]);

ptype    = fout(1);
alpha    = fout(2);
beta     = fout(3);
xcalnorm = fout(4);
nres     = fout(5);
nerr     = fout(6);
numptb   = fout(7);
ref = alpha*xcalnorm + beta;

fout = fscanf(fid,'%lf %lf %lf %lf',[5 inf]);
fclose(fid);
fout = fout';

Absc = fout(:,1);
I  = fout(:,2);
L  = fout(:,3);
K  = fout(:,4);
```

```
Er = fout(:,5);

% get backward error estimate from graph
%% cla;
%% loglog(Absc,I);
%% title('SELECT POINT of BACKWARD ERROR:');
%% [berr,qual] = ginput(1);
%% if (berr > 1.e-14)
%%    Er = berr*K;
%% else
%%    Er = (nres/ref)*K;
%% end
%% cla;

%
if (nerr > 0)
   % set forward error value for graph
   Ex = ones(size(Er));
   Ex = Ex*(nerr/xcalnorm);
   graph(2,Absc,I,L,K,Er,Ex);
else
   graph(2,Absc,I,L,K,Er);
end

%
% show computed error values
%
subplot(2,3,5);
axis('off')
ti = sprintf('Backward error  :   %g\n',nres/ref);
text(0.0,1.0,ti);
ti = sprintf('Residual norm   :   %g\n',nres);
text(0.0,0.9,ti);
ti = sprintf('||x|| :   %g\n',xcalnorm);
text(0.0,0.8,ti);
ti = sprintf('alpha :   %g\n',alpha);
text(0.0,0.7,ti);
ti = sprintf('beta  :   %g\n',beta);
text(0.0,0.6,ti);

if (ptype==1)
   ti = 'Normwise perturbations';
else
   ti = 'Componentwise perturbations';
end
text(0.0,0.55,ti);

%% ti = sprintf('Precise estimate:   %g\n',berr);
%% text(-0.5,0.85,ti);
if (nerr > 0)
   ti = sprintf('Forward error   :   %g\n',nerr/xcalnorm);
   text(0.0,0.4,ti);
end
```

B.2 Module 1 - Standard Eigenvalues

Here is an example graphics routine to generate plots of the statistical indicators for standard eigenproblems. Note that additional information saved in the files is also displayed.

```
%
% Graphics for PRECISE Module 1
% Statistical Indicators for standard eigensolvers.
%
dbmin = 1.e-14;

filename='fout.dat';
fid=fopen(filename);
if (fid <= 0)
   filename
   error('cannot open file');
end;

fout = fscanf(fid,'%lf',[1 7]);

ptype  = fout(1);
alpha  = fout(2);
ascent = fout(3);
xcalnorm = fout(4);
nres   = fout(5);
nerr   = fout(6);
numptb = fout(7);
ref = alpha*xcalnorm;

fout = fscanf(fid,'%lf %lf %lf %lf',[7 inf]);
fclose(fid);
fout = fout';

Absc = fout(:,1);
I    = fout(:,2);
L    = fout(:,3);
K    = fout(:,4);
Er   = fout(:,5);
Ka   = fout(:,6);
Ea   = fout(:,7);

% get backward error estimate from graph
%% cla;
%% loglog(Absc,I);
%% title('SELECT POINT of BACKWARD ERROR:');
%% [berr,qual] = ginput(1);
%% if (berr > dbmin)
%%    Er = berr*K;
%% else
%%    Er = (nres/ref)*K;
%% end
%% cla;
%% if (nres>0.0)
%%    Er = (nres/ref)*K;
%% else
```

```
%%    Er = (dbmin/ref)*K;
%% end;

if (nerr > 0)
   % set forward error value for graph
   Ex = ones(size(Er));
   Ex = Ex*(nerr/xcalnorm);
   grapheig(2,Absc,I,L,K,Er,ascent,Ka,Ea,Ex);
else
   grapheig(2,Absc,I,L,K,Er,ascent,Ka,Ea);
end;

