Lund University

thesis submitted for the degree of bachelor of science

Centre for mathematical sciences

Strategies for computing the condition number of a matrix

Author:
Per Niklas Waaler

Supervisor:
Claus Führer

June 28, 2017

Abstract

The main objective of this thesis is to present the article *An estimate of the condition number of a matrix*, written by Cline et.al., which describes an algorithm for obtaining a reliable order of magnitude estimate of the condition number of a matrix in $\mathcal{O}(n^2)$ operations. In addition, the thesis introduces an iterative process for estimating the condition number in the 2-norm with arbitrary precision. It is a generalisation of the algorithm in the article, and it establishes a connection between power iteration and the algorithm from the article, and provides an alternative route to deriving its key equations. Also, the thesis covers basic theory surrounding the condition number, and for completeness it includes a brief presentation on Skeel's condition number which is based on the article *Scaling for numerical stability in Gaussian elimination*, written by Skeel et.al..

Acknowledgements

I would like to thank my thesis supervisor Claus for all his help and his insistence on understanding everything on a more intuitive level, and my friend Rasmus Høier for all his good advice and all his time spent reading my work to weed out errors.

Contents

Introduction

When solving linear systems of equations it is important to have an idea of how much accuracy we can expect in our computations. The condition number of a matrix provides a measurement of how sensitive the solution is to perturbations of the input data, and this leads to the desire of a cheap way to estimate it. In the article *An estimate of the condition number of a matrix*, the authors introduce an algorithm for obtaining an estimate of the condition number which reliably indicates its order of magnitude. For the purposes of many applications, this estimator offers a good balance between computational cost and accuracy, considering that it is often sufficient to know the order of magnitude of the condition number.

Chapter 1 is dedicated to presenting the relevant theory. We discuss topics such as the definition of the condition number of a problem, the significance of choosing an appropriate norm, and derivations of the condition number of a matrix. At the end of the chapter we briefly cover Skeel's condition number, which is rarely presented in basic textbooks on numerical linear algebra, but is frequently used in practice.

In Chapter 2, we consider ways of estimating the condition number of a matrix in an economical way. We derive the equations $A^{\mathrm{T}}x = b$ and $Ay = x$ which allows us to estimate the most costly part of the condition number, namely $\|A^{-1}\|$. We also introduce an iterative algorithm for estimating the condition number in the 2-norm which is based on power iteration. This algorithm is a generalisation of the algorithm from the article, and it provides an alternative way to derive the above equations. A benefit to having this iterative algorithm is that it provides additional freedom in finding a suitable trade-off between accuracy and reliability on one hand, and computational cost on the other.

In Chapter 3 we consider the task of choosing the right hand side in the equation $A^{\mathrm{T}}x = b$ in a way that ensures the success of the algorithm. Two main strategies are presented. One of these is very cheap and simple, but its simplicity comes at the cost of being somewhat risky, as there is a potential for it to fail completely. The second strategy is modified so that it is far more dependable, although more costly.

Finally, in chapter 4 we present the results from numerical testing of some different strategies by testing them on large numbers of randomly generated matrices. We test the algorithms using both QR and LU decomposition, and determine which strategy for picking an initial vector b is most successful under various circumstances.

Chapter 1

Theory of error sensitivity

Whenever we use a computer to estimate a solution of a problem given an input, we generally want to have some way of measuring how much uncertainty we can expect there to be in the output. The uncertainty can be due to uncertainty in the measurements that make up the input data, and it can be due to round-off errors which are introduced when we convert the data into the computers floating point number base [2]. The extent to which these errors gets amplified depends on the mathematical properties of the algorithm we use to compute an output. Since errors will be present to some extent in virtually all applications, it is therefore useful to have some way of measuring how sensitive a problem is to perturbations in the input data, which motivates the question which underlies the discussions of this chapter: if the input data is perturbed slightly, how much can it be expected to change the output data?

To make these ideas more explicit, we start by introducing some basic concepts. For instance, it is useful to specify what we mean when use the word *problem*. In our context, a problem is thought of as a map f from some normed vector space of inputs X to some normed vector space of outputs Y [3]. We have the option then to look at the behaviour of f globally, but sometimes we are more interested in the local behaviour of f. We then have a specific input x in mind and we want to know the behaviour of f in a neighbourhood of x, and so we consider the effect on the output when we perturb x slightly. When we have a particular x in mind combined with some problem f we sometimes call it a *problem instance*, but it is usually referred to simply as a problem.

We mentioned that the vector spaces are normed. A space being *normed* means that it is equipped with some notion of distance between elements of that space. Choosing a way to measure distance is central when considering issues relating to numerical stability, as the choice of norm will be used to determine what is large and small, and what is significant and insignificant. There are many ways to define distance, besides the familiar Euclidian norm, and which norm is appropriate to use depends on the problem. For instance, if we are solving the system $Ax = b$, where b is input data and x is output data, it might be tempting to just use the 2-norm as a measurement of the perturbation of b. But it might be the case that the various components of b are measurements of greatly varying magnitude, and that perturbing one component by some fixed amount represents a catastrophe in terms of the precision of that measurement(in physics for instance, the least precise measurement tends to be somewhat of a bottleneck in terms of precision), whereas perturbing another component by the same amount would not matter much at all if that measurement is of a comparatively large order of magnitude.

This consideration is not taken into account with the 2-norm, and so in some cases it is not a useful way to measure distance. In fact, if we perturb a component x_i by δ, and take the 2-norm of x we will have under the square-root sign the term $(x_i + \delta)^2 = x_i^2 + 2\delta \cdot x_i + \delta^2$. We can see that the amount by which the perturbation changes the length of x is determined by the sum $2\delta \cdot x_i + \delta^2$, and for positive x_i and δx_i the increase in length becomes greater when the perturbed component x_i is large. We see here a disagreement between distance as measured in the 2-norm and our intuitive notion of distance between two sets of measurements; intuitively, we think of an error of fixed size as making more of a difference when the error is in a measurement of small magnitude than when it is in a measurement of large magnitude, and we would like our definition of distance to reflect this intuition. Considerations such as these have lead to the formulation of Skeels condition number, which we will get to later.

1.1 The condition number of a problem

A concept which is of great importance when discussing matters concerning error sensitivity, and which will eventually lead us to the definition of the condition number of a matrix, is the *condition number* of a problem f, denoted κ. It comes in two forms, the absolute condition number and the relative condition number. The former deals with the absolute size of the perturbation. We shall only cover the relative condition number here, as it is more commonly used in numerical analysis [3], and hereafter it will be referred to simply as *the* condition number.

If f is a function from a normed vector space X to a normed vector space Y, and $\bar{x}$ represents the perturbed input so that $\bar{x} = x + \delta x$, then the condition number of f at x can be defined as

$$\kappa := \lim_{\delta \to 0} \sup_{\|\delta x\| \leq \delta} \frac{\text{"relative distance between } f(\bar{x}) \text{ and } f(x)\text{"}}{\text{"relative distance between } \bar{x} \text{ and } x\text{"}}, \tag{1.1}$$

see [2], where we assume that $\|\delta x\| \neq 0$, and where by relative we mean relative to $\|f(x)\|$ and $\|x\|$ respectively. To avoid cluttering up notation, we sometimes use $\|\cdot\|$ without specifying the norm by an index. The norm that is intended can be read from the context, so when we write $\|f(x)\|$, we mean $\|f(x)\|_{(Y)}$, hence the norm can be inferred by asking which space the element belongs to. Now, if we let $\delta f = f(\bar{x}) - f(x)$, then this can be expressed as

$$\kappa = \lim_{\delta \to 0} \sup_{\|\delta x\| \leq \delta} \frac{\|\delta f\|/\|f(x)\|}{\|\delta x\|/\|x\|} \tag{1.2}$$

With this definition at hand, we turn our attention to finding the condition numbers related to problems involving system of equations, which we will in turn use to define the condition number of a matrix. Note that there are several ways to view the equation $Ax = b$ in terms of what to consider as input and output. For instance, we can view x as output, and then consider the effect of perturbing both A and b simultaneously, or we can perturb just A or just b, which is sometimes done for simplicity. Often, b will contain measurements, and A will be a matrix of coefficients that reflects the physical laws or properties that characterize the physical system under observation, and it is therefore useful to consider perturbations to both or either of these.

As an example, when applying Kirchhoff's rules in order to determine how the currents will flow in an electrical circuit, we will obtain a system of equations equivalent to the matrix equation $Ax = b$, where the components of A are determined by the circuits resistances (and the directions of the currents in the circuit) as well as how the circuits are connected, b is determined by the voltages of the emf sources and how the circuits are connected, and x is the vector of unknown currents. In this case we can expect there to be uncertainty in the data that goes in to both b and A, and so ideally we want to consider the effect of introducing perturbations to both when we compute the condition number of the problem. Note that if we consider the problem in this way, the elements of the input space would be matrix-vector pairs, which raises the question of how to define distance between such objects. Note also that some of the elements of A and b will be exact zeros, and as such they should not be considered as measurements, which are susceptible to errors. This means that it would be inappropriate to include perturbations to these elements in our error-sensitivity analysis, as this yields an overly dramatic estimate of the error sensitivity(the reason that it becomes larger is that we are maximizing the relative perturbation over a larger set of input perturbations). The subjects of how to deal with exact zeros and how to define distance between matrix-vector pairs are addressed in the section on Skeel's condition number.

1.2 The induced matrix norm

Before we proceed further in discussing the condition of problems related to solving linear systems of equations, we will quickly go through the definition of the induced matrix norm, as it is necessary to know in order to understand the definition of the condition number of a matrix.

If A is an $m \times n$ matrix, and $\|\cdot\|_{(n)}$ and $\|\cdot\|_{(m)}$ are the norms of the domain and range of A respectively, then the induced matrix norm $\|A\|_{(m,n)}$ is defined as

$$\|A\|_{(m,n)} \coloneqq \sup_{x\in\mathbb{R}^n} \frac{\|Ax\|_{(m)}}{\|x\|_{(n)}} = \max_{\substack{x\in\mathbb{R}^n \\ \|x\|_{(n)}=1}} \|Ax\|_{(m)} \tag{1.3}$$

Intuitively, if we take on the view that A is an operator that operates on x, and then think of A as stretching or compressing x as it operates on it (also changing its direction of course, but in this definition only changes to length are relevant), one can think of this norm as the largest factor by which A can scale up the length of any vector x that lies in its domain.

1.3 The condition number of a matrix

Now we are ready to set an upper bound on the condition numbers of various problems relating to systems of equations, and these results will lead naturally to a definition of the condition number of a matrix. Deriving the upper bounds rather than the actual condition numbers is motivated by the fact that they are simpler, and therefore cheaper to compute than the actual condition numbers.

We simplify notation at this point by dropping the limit in the definition of the condition number, and simply consider the supremum over all infinitesimal perturbations of the input, similar to how we sometimes consider the derivative of a real valued function at x as the change in output value relative to the infinitesimal change in input value. We also assume that A is invertible, and that the norm of the output space is the same as the norm of the input space. If we consider first x as input with b as output and A held fixed (hence we only consider perturbations in b), and if δb is the perturbation that corresponds to a perturbation in input δx, we get

$$\begin{aligned} k =& \sup_{\delta x} \frac{\|\delta b\|/\|b\|}{\|\delta x\|/\|x\|} = \sup_{\delta x} \frac{\|\delta b\|}{\|\delta x\|}\frac{\|x\|}{\|b\|} = \\ & \sup_{\delta x} \frac{\|A\delta x\|}{\|\delta x\|}\frac{\|A^{-1}b\|}{\|b\|} \leq \|A\|\|A^{-1}\| \end{aligned}$$

Note that if we swap the roles of x and b and consider b as input and x as output, then we have essentially the same problem as before, with the roles of A and A^{-1} interchanged. Hence we would end up with the same bound.

This bound also turns up when we consider A as input and x as output, with b held fixed. Let δA be the perturbation in A, and let δx the resulting perturbation in x. We then have $(A + \delta A)(x + \delta x) = b$. Since the double infinitesimal $\delta A \delta x$ will become vanishingly small in size relative to the other terms for small perturbations, we drop it, and obtain $Ax + A\delta x + \delta A x = b$. After subtracting $b = Ax$ and multiplying by A^{-1} on each side, we get $\delta x = -A^{-1}\delta A x$. Taking the norm on each side yields $\|\delta x\| = \|A^{-1}\delta A\, x\| \leq \|A^{-1}\|\|\delta A x\| \leq \|A^{-1}\|\|\delta A\|\|x\|$ (where we use the fact that $\|Ax\| \leq \|A\|\|x\|$ twice). Finally, using this upper bound for $\|\delta x\|$, we compute

$$\sup_{\delta A} \frac{\|\delta x\|/\|x\|}{\|\delta A\|/\|A\|} \leq \frac{(\|A^{-1}\| \cdot \|\delta A\| \cdot \|x\|)/\|x\|}{\|\delta A\|/\|A\|} = \|A^{-1}\| \cdot \|A\|$$

Which is the same bound as before. Note that in this case the norm of the input space is the induced matrix norm.

Since this bound turns so often, it is defined as *the condition number of* A, denoted $\kappa(A)$. Note that it is independent of x and b, and it can be considered a property of A which gives us information about the tendency of A and A^{-1} to amplify errors in the vectors they operate on. In the following we will use the notation $\kappa(A)_{\text{l}}$, where l indicates the norm we are using.

Besides giving an estimate to the accuracy in our solution, $\kappa(A)$ can also be used as a way to detect potential errors that have been made. For instance, a matrix which is "nearly invertible" (imagine for instance taking a triangular matrix with a zero on the diagonal and perturbing the zero by a small amount) tends to be highly ill conditioned, and an extremely large condition number could be a sign that A in its pre-converted form (before having its entries converted to floating point numbers) was actually singular, but due to round-off errors introduced in the conversion it has become invertible.

1.4 Skeel's condition number

As useful as $\kappa(A)$ is, it has its drawbacks. In either of the derivations we have presented, note that we did not put up any restrictions on the perturbations of x or b other than their lengths. This is a rather unrealistic assumption, since in practice the matrices we are working with often have structural zeros, which are exact, and as such, they will not contain any measurement error. In addition, the conversion of input data to the computers floating point number base leads to errors that are of the same relative size as the datum they perturb [2], hence any perturbation to a structural zero is unrealistically large given these two considerations. Another issue with $\kappa(A)$ is the norm used to measure the size of perturbations; in the discussion on norms we touched on how the 2-norm (or the the 1-norm for that matter) does not always measure the size of a perturbation in an appropriate way, in particular when the various components are measurements that vary greatly in magnitude, and it is blind to the physical units that we are using. Considerations like these have led to the formulation of another condition number called Skeels condition number, denoted $\mathrm{Cond}(A, x)$, which we will cover here briefly.

In its derivation [2], the effect of perturbing both A and b simultaneously is considered, with x as output, and we use as our metric the *largest relative perturbation of the components of A and b*, i.e. the smallest number $\epsilon \geq 0$ s.t.

$$|\bar{a_{ij}} - a_{ij}| \leq \epsilon |a_{ij}|, \quad \forall\ i, j \tag{1.4}$$

and

$$|\bar{b_i} - b_i| \leq \epsilon |b_i|, \quad \forall\ i \tag{1.5}$$

Using this metric effectively restricts the set of perturbations over which we maximize the relative perturbation, since for any $\epsilon > 0$ the perturbation in an exact zero is bounded by 0. Also, it measures distance entirely in terms of relative component-wise error which means that it is meaningful regardless of the physical units of the measurement data. It also is a neat solution to the problem of how to measure distance between elements of the form (A, b).