%
% show computed error values
%
subplot(2,3,6);
axis('off')
ti = sprintf('Backward error  : %g\n',nres/ref);
text(0.0,0.6,ti);
ti = sprintf('Residual norm   : %g\n',nres);
text(0.0,0.5,ti);
ti = sprintf('Eigenvalue mod. : %g\n',xcalnorm);
text(0.0,0.4,ti);
ti = sprintf('alpha :   %g\n',alpha);
text(0.0,0.3,ti);

if (ptype==1)
   ti = 'Normwise perturbations';
else
   ti = 'Componentwise perturbations';
end
text(0.0,0.2,ti);

%% ti = sprintf('Precise estimate:   %g\n',berr);
%% text(-0.5,0.85,ti);
if (nerr > 0)
   ti = sprintf('Forward error   :   %g\n',nerr/xcalnorm);
   text(0.0,0.1,ti);
end
```

B.3 Module 2 - Perturbed Spectra

Here is an example of the graphical routine that has been used to produce the figure 4.3.

```
function PineaplPerturb(filename)
%
% PINEAPLPERTURB  displays perturbed spectra.
%
%               ShowPerturb('filename') displays the perturbed spectra
%               held in 'filename'.
%
%
figure
if (nargin == 0)
```

```
  fid=fopen('results');
else
  fid=fopen(filename);
end
if (fid <= 0)
   error('cannot open file: results');
end
n=fscanf(fid,'%i',1);
nbpertu=fscanf(fid,'%i',1);
nbiter=fscanf(fid,'%i',1);
val1=zeros(2,n);
val2=zeros(2,n*nbiter);
val1=fscanf(fid,'%lf %lf',[2 n]);
for i=1:n
    eig1(i)=val1(1,i)+sqrt(-1)*val1(2,i);
end
plot(real(eig1),imag(eig1),'*');
hold on;
for j=1:nbpertu*nbiter
    size_of_perturbation=fscanf(fid,'%lf',1);
    val2=fscanf(fid,'%lf %lf',[2 n]);
    for i=1:size(val2,2)
        plot(val2(1,i),val2(2,i),'.')
    end
    hold on;
end
%
end
```

B.4 Module 2 - Spectral Portrait

Here is an example of the graphical routine that has been used to produce all the spectral portraits shown in this document.

```
function ShowPortrait(filename)
% ShowPortraitdisplays a spectral portrait from data in a file.
%
%     ShowPortrait displays the portrait held in 'spp_results'.
%     ShowPortrait('filename') displays the portrait held in 'filename'.
%
%     File Format:
%          The boundary of the region is defined by [xmin xmax ymin ymax].
%          The region is discretised into points [1:xgrid, 1:ygrid].
%          The results are read in the format
%
%                     <<filename string>>
%                     xgrid  xmin  xmax
%                     ygrid  ymin  ymax
%                     i      j     res(i,j)
%                     i      j     res(i,j)
%                     .      .     .
%                     .      .     .
%              for i=1:xgrid, j=1:ygrid.
%
```

```
%          Note that the results are not required to be in order.
%
figure
ncol = 64;

if (nargin == 0)
  filename = 'spp_results';
end;
fid=fopen(filename);
if (fid <= 0)
   filename
   error('cannot open file');
end;

% scan first lines, if they are text just count them
iscnt = 0;
S = fscanf(fid,'%s',1);
while(size(str2num(S),1)==0)
  iscnt=iscnt+1;
  S = fscanf(fid,'%s',1);
end;
fclose(fid);
%% now, reopen and show those lines
fid=fopen(filename);
for i=1:iscnt
  S = fscanf(fid,'%s',1)
end;

% next 2 lines hold  mesh,min,max
G = fscanf(fid,'%i %lf %lf',[3 2]);
xmesh = G(1,1);
x1 = G(2,1);
x2 = G(3,1);
ymesh = G(1,2);
y1 = G(2,2);
y2 = G(3,2);
% read all the elements:  ix, iy, res
G = fscanf(fid,'%i %i %lf',[3 xmesh*ymesh]);
fclose(fid);