We will not include the derivations of $\mathrm{Cond}(A)$ as it takes us beyond the scope of this thesis. By using definition 1.1 one can show that the condition number of the problem where A and b are inputs and x is output - using our newly defined measure of distance - is given by

$$\mathrm{Cond}(A, x) = \frac{\| |A^{-1}||A||x| + |A^{-1}||b| \|_\infty}{\|x\|_\infty} \tag{1.6}$$

where the notation $|\cdot|$ used on arrays signifies that the arrays are component-wise non-negative, but otherwise the same as the array inside the modulus sign. If we hold b fixed, then the we have

$$\mathrm{Cond}(A, x) = \frac{\| |A^{-1}||A||x| \|_\infty}{\|x\|_\infty}. \tag{1.7}$$

To simplify matters further, we can define Skeels condition number as the maximum over all $\|x\|_\infty = 1$. Since $|A^{-1}||A|$ has only non-negative components, the ratio is maximized by setting $x = (1, 1, ..., 1)^T$. Since $\| |A^{-1}||A|(1, 1, ..., 1)^T| \|_\infty = \| |A^{-1}||A| \|_\infty$, we get that

$$\mathrm{Cond}(A) = \| |A^{-1}||A| \|_\infty. \tag{1.8}$$

1.5 Difference in terms of scaling

Another important way in which the two condition numbers differ is in how they behave in terms of scaling. Scaling in this context means to multiply A by a diagonal matrix D from the left. As we will show, $\mathrm{Cond}(A)$ is invariant to scaling of A, whereas the difference in $\kappa(A)$ can in some cases be made arbitrarily large by scaling A. This is a considerable disagreement between the two estimates of stability, and we may wonder whether or not invariance to scaling is a desirable property, and why the two condition numbers respond so differently to scaling.

That $\mathrm{Cond}(A)$ is invariant to scaling is clear from its definition: if we scale A by a diagonal matrix D, we get that $\mathrm{Cond}(DA) = \||(DA)^{-1}||DA|\| = \||A^{-1}||A|\| = \mathrm{Cond}(A)$(multiplying a matrix from the left by D scales row i by d_i, and multiplying A^{-1} from the right by D^{-1} scales column j by $1/d_j$, hence these scaling factors will cancel out upon multiplication). The fact that $\kappa(A)$ is not invariant to scaling can be illustrated by scaling row 3 of the 3×3 identity matrix by a factor ϵ, resulting in the matrix

$$A' = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \epsilon \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \epsilon \end{pmatrix} \tag{1.9}$$

Let us compute $\kappa(A')_2$. To compute $\|A'\|_2$, we find the vector x of length 1 that maximizes $\|A'x\|_2$. This is a straight-forward maximization problem subject to the constraint $x_1^2 + x_2^2 + x_3^2 = 1$, and if we solve it we find that a maximizer is $x = (\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, 0)^T$, hence $\|A'\|_2 = \|x\|_2/\|b\|_2 = 1$. By the same method we also find that $\|(A')^{-1}\|_2 = 1/\epsilon$. Hence $\kappa(A)_2 = 1 \cdot 1/\epsilon = 1/\epsilon$, which can be made arbitrarily large by choosing ϵ sufficiently small. This number is a reflection of the fact that - given that we measure the sizes of the perturbations in the 2-norm - the system $Ax = b$ is highly sensitive to perturbations in A_{33}, x_3 and b_3 due to the small size of ϵ, provided that we place no restrictions on the direction (that is, the size of the components relative to each other) of the perturbation vectors we consider.

So how do we account for the fact that the two estimates vary so dramatically in this regard? This has to do with the different ways in which distance is measured in each condition number. In the 2-norm, if we perturb a small component with a small perturbation δx_i (here, small means small relative to $\|x\|_2$) then $\|\delta x\|_2$ will be small, as we have touched on before, and so the perturbation will be measured as being small. Hence we get a large perturbation of output resulting from a small perturbation of input, and are therefore given the impression that the system is sensitive to perturbations. However, if we measure the size of the same perturbation using the metric of *relative component-wise perturbation*(as we do in Skeel's condition number), the same perturbation will be measured as being very large if the perturbation is large relative to x_i. Consequently, the two norms can give very different impressions of the perturbation sensitivity of

A. Another reason for the disagreement is that in $\kappa(A)$ we are maximizing the relative perturbation over a larger set of perturbations, since in $\mathrm{Cond}(A)$ we are effectively placing the restraint that perturbation of exact zeros are not allowed.

Chapter 2

Estimating the condition number of A

In this chapter we cover the main subject of this thesis, which is the question of how to estimate $\kappa(A)$. Given that $\kappa(A) = \|A\|\|A^{-1}\|$, our efforts to estimate $\kappa(A)$ can be broken down into the task of estimating $\|A\|$ and $\|A^{-1}\|$. Computing $\|A\|$ in the 1-norm or infinity norm is particularly simple as it is just a matter of finding the column vector (in the case of the 1-norm) or row vector (in the case of the ∞ norm) with the largest 1-norm. We do not know A^{-1} however, and although computing it would yield an exact estimate (ignoring round-off errors), this would be too expensive to be worthwhile as it is a task that requires $\mathcal{O}(n^3)$ operations, especially considering that in many applications only an *order of magnitude* estimate is required.

The ease with which we can compute $\|A\|_1$ makes it tempting to compute the condition number in the 1-norm. It is not immediately obvious how to estimate $\|A^{-1}\|_1$ however. The approach presented in *The condition number of a matrix* is in essence to construct a right hand side in $Ax = b$ in a way that tends to maximize the ratio $\|x\|/\|b\|$, which then yields a lower bound estimate as we see from

$$\|A^{-1}\| \geq \frac{\|A^{-1}b\|}{\|b\|} = \frac{\|x\|}{\|b\|}. \tag{2.1}$$

Finding a strategy for maximizing this ratio appears to be easier to do in the 2-norm, since they are related to the *singular value composition* of A, abbreviated to SVD; a factorization we give a brief introduction to later. Due to the simplicity offered by working with the 2-norm, one approach to obtain a 1-norm estimate is to find a vector pair b and x which maximizes equation 2.1 in the 2-norm, and then simply compute the quotient in terms of the 1-norm instead.

This - as one might expect, and which we will show examples of later - leads to a less accurate estimate, but one which also seems reliable in the sense that it reliably indicates the correct order of magnitude of the condition number.

2.1 Analysis based on the SVD of A

Now that we have an overarching idea on how to to estimate $\|A^{-1}\|_1$, we proceed to do a more in-depth analysis by using the SVD of A. This factorization is highly useful in that it helps us investigate under what conditions $\|x\|_2/\|b\|_2$ is maximized. In the SVD, a matrix A is factorized into the product $U\Sigma V^T$, where U and V are orthogonal with columns denoted u_i and v_i respectively, and where Σ is a diagonal matrix. The values on the diagonal of Σ are called singular values of A and are denoted σ_i. By convention they are always non-negative, and they are ordered according to size, with the largest, σ_1, sitting at the upper end of the diagonal. The reason why the SVD is so useful to us is due to the fact that $\sigma_1 = \|A\|_2$ and $\sigma_n^{-1} = \|A^{-1}\|_2$. The latter equation follows from the first since $A^{-1} = V\Sigma^{-1}U^{\mathrm{T}}$, from which we can see that the largest singular value of A^{-1} is σ_n^{-1}(since σ_n is the smallest singular value of A). Another useful property is revealed by $A^{\mathrm{T}} = V\Sigma U^{\mathrm{T}}$, which shows us that A and A^{T} have the same singular values and hence the same condition number.

There is a neat geometrical interpretation which might be helpful for gaining an intuitive understanding of why the SVD is related to $\kappa(A)_2$: any $m \times n$ matrix A maps the unit sphere in $\mathbb{R}^m$ to a hyper-ellipse, and the singular values are the lengths of its semi-axes [3]. As an example, imagine that we transform the unit circle in $\mathbb{R}^2$ via a 2×2 matrix C with a very large condition number in the 2-norm. The fact that $\kappa(A)_2$ is large means that $\sigma_1 >> \sigma_n$, hence the unit circle gets mapped to a highly eccentric ellipse. A consequence of this is that two vectors which are close to each other on the unit circle could be mapped to two vectors of very different lengths, hence the sensitivity to perturbations.

With these facts at hand, we turn our attention again to the ratio $\|x\|/\|b\|$. Since what we are looking for, σ_1 and σ_n^{-1}, are linked so intimately to the SVD of A, a promising place to start our analysis is to express A in terms of its SVD, and then expand b in terms of the basis $\{u_i\}$ or $\{v_i\}$(it will become obvious which choice is suitable). Now, note that

$$Ax = U\Sigma V^{\mathrm{T}}x = b \tag{2.2}$$

from which we see that

$$x = V\Sigma^{-1}U^{\mathrm{T}}b. \tag{2.3}$$

Observe that the multiplication U^Tb yields a vector where the elements are formed by computing the inner products $u_i^T b$, which suggests that matters are simplified by expressing b in in terms of the basis $\{u_i\}$, since $u_i^{\mathrm{T}} u_j$ equals 1 if $i = j$, and 0 otherwise. So we let

$$b = \|b\| \sum \alpha_i u_i, \qquad \sum \alpha_i^2 = 1. \tag{2.4}$$

Inserting this expression for b into equation 2.3, and letting e_i denote the unit vector where the *i'th* component equals 1 and all other components equal 0, we obtain

$$\begin{aligned} x &= V\Sigma^{-1}U^{\mathrm{T}}\|b\| \sum \alpha_i u_i = \|b\| V\Sigma^{-1} \sum \alpha_i e_i \\ &= \|b\| V \sum \frac{\alpha_i}{\sigma_i} e_i = \|b\| \sum \frac{\alpha_i}{\sigma_i} v_i \end{aligned} \tag{2.5}$$

We can now express the ratio 2.1 in a more revealing way:

$$\frac{\|x\|}{\|b\|} = \frac{\|b\|\sqrt{\sum(\alpha_i/\sigma_i)^2}}{\|b\|\sqrt{\sum \alpha_i^2}} = \sqrt{\sum(\alpha_i/\sigma_i)^2} \tag{2.6}$$

This expression is clearly maximized when all the weight is given to the term with the largest coefficient $1/\sigma_n$, i.e. when $\alpha_n = 1$ and $\alpha_i = 0$ for $i \neq n$. From equation 2.6 it seems plausible (assuming that σ_i are randomly chosen) that $\|x\|/\|b\|$ is of the order $\|A^{-1}\|_2$, unless we get unlucky and α_n is particularly small. Note also that the probability of the ratio indicating the right order of magnitude increases when σ_n is very small relative to the other σ_i, which implies that A is ill-conditioned.

The most important information we gather from equation 2.6, howerever, is that $\|x\|/\|b\|$ provides a good estimate when u_n is well represented in the right hand side b of our equation; that is, when u_n has a coefficient that is large relative to the coefficients of all the other u_i. A natural next step then is to try to construct a b where u_n is well represented.

Taking a look at x as expressed in equation 2.6, note that v_n is likely to be well represented in x due to the presence of the σ_i in the denominators since $\sigma_n^{-1} > \sigma_i^{-1}$ for $i \neq n$. It is tempting to exploit this amplification of v_n by somehow using x as a new right hand side. The only complication is that it is v_n, not u_n, that is scaled up. Adjusting for this turns out to be quite simple however. To see how, notice that if we were to carry out the above SVD analysis using A^{T} instead of A (recall that both matrices have the same condition number), nothing would change fundamentally; the only difference would be that the roles of U and V (and thus also the roles of the vectors u_i

and v_i) are interchanged, since $A^{\mathrm{T}} = V\Sigma U^{\mathrm{T}}$. Hence the x that we obtain in equation 2.3 would be a suitable right hand side of the system $A^{\mathrm{T}}y = x$.

We can now summarize the strategy for estimating σ_n^{-1} and $\|A^{-1}\|_1$: first solve $Ax = b$ for x, where x is then tailored to be a suitable right hand side for the system $A^{\mathrm{T}}y = x$, then proceed to solve for y, and finally compute the ratio $\|y\|/\|x\|$ which will be the estimate of either $\|A^{-1}\|_1$ or $\|A^{-1}\|_2$ depending on which norm we use. If we want a 1-norm ratio we compute $(\|x\|_1/\|b\|_1) \cdot \|A\|_1$.

Note that computing the analytical expression of the ratio $\|y\|/\|x\|$ is completely analogous to how it is done for $\|x\|/\|b\|$. Since (α_i/σ_i) are the coefficients in the right singular vector expansion of x just like α_i are the coefficients in the right singular vector expansion of b, it follows by analogy that we only need to substitute α_i with α_i/σ_i in the expression for $\|x\|/\|b\|$, and thus we get that $\|y\|/\|x\| = \sqrt{\sum(\alpha_i/\sigma_i^2)^2}$. Observe also that there is nothing preventing us from first solving $A^T x = b$ and then solving $Ay = x$; this merely amounts to swapping the roles U and V in the preceding arguments, and so the steps are analogous.

2.2 The power iteration approach

It turns our that there is another approach we may take to arrive at the equations $A^{\mathrm{T}}x = b$ and $Ay = x$. This approach is based on an algorithm called power iteration, in fact the algorithm we will develop will essentially be power iteration with the addition of what can be viewed as *in between* steps, and the estimator derived previously will turn out to be the result of running one iteration of this algorithm. This method can be used to obtain an estimate of σ_n^{-1} as well as σ_1, and so this algorithm can be used to obtain a 2-norm estimate of $\kappa(A)$.

Power iteration is a very simple iterative process for estimating the largest eigenvalue of a matrix B and its corresponding eigenvector. It essentially involves taking some vector z_0 and repeatedly multiplying it from the left by B (in practice we normalize at the end of each iteration for stability). The sequence of vectors $\{z_k\}$ will then converge to an eigenvector, provided that its two largest eigenvalues are not equally large. If we instead wish to find the smallest eigenvalue of B, we can perform *inverse power iteration* where we repeatedly multiply by B^{-1} instead of B. The eigenvalue estimate at each step is computed by taking the Rayleigh quotient

$$r(A, z_k) = \frac{z_k^{\mathrm{T}} B z_k}{z_k^{\mathrm{T}} z_k} \tag{2.7}$$

which is an eigenvalue estimator that approximates the eigenvalue λ_i when the corresponding eigenvector q_i is dominant in z_k [3].

To get a sense how $\kappa(A)_2$ and eigenvalues are connected, we can notice a similarity between eigenvalues and singular values; both can be found on the diagonal of a diagonal matrix which is located between two orthogonal matrices in a factorization of A. This similarity is due to the fact that the SVD is a generalization of eigenvalue decomposition so that it can be applied to all matrices, not just symmetric ones. In fact, when A is symmetric, its eigenvalues *are* its singular values. The problem is that A is not always going to be symmetric. However, the matrices $A^{\mathrm{T}}A$ and AA^{T} are symmetric, and we will show next that σ_i^2 are their eigenvalues. This means that we can compute both σ_1 and σ_n^{-1} by performing power iteration or inverse power iteration on either of these matrices.