%% set up grid vectors X (columns) and Y (rows)
xstep = (x2 - x1) / (xmesh - 1);
ystep = (y2 - y1) / (ymesh - 1);
X = zeros(xmesh,1);
Y = zeros(ymesh,1);
for ky = 0:ymesh - 1
   Y(ky+1) = y1 + ky*ystep;
end
for kx = 0:xmesh - 1
   X(kx+1) = x1 + kx*xstep;
end

%% reformat for display (note tranposition for pcolor)
A = zeros(ymesh,xmesh);
```

```
for i = 1:xmesh*ymesh
  kx = G(1,i);
  ky = G(2,i);
  A(ky,kx) = G(3,i);
end;

pcolor(X,Y,A);
colormap(hot(ncol));
axis('square');
shading flat;
colorbar;
end
```

Bibliography

[1] M. Arioli, I. S. Duff, and D. Ruiz. Stopping criteria for iterative solvers. *SIAM J. Matrix Anal. Appl.*, 13:138–144, January 1992.

[2] R. Barrett, M. Berry, T. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. van der Vorst. *Templates for the solution of linear systems : building blocks for iterative methods.* SIAM, Philadelphia, 1994.

[3] T. Braconnier, A. McCoy, and V. Toumazou. Using the field of values for pseudospectra generation. Tech. Rep. TR/PA/97/28, CERFACS, Toulouse, France, 1997.

[4] F. Chaitin-Chatelin. Is nonnormality a serious computational difficulty in practice ? In R. Boisvert, editor, *The quality of Numerical Software*, pages 300–314. Chapmann and Hall, 1997.

[5] F. Chaitin-Chatelin and V. Frayssé. *Lectures on Finite Precision Computations.* SIAM, Philadelphia, 1996.

[6] F. Chatelin. *Valeurs propres de matrices.* Masson, Paris, 1988.

[7] F. Chatelin. *Eigenvalues of matrices.* Wiley, Chichester, 1993. Enlarged Translation of the French Edition with Masson.

[8] F. Chatelin and T. Braconnier. About the qualitative computation of Jordan forms. *ZAMM*, 74:105–113, 1994.

[9] F. Chatelin, V. Frayssé, and T. Braconnier. Computations in the neighbourhood of algebraic singularities. *Num. Funct. Anal. Opt.*, 16:287–302, 1995.

[10] F. Collino. Private communication, 1995.

[11] V. Frayssé, Michel Gueury, Frank Nicoud, and V. Toumazou. Spectral portraits for matrix pencils. Tech. Rep. TR/PA/96/19, CERFACS, Toulouse, France, 1996.

[12] V. Frayssé and V. Toumazou. A note on the normwise perturbation theory for the regular generalized eigenproblem $Ax = \lambda Bx$. Technical Report TR/PA/96/18, CERFACS, 1996. To appear in J. Numer. Linear Algebra Appl.

[13] S. Gratton. On the condition number of linear least squares problems in Frobenius norm. *BIT*, 36:523–530, 1996.

[14] E. Hairer and Ch. Lubich. The life-span of backward error analysis for numerical integrators. *Numerische Mathematik*, 76:441–462, 1997.

[15] N. J. Higham. A collection of test matrices in MATLAB. *ACM Trans. Math. Software*, 17:289–305, September 1991.

[16] N. J. Higham. *Accuracy and Stability of Numerical Algorithms.* SIAM, Philadelphia, 1996.

[17] Y. Saad. Sparskit: a basic tool kit for sparse matrix computations. version2. June 1993.

[18] The MathWorks. *MATLAB Reference Guide.* The MathWorks Inc., Natick, MA, 1992.

[19] V. Toumazou. *Portraits spectraux de matrices: un outil d'analyse de la stabilité.* Ph.D. dissertation, Université H. Poincaré, Nancy, 1996.

[20] J. H. Wilkinson. Error analysis of floating point computation. *Numer. Math.*, 2:219–340, 1960.