For now we will focus the estimation of σ_n^{-1}, and then return to the estimation of σ_1 at the end of the chapter. To this end, we start out by writing out the following expressions:

$$(A^{\mathrm{T}}A)^{-1} = (V\Sigma U^{\mathrm{T}}U\Sigma V^{\mathrm{T}})^{-1} = V\Sigma^{-2}V^{\mathrm{T}} \tag{2.8}$$

$$(AA^{\mathrm{T}})^{-1} = (U\Sigma V^{\mathrm{T}}V\Sigma U^{\mathrm{T}})^{-1} = U\Sigma^{-2}U^{\mathrm{T}}. \tag{2.9}$$

The diagonal elements of Σ^{-2} are σ_i^{-2}. We know from elementary linear algebra that if we can express a matrix B on the form $B = QDQ^T$ - where D is diagonal and Q is orthogonal - then the column vectors of Q are the eigenvectors of B, and the diagonal elements of D are the corresponding eigenvalues. Therefore, we can see in the above equations that $(A^{\mathrm{T}}A)^{-1}$ and $(AA^{\mathrm{T}})^{-1}$ both have eigenvalues σ_i^{-2}, and eigenvectors v_i and u_i respectively.

We first use the same notation that we used earlier, with the vectors x, y and b, in order to make it clear that we end up with the same equations. Later we adopt new notation which is more convenient for expressing the entire algorithm. So, let b be the initial vector to start of the power iteration where we apply the matrix $(A^TA)^{-1}$. In the first step we solve the equation

$$y = (A^{\mathrm{T}}A)^{-1}b \ \Leftrightarrow A^{\mathrm{T}}(Ay) = b. \tag{2.10}$$

If we let $x = Ay$, then we first solve $A^{\mathrm{T}}x = b$, and then proceed to solve $Ay = x$ for y. At this point there are two estimates we may compute, which is made clear when we express y and x in terms of the singular vectors of A. Letting $b = \sum \alpha_i v_i$, we get that

$$y = (A^{\mathrm{T}}A)^{-1}b = V\Sigma^{-2}V^{\mathrm{T}} \cdot \sum \alpha_i v_i = \sum \frac{\alpha_i}{\sigma_i^2} v_i \tag{2.11}$$

and

$$x = Ay = U\Sigma V^{\mathrm{T}} \cdot \sum \frac{\alpha_i}{\sigma_i^2} v_i = \sum \frac{\alpha_i}{\sigma_i} u_i. \tag{2.12}$$

In the expression for y the presence of σ_i^2 in the denominators increases the probability that v_n will dominate the expression, which is also the eigenvector of $(A^{\mathrm{T}}A)^{-1}$ corresponding to the eigenvalue σ_n^{-2}. Hence we are likely to obtain a good estimate of σ_n^{-2} from Rayleigh quotient

$$\begin{aligned} r((A^{\mathrm{T}}A)^{-1}, y) &= \frac{y^{\mathrm{T}}(A^{\mathrm{T}}A)^{-1}y}{y^{\mathrm{T}}y} = \frac{y^{\mathrm{T}}A^{-1}A^{-\mathrm{T}}y}{y^{\mathrm{T}}y} = \frac{(A^{-\mathrm{T}}y)^{\mathrm{T}}A^{-\mathrm{T}}y}{y^{\mathrm{T}}y} \\ &= \frac{z^{\mathrm{T}}z}{y^{\mathrm{T}}y} = \frac{\|z\|^2}{\|y\|^2} \end{aligned} \tag{2.13}$$

where we have set $A^{\mathrm{T}}z = y$. This introduces a new system to solve, and if we wish to avoid the additional computation, equation 2.12 indicates that we could instead settle for a Rayleigh quotient involving x for a cheaper but less accurate estimate. Now it is u_n - an eigenvector of $(AA^T)^{-1}$ - that is likely to be well represented, and so we get an estimate of σ_n^{-2} from the Rayleigh quotient

$$r((AA^{\mathrm{T}})^{-1}, x) = \frac{x^{\mathrm{T}}(AA^{\mathrm{T}})^{-1}x}{x^{\mathrm{T}}x} = \frac{(A^{-1}x)^{\mathrm{T}}A^{-1}x}{x^{\mathrm{T}}x} = \frac{y^{\mathrm{T}}y}{x^{\mathrm{T}}x} = \frac{\|y\|^2}{\|x\|^2}. \tag{2.14}$$

Obtaining the cheaper of the two estimates is then a matter of solving $A^{\mathrm{T}}x = b$ and $Ay = x$, and then computing $\|y\|/\|x\|$, which are the same equations that we encountered in the previous section.

Now, we could keep going to obtain a finer estimate. To this end, we will change notation, and present the first few steps in order to make the pattern clear. Using the notation $Ay_k = y_{k-1}$ when k is even, and $A^{\mathrm{T}}y_k = y_{k-1}$ when k is odd, we get

$$y_2 = (A^{\mathrm{T}}A)^{-1}y_0 \;\Leftrightarrow\; A^{\mathrm{T}}Ay_2 = y_0 \;\Leftrightarrow\; A^{\mathrm{T}}y_1 = y_0 \tag{2.15}$$

and so we solve the equations $A^{\mathrm{T}}y_1 = y_0$ and then $Ay_2 = y_1$. In the next iteration, we similarly have

$$y_4 = (A^{\mathrm{T}}A)^{-1}y_2 \;\Leftrightarrow\; A^{\mathrm{T}}Ay_4 = y_2 \tag{2.16}$$

from which we obtain the equations $A^Ty_3 = y_2$ and $Ay_4 = y_3$. Continuing in this fashion, we get the following sequence of equations

$$\begin{aligned}
A^{\mathrm{T}} y_1 &= y_0 \\
A_2^{\mathrm{T}} &= y_1 \\
A^{\mathrm{T}} y_3 &= y_2 \\
A y_4 &= y_3 \\
&\vdots \\
A y_{2k} &= y_{2k-1} \\
A^{\mathrm{T}} y_{2k+1} &= y_{2k}
\end{aligned}$$

The vector y_k is then obtained from y_{k-1} via the recurrence relation $y_k = A^{-\mathrm{T}} y_{k-1}$(with k referring to the iteration we are in, so starting at $k = 1$) when k is odd, and $y_k = A^{-1} y_{k-1}$ when k is even. To make clear which Rayleigh quotients to use for each y_k, we observe that

$$\begin{aligned}
y_1 &= A^{-\mathrm{T}} y_0 = U\Sigma^{-1}V^{\mathrm{T}} \cdot \Sigma\alpha_i v_i = \Sigma\frac{\alpha_i}{\sigma_i} u_i \\
y_2 &= A^{-1} y_1 = V\Sigma^{-1}U^{\mathrm{T}} \cdot \Sigma\frac{\alpha_i}{\sigma_i} u_i = \Sigma\frac{\alpha_i}{\sigma_i^2} v_i \\
y_3 &= A^{-\mathrm{T}} y_2 = U\Sigma^{-1}V^{\mathrm{T}} \cdot \Sigma\frac{\alpha_i}{\sigma_i^2} v_i = \Sigma\frac{\alpha_i}{\sigma_i^3} u_i \\
&\vdots \\
y_{2k} &= \Sigma\frac{\alpha_i}{\sigma_i^{2k}} v_i \\
y_{2k+1} &= \Sigma\frac{\alpha_i}{\sigma_i^{2k+1}} u_i
\end{aligned}$$

From this we gather that the Rayleigh quotients when k is odd has the form

$$\frac{y_k^T (AA^{\mathrm{T}})^{-1} y_k}{y_k^{\mathrm{T}} y_k} = \frac{y_k^{\mathrm{T}} A^{-\mathrm{T}} A^{-1} y_k}{y_k^{\mathrm{T}} y_k} = \frac{(A^{-1} y_k)^{\mathrm{T}} A^{-1} y_k}{y_k^{\mathrm{T}} y_k} = \frac{\|y_{k+1}\|^2}{\|y_k\|^2}$$

and similarly when k is even we get

$$\frac{y_k^{\mathrm{T}} (A^{\mathrm{T}} A)^{-1} y_k}{y_k^{\mathrm{T}} y_k} = \frac{y_k^{\mathrm{T}} A^{-1} A^{-\mathrm{T}} y_k}{y_k^{\mathrm{T}} y_k} = \frac{(A^{-\mathrm{T}} y_k)^{\mathrm{T}} A^{-\mathrm{T}} y_k}{y_k^{\mathrm{T}} y_k} = \frac{\|y_{k+1}\|^2}{\|y_k\|^2}$$

Hence the estimate at step k is obtained by computing $\frac{\|y_k\|}{\|y_{k-1}\|}$.

If we instead want to estimate σ_1, we start out the iteration with the equation $y_2 = A^{\mathrm{T}}Ay_0$. The arguments leading to the sequence of equations that follow from this are analogous to those in the previous derivation, and if we fill out the details we obtain the formulas

$$\begin{aligned} y_k &= A^{\mathrm{T}}y_{k-1}, && k \text{ even} \\ y_k &= Ay_{k-1}, && k \text{ odd} \end{aligned}$$

As before the estimations of σ_1 at step k are obtained by computing $\|y_k\|/\|y_{k-1}\|$. Note that this process fails if y_0 has no component along v_n, that is if $y_0^T v_n = 0$. Letting y_0 be a random vector is a simple way to ensure that the probability for this happening is virtually zero. This does not mean however that we can not be unfortunate and get a y_0 which is *nearly* orthogonal to v_n, which would result in slow convergence. In Chapter 3 we discuss ways to choose y_0 in such a way as to prevent this kind of worst case scenario from happening.

In the following we will refer to this method for obtaining estimates of σ_1 or σ_n^{-1} as the *power iteration algorithm*, abbreviated to PIA.

2.3 Convergence of the power iteration algorithm

If we are going to use PIA for estimating $\kappa(A)$, it is important to investigate how quickly we can expect it to return an estimate of acceptable accuracy. Unfortunately, a well known disadvantage of power iteration is that its rate of convergence can be quite slow. If λ_1 and λ_2 are the largest and second largest eigenvalues of A respectively then

$$|\lambda_k - \lambda_1| = \mathcal{O}((|\lambda_2|/|\lambda_1|)^{2k}), \tag{2.17}$$

see [3]. Keep in mind that this is referring to the normal power iteration algorithm as opposed to the algorithm we just derived, and the increase in accuracy with each iteration k that equation 2.17 indicates is for us only obtained with every second iteration we perform.

It might be tempting to use the Rayleigh quotient at each step in order to make shifts to speed up the rate of convergence, as is done in Rayleigh quotient iteration. The problem with this approach is that it ruins the simplicity of the algorithm. In practice - as we will discuss later - factorization of A into a QR or LU decomposition is done simultaneously as we compute the condition number, which reduces most of the computation down to solving triangular systems of equations. Shifting by a constant μ would then result in that matrix no longer being triangular, which means that the algorithm would require $O(n^3)$ operations, and here we are looking specifically for algorithms that require $O(n^2)$ operations. If accuracy was essential however, it might be preferable to use this approach instead, given that Rayleigh quotient iteration is much faster; it triples the number of digits of accuracy with each iteration [3].

Although the rate of convergence is slow for power iteration, the PIA is sped up by the fact that the eigenvalues in our case are σ_i^2 or $1/\sigma_i^2$, which means that the inaccuracy of the estimate at each even-numbered step k is proportional to $(|\sigma_2|/|\sigma_1|)^{4k}$ or $(|\sigma_n|/|\sigma_{n-1}|)^{4k}$ respectively. Hence, if we are estimating σ_1, then adding two steps of iteration reduces the error by a constant factor $\approx (|\sigma_2|/|\sigma_1|)^4$. Also, when A is ill conditioned these ratios will tend to be larger, which further speeds up the rate of convergence. The fact that the convergence is slow for well conditioned matrices is not a big issue. The worst thing that can happen if $\kappa(A)$ is small is that we underestimate it by a large factor(since we are computing a lower bound), but this is not catastrophic since the matrix is well conditioned anyway. It would be problematic however if there was a substantial risk of significantly underestimating $\kappa(A)$ in cases where it is very large, but the likelihood of this becomes smaller the more ill-conditioned A is, which is a reassuring.

Chapter 3

Factorization of A and a strategic choice of b

The computational cost associated with a strategy for computing $\kappa(A)$ is an important aspect to consider when deciding how to estimate it. In practice - since $\kappa(A)$ is normally computed in relation to some system (or systems) that is to be solved - it is sensible to combine computation of $\kappa(A)$ with either QR or LU factorization of A. We would need to perform such factorization in any case, and so we might as well take advantage of it when estimating $\kappa(A)$. In terms of estimating $\kappa(A)$, QR factorization is particularly appealing, since we then only have to work with one of the factors, namely the upper triangular matrix R. This is because R has the same condition number as A in the 2-norm, as becomes apparent when we look at the SVD of R: $R = Q^{\mathrm{T}}A = (Q^{\mathrm{T}}U)\Sigma V^{\mathrm{T}}$. In the following 2 sections we will assume that a QR factorization is available to us for simplicity.

3.1 A local strategy for picking b

Besides from using factorizations to improve efficiency, we also have the opportunity to speed up the rate of convergence by making a good choice of initial vector $b = y_0$. We should expect an algorithm which does this to be very simple if it is going to be worthwhile, since it might otherwise be better to simply generate its components randomly, and then add an extra iteration in the PIA. Given the emphasis on simplicity, an appealing idea is to compute the components of b successively as we solve $R^{\mathrm{T}}x = b$. In each equation we solve for one component x_i, and in that equation only one component of b, b_i, is present. We then have the freedom to choose b_i in such a way that it promotes growth in $\|x\|_2$. In order to keep the algorithm cheap, we restrict our choice of each

b_i to two choices: $+1$ and -1. When solving the i'th equation, we then have $r_{ii}x_i = -r_{1i}x_1 - r_{2i}x_2 - ... - r_{i-1,i}x_{i-1} \pm 1$. A simple way to make this choice is to compute x_i for each choice, which we denote x_i^+ and x_i^-, and choose whichever sign maximizes $|x_i|$. In the following this strategy will be denoted as the *local strategy*, abbreviated to LS.

A drawback of this strategy is that it is blind to the effect that a choice will have on subsequent equations. If we are dealing with randomly generated matrices, this is very unlikely to cause severe issues. As an example, if the elements of R are random numbers that are uniformly distributed over the interval $(-1, 1)$, then the larger the modulus of each x_k, $1 \leq k < j$, the greater the likelihood that x_j will be large, since we see from

$$x_j = \sum_{k=1}^{j-1} \frac{-x_k}{b_j} \cdot r_{ij} + 1 \tag{3.1}$$

that the first $j-1$ terms of the sum are independent random numbers each uniformly distributed on an interval $(-\frac{|x_k|}{|b_j|}, \frac{|x_k|}{|b_j|})$. The wider these intervals are, the more likely that the modulus of the sum is large, since its variance then increases and its probability density function becomes more spread out. This demonstrates that the strategy, given a random matrix, will be successful on average, and has virtually zero probability to generate a "worst case b", i.e. a b which is orthogonal to v_n.

The problem, however, is that in practice matrices are not generally random. Often they have a particular structure, and this structure could be such that our strategy fails completely. To demonstrate this, consider the following example [1]. Let

$$R^{\mathrm{T}} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ k & -k & 1 & 0 \\ -k & k & 0 & 1 \end{pmatrix} \tag{3.2}$$

When computing x_1 and x_2, our strategy so far offers no way to determine the sign of b_1 and b_2. Let us arbitrarily decide that the default choice is the positive sign, in which case $x_1 = 1$ and $x_2 = 1$. But then it becomes immediately clear that k will not appear in equations 3 or 4 due to cancellation, and hence will have no influence on the estimate of the condition number. This is a problem, as we see when we consider how ill conditioned R is when k is large: $\|R\|_\infty = \|R^{\mathrm{T}}\|_\infty = 1 + 2k$.

The situation can be somewhat remedied by generating at each step a random number θ between 0.5 and 1, and choosing between $\pm\theta$ instead of ± 1. Then the probability of getting the kind of exact cancellation of terms as in the above example becomes practically zero, though we can still run into situations where

v_n is poorly represented, which would correspond to the terms nearly cancelling out. Note, however, if k is large we can still be confident that this randomized version of the strategy will work well, since we only need to avoid exact cancellation in order to ensure fast convergence in the cases where A is ill-conditioned. In the following this modified version of LS will be denoted RLS, as in "randomized local strategy".

3.2 A look ahead strategy for choosing b

As was mentioned earlier, a weakness of the strategy discussed so far is that it is entirely local, which is why we ran into trouble in the previous example. A natural next step then, is to find a strategy that has some kind of "look ahead" feature built into it. At step i, we cannot know which choice of sign will be optimal, one reason being that we cannot at that point compute $r_{jj}x_j = r_{1j}x_1 + ... + r_{j-1,j}x_{j-1} + b_j$ for $i < j \leq n$, since we do not know the value of x_j for $j > i$. But if we are less ambitious, we can make the choice of b_i which tends to maximize the part of each such sum (in terms of its absolute value) that we *can* compute, namely the sum of the first i terms: $r_{1j}x_1 + ... + r_{ij}x_i$. We can then proceed to make our choice of b_i according to whichever choice maximizes the sum

$$|\sum_{k=1}^{i-1} r_{ki}x_k + b_i| + \sum_{i+1}^{n} |\sum_{k=1}^{i} r_{kj}x_k|. \tag{3.3}$$

The first part of this expression ensures that the effect on $|r_{ii}x_i|$ is taken into account, and the second part of the expression takes into account the effect on the remaining $n-i$ equations. With this strategy, we will avoid the unfortunate scenario we encountered in our example regardless of how we choose the sign for the k's in R^{T} since - if k is large enough - the algorithm will choose the sign which does not lead to the cancellation of the terms involving k.

This strategy is more costly than the first one. In the first strategy, the cost is approximately the same as when we solve the system for a given b; for each row we are adding one extra multiplication and addition to the workload when we compute x_i^+ and x_i^-, and for large n, this cost is negligible compared to the cost of computing the first $i-1$ terms in $\sum_{k=1}^{i-1} r_{ki}x_k$ for each x_i. So for large n, the cost of the first strategy is approximately n^2 flops, about the same as the cost of solving a triangular system of equations in a regular way.

So far we have been discussing how to choose b in a way so that we get a good estimate of σ_n^{-1}, but the ideas that we have discussed applies also when computing $\|A\|_2$, since the success of the PIA relies on u_1 being well represented, which will be the case when x is large. The difference in how to apply of the growth-promoting strategies in each case is a matter of details.

Let us now compute the cost of the new strategy. Consider the i'th iteration where we compute x_i. The bulk of the workload lies in computing $\sum_{k=1}^{i} r_{kj}x_k$ for all $i < j \leq n$, so we will do $n - i$ such computations. But since $r_{1j}x_1 + r_{2j}x_2 + ... + r_{i-1,j}x_{i-1}$ was computed in the previous iteration for each j, we only count the cost associated with adding the new term $r_{ij}x_i$ to the sum for each j. For each computed value x_i^+ and x_i^- we will perform two operations (1 multiplication and 1 addition), so 4 operations in total for each j. And so, the number of flops in iteration i is $\sim 4(n - i)$. Summing together, we get

$$\sum_{i=1}^{n} 4(n-i) = 4\{n^2 - \frac{n(n+1)}{2}\} = 4(n^2 - \frac{n^2}{2} - \frac{n}{2}) = 2n^2 - 2n \tag{3.4}$$

The factor by which the cost goes up with the "look ahead strategy", which in the following will be denoted LAS, can now be computed. Ignoring the first order term, we get that $\frac{2n^2}{n^2} = 2$, and so this strategy is roughly twice as expensive as the LS. If we are using the QR decomposition and perform two iterations to obtain an estimate, the overall increase in work is by a factor $\frac{2n^2+n^2}{n^2+n^2} = \frac{3n^2}{2n^2} = 3/2 = 1.5$. At this point we may note that the cost of choosing b by the LAS is about the same as the cost of an extra iteration, and so we may wonder if we gain more accuracy by using the cheaper RLS for choosing b, and then use the work we save in doing so to perform an extra iteration in the PIA. We return to this speculation in Chapter 3, where we test it numerically.

3.3 Choice b when using LU factorization

Before we get to how to choose b when we have an LU factorization, where L is lower triangular and U is upper triangular, we consider which equations we must solve now that we have a different factorization. The fact that we have a different factorization of A does not change our overall strategy; as before, we may obtain an estimate by solving the systems $A^{\mathrm{T}}x = b$ and $Ay = x$, and then compute $\|y\|/\|x\|$. Concerning the first equation, we have $(LU)^T x = b \Leftrightarrow U^{\mathrm{T}}L^T x = b$. So, letting $z = L^T x$, we solve

$$U^{\mathrm{T}}z = b \tag{3.5}$$
$$L^{\mathrm{T}}x = z \tag{3.6}$$

to obtain x. Next we have $Ay = x \Leftrightarrow LUy = x$. Letting $Uy = w$, we solve

$$Lw = x \tag{3.7}$$
$$Uy = w \tag{3.8}$$

After which we can compute $\|y\|/\|x\|$.

It is less obvious how to go about choosing b in this case than it was when we had an QR factorization, since we now have two factors instead of one. The overall objective is still the same; we are looking to find a strategy for choosing the components of b in such a way so as to promote growth in the size of x. The strategy suggested in the article is to choose b such that z becomes large in equation 3.6, and hope that this will result in a large x. However, as of yet we have no assurance that this will be the case. In the QR case this was not an issue, since the choices of b_i were designed to ensure large components of x, or atleast avoid the worst case scenario where b is orthogonal to v_n, or nearly so. In practice, the strategy of choosing b such that z becomes large was successful, in fact it was about as successful as it was in the QR case when tested on a large number of random matrices of various sizes.

This raises the question of why it was so successful. We can get some intuition for this by considering the following; due to the way row pivoting is done in the factorization of A, the ill-condition of A tends to be mostly reflected in the matrix U [1], and therefore U tends to be more ill-conditioned than L. The fact that U is ill-conditioned helps us promote growth in the length of z in the first step, and the fact that L is well conditioned helps us keep the advantage we gained in the first step.

To illustrate this point, consider the cases where L is well conditioned and $\sigma_1/\sigma_n \sim 1$, with σ_1 and σ_n being the largest and smallest singular values of L respectively. In the extreme case when $\sigma_1/\sigma_n = 1$, the unit ball gets mapped to a hyper ellipse where all the semi-axes have the same length, and so the image of the unit ball under L is a scaled version of the unit ball. This means that, if $\sigma_1/\sigma_n = 1$, it does not matter which direction z has in terms of how its length will be altered when multiplied by $L^{-\mathrm{T}}$, and so we maximize $\|x\|_2$ by choosing b such that $\|z\|_2$ is maximized. The more general point is that - given that U is more ill-conditioned than L - the first step is more important in terms of determining the length of x than the second step, and therefore the strategy of picking a b such to maximize $\|z\|_2$ is expected to be successful on average.

3.4 Operation counts associated with LU factorization

Estimating the condition number using an LU decomposition is more costly than it is with a QR factorization, since it doubles the number of triangular systems to solve per iteration in the PIA. It is therefore tempting to always use the QR decomposition, but we must keep in mind that the condition number will typically be computed in combination with one or more systems of equations that are to be solved, and it is considerations with regards to these computations that determines which factorization is used. Also, since QR factorization costs $\sim 4/3 \cdot n^3$ flops whereas LU factorization costs only $\sim 2/3 \cdot n^3$ flops [1], we are better of with an LU factorization even if our only goal is to estimate the condition number - given that the number of iterations is not to high - since the part of the process where we estimate $\kappa(A)$ is only $\mathcal{O}(n^2)$, and thus for large n the overall cost associated with estimating $\kappa(A)$ will be dominated by the cost of the factorization.

Chapter 4

Numerical testing

4.1 Numerical testing using QR decomposition

There are many ways to obtain an estimate of $\kappa(A)$ depending on which norm we are using, such as the number of iterations in the PIA, and the strategy for choosing b. If accuracy is an important aspect, then we can compute a 2-norm estimate, since we have the tools to reach arbitrary precision in that case. If computational cost is the most important aspect, we can settle for a 1-norm estimate, since the cost of computing $\|A\|_1$ exactly is only n^2 flops. If we then use two iterations in the PIA, using RLS to choose b, then this brings the total cost down to $n^2 + 2n^2 = 3n^2$, which is about as cheap as we can make it and still get a reliable order of magnitude estimate. To test the quality of this estimate, 500 random[1] 50×50 matrices where generated, and then for each such matrix we computed *the relative deviation*: $\frac{|\text{estimate}(\kappa(A)_1) - \kappa(A)_1|}{\kappa(A)_1}$. The relative deviation was greater than 1 for 7 out of the 500 matrices generated, and the highest such value was 1.52. Hence this method is successful in terms of indicating the correct order of magnitude. Note that the accuracy we can get in the 1-norm estimate is limited by the fact that the vector pair y and x that maximizes $\|y\|_2/\|x\|_2$ need not be the same pair of vectors that maximizes $\|y\|_1/\|x\|_1$, hence we can't improve accuracy indefinitely by just adding more iterations the way we can with a 2-norm estimate. Also note that we got a relative deviation greater than 1, which might seem impossible since we should have that $|\text{estimate}(\kappa(A)_1) - \kappa(A)_1| \leq 1$ given that our estimate is a lower bound, but we must keep in mind that we are actually obtaining a lower bound on $\kappa(R)_1$ which may very well be larger than $\kappa(A)_1$; equality between these two holds only in the 2-norm.

[1]By random we mean that the components where uniformly distributed on the interval $[-1, 1]$.

Next we consider the two different strategies for estimating σ_n^{-1} that were mentioned at the end of section 3.2; one strategy where we use LAS combined with 2 iterations, and one where we use RLS combined with 3 iterations. These are of interest given that both use approximately the same number of flops. We will refer to these strategies simply as the *2-iterations strategy* and *3-iterations strategy* respectively. 3000 random matrices of dimension 20×20 where generated, and for each matrix we computed an estimate using the 2-iterations and 3-iterations strategy. For 87% of the matrices, the 3-iterations strategy resulted in a better estimate, with the average of the ratio estimate$(\sigma_n^{-1})/\sigma_n^{-1}$ - which we refer to as the *success ratio* - being 0.96, and the smallest such ratio being 0.22. For the 2-iterations strategy the corresponding values where 0.92 and 0.09 respectively, and similar results where obtained for matrices of various sizes.

The two strategies were also tested on the matrix from the example in section 3.1 with $k = 1000$, and the average success ratio was 0.99979 using the 3-step strategy, and 0.999996 using the 2-step strategy. We also computed 100 estimations for this matrix with the 3-step strategy to see how likely it was to generate a bad estimate (since we have added an element of randomness into the process), and out of the 100 iterations the smallest success ratio was 0.9967, hence the modified version of LS seems to cope very well in this situation. Based on these results, it appears that out of the strategies for estimating σ_n^{-1} that require around $3n^2$ flops, the best one is the 3-iterations strategy.

4.2 Numerical testing using LU decomposition

First we wanted to test how well the two strategies for picking the components of b were now that we are using an LU factorization. To this end, we generated 4000 random matrices of dimensions 40×40 matrices and estimated σ_n^{-1} for each using RLS and LAS, using 2 iterations of the PIA in each case, and we obtained an average success ratio of 0.87 and 0.89 respectively, the smallest success ratios being 0.07 and 0.12 respectively, and with the LAS being more successful 54% of the times, hence LAS performed slightly better. We also wanted to see how much we would gain if we instead of using 2 iterations and LAS, we used the RLS and add an extra iteration, and so we tested using 3 iterations combined with RLS. The average success ratio was 0.96, which is a 7% increase in average accuracy compared to the strategy of performing 2 iterations combined with the LAS, and the cost of this improvement is an additional n^2 flops. A 7% increase in the average success ratio for an additional n^2 flops seems like a decent trade-off given the fact that we are only increasing the amount of work by a factor of $6n^2/5n^2 = 6/5 = 1.2$, not including the factorization cost of course.

The exact same strategies were tested on their ability to estimate σ_1. When comparing RLS against LAS we found that LAS was more successful 75% of the times, and the average success ratios where 0.74 and 0.79 respectively, a larger performance difference than when we estimated σ_n^{-1}. Performing RLS combined with 3 iterations gave an average success ratio of 0.82, which is only marginally better than the result we got when using 2 iterations combined with the LAS. In contrast to when we where estimating σ_n^{-1}, it seems that the LAS is preferable to using RLS.

Finally we generated 4000 matrices of dimension 40×40, and estimated $\kappa(A)_2$ and $\kappa(A)_1$ for each. For each A we used 3 iterations combined with RLS to estimate σ_n^{-1}, and 3 iterations combined with LAS to estimate σ_1, which adds up $6n^2 + 7n^2 = 13n^2$ flops to estimate $\kappa(A)_2$. To estimate $\kappa(A)_1$ we used 3 iterations combined with RLS to estimate $\|A^{-1}\|_1$, which together with the computation of $\|A\|_1$ then adds up to $6n^2 + n^2 = 7n^2$ flops to estimate $\kappa(A)_1$. When estimating $\kappa(A)_2$ we got that the average success ratio was 0.80, and the smallest success ratio was 0.11. Estimating $\kappa(A)_1$ we got that the average success ratio was 0.44, the smallest success ratio was 0.06, and 0.43% of the estimations had a success ratio below 0.1. Although the average success ratio was much lower for the 1-norm estimate, the lowest success ratio was not that much lower given the fact that it only uses $7n^2$ flops as opposed to the $13n^2$ flops used to estimate $\kappa(A)_2$. Hence the 1-norm does indeed seem to be a suitable choice when an order of magnitude estimate is all that is required.

Bibliography

[1] Alan K Cline, Cleve B Moler, George W Stewart, and James H Wilkinson. *An estimate for the condition number of a matrix*, volume 16. SIAM, 1979.

[2] Robert D Skeel. *Scaling for numerical stability in Gaussian elimination*, volume 26. ACM, 1979.

[3] Lloyd N Trefethen and David Bau III. *Numerical linear algebra*, volume 50. Siam, 1997.