The topic of the course and the book is numerical analysis. This is the sub-
ject which studies algorithms for computing expressions which are defined with real
numbers. There are three distinct issues that we will address:

- The development of algorithms.
- The analysis of algorithms.
- The effects of finite precision arithmetic.

The latter subject is one that is still not well understood at the most basic level.
Numerical analysis is an ancient subject. For example, one of the earliest tasks of interest was to compute square roots (and other roots), and Heron (or Hero, of Alexandria, circa 10-80 CE) had a method to do that which we recognize now as a variant of the Newton-Raphson method.

Let us illustrate this with the classical iterative method of Heron:

\[ x \leftarrow .5 \times (x + y/x) \tag{1} \]

This is an example of what is known as fixed-point iteration, in which one hopes to find a fixed point, that is, an \( x \) where the iteration quits changing. The reason this works is at a fixed point,

\[ x = .5 \times (x + y/x) \tag{2} \]

so that \( x = y/x \), or \( x^2 = y \).

We can implement (1) in Matlab/octave as

```matlab
function x=heron(x,y)
x=.5*(x+y/x);
end
```

To use this function, you need to start with some initial guess, say \( x = 1 \):

```matlab
x=heron(1,y)
```

But then you simply iterate:

```matlab
x=heron(x,y)
```

until you don’t see \( x \) changing.

We can examine the accuracy by a simple code:

```matlab
function x=errheron(x,y)
for i=1:6
    x=heron(x,y);
    errheron=x-sqrt(y)
end
```
This algorithm seems to be quite magic because it seems to ‘home in’ on the solution. We will see that the accuracy is doubling at each step.

Although the subject has ancient components, it has had periods of intense development followed by long periods of stagnation. In many cases, the new developments have coincided with the introduction of new forms of computing machines. For example, many of the basic theorems about computing solutions of ordinary differential equations were proved right after desktop “adding machines” became common at the turn of the 20th century. The advent of the digital computer in the mid-20th century spurred interest in solving partial differential equations and large systems of linear equations, as well as many other topics. The advent of parallel computer similarly stimulated research on new classes of algorithms. However, many fundamental questions remain open, and the subject is an active area of research today.

The book is a classic in the field. It was written by experts in the subject who were working at one of the most active research centers at the time. Some topics are dated, but the level of presentation remains superb. There are many missing topics, as the introduction to the Dover edition indicates, but the topics covered are still basic and a required part of the syllabus of numerical analysis. Subsequent books have generally tried to present the subject in a less rigorous way.

1 Preliminaries, Chapter 1

1.1 Norms, Chapter 1, Section 1

The main point of the section is to introduce ways to estimate accurately the size of things which have complex forms. A norm provides such a measure. This is a simple generalization of Euclidean distance, but it can apply to rather complex objects such as operators on vector spaces.

Section 0 of Chapter 2 provides an introduction to (or review of) linear algebra concepts. This should be read first.

There is a special class of vector spaces that we want to distinguish: the linear space of operators on a vector space. For these linear spaces, we introduce an additional property of the norm that respects the product structure of operators. There is a natural class of operator norms that come from duality.

Since operators can be represented as matrices, one might think that it is sufficient just to have norms on Euclidean spaces. However, some operator norms cannot be written as a norm on a Euclidean space consisting of the coefficients of the corresponding matrix. Conversely, there are some norms on matrices that cannot be written as an operator norm (exercise 2 on page 16).

The “big” result of the section (Theorem 3) is that we can almost think of the spectral radius as a norm. That is, we can always find a vector norm such that the corresponding operator norm is arbitrarily close to the spectral radius.
Another result that is important is that all norms on a finite dimensional vector space are equivalent. Norms are also Lipschitz continuous. By the triangle inequality,

\[ \|x\| = \|(x - y) + y\| \leq \|x - y\| + \|y\| \]  

(3)

Rearranging, we find

\[ \|x\| - \|y\| \leq \|x - y\| \]  

(4)

Reversing the names of \( x \) and \( y \), and using the fact that \( \|-x\| = \|x\| \), we find

\[ \|\|x\| - \|y\|| \leq \|x - y\| \]  

(5)

This provides an alternate route to proving continuity of the norm.

The Cauchy-Schwarz inequality holds for any non-negative, symmetric bilinear form:

\[ a(x, y) \leq \sqrt{a(x, x)} \sqrt{a(y, y)} \]  

(6)

The proof is the same as in the book, by expanding \( a(u + tv, u + tv) \) as a function of the scalar \( t \). This is the key step in proving the triangle equality for the associated norm: \( \|x\| = \sqrt{a(x, x)} \).

Theorem 1 on page 2 presents a result similar to the Jordan decomposition of a matrix, but one that is less precise and much simpler to prove. The theorem says that any matrix is similar to a triangular matrix with diagonal entries given by the eigenvalues of the matrix. The Jordan canonical form is one such representation, however, we do not need such specific information. For us, it is also significant that the proof is somewhat algorithmic, similar to algorithms we will consider later. However, it is only superficially so in that each step requires the determination of an eigenvector and eigenvalue pair, something addressed later in the book.

What sort of uniqueness holds for the decompositon in Theorem 1? Consider the following fact. Suppose that \( A \) and \( B \) are upper-triangular matrices of the form

\[
A = \begin{pmatrix}
a_1 & a_{12} & a_{13} & \cdots & a_{1n} \\
0 & a_2 & a_{23} & \cdots & a_{2n} \\
0 & 0 & a_3 & \cdots & \vdots \\
0 & 0 & 0 & \cdots & a_n
\end{pmatrix}
\quad
B = \begin{pmatrix}
b_1 & b_{12} & b_{13} & \cdots & b_{1n} \\
0 & b_2 & b_{23} & \cdots & b_{2n} \\
0 & 0 & b_3 & \cdots & \vdots \\
0 & 0 & 0 & \cdots & b_n
\end{pmatrix}
\]  

(7)

Then the product \( AB \) is (a) also an upper-triangular matrix with the property (b) that

\[
AB = \begin{pmatrix}
a_1b_1 & c_{12} & c_{13} & \cdots & c_{1n} \\
0 & a_2b_2 & c_{23} & \cdots & c_{2n} \\
0 & 0 & a_3b_3 & \cdots & \vdots \\
0 & 0 & 0 & \cdots & a_nb_n
\end{pmatrix}
\]  

(8)

for some coefficients \( c_{ij} \). That is, the diagonal entries of the product are the product of the diagonal entries.
One corollary of (8) tells us about the lack of uniqueness in Theorem 1. If \( B = P^{-1}AP \) is upper-triangular, then for any upper-triangular matrix \( U \) with non-zero diagonal entries,

\[
U^{-1}BU = U^{-1}P^{-1}APU = (PU)^{-1}A(UP)
\]  

provides another representation with the same properties stated in Theorem 1.

In the proof of Theorem 3, there is an interesting intermediate result, namely that with a suitable similarity transformation, any matrix can be transformed so that

\[
P^{-1}AP = \Lambda + \Delta
\]  

where \( \Lambda \) is a diagonal matrix (having the eigenvalues of \( A \) on the diagonal) and \( \Delta \) can be as small as we like (at the expense of making \( P \) very large).

1.1.1 Convergent matrices, Chapter 1, Section 1.1

There are many situations in which the result of an algorithm can be written as multiplication by a fixed matrix \( A \). Thus repeating the algorithm \( n \) times is equivalent to applying the matrix \( A^n \). Sometimes, this might represent the error in some process. Thus we are interested in precise conditions when \( A^n \to 0 \) as \( n \to \infty \).

Theorem 4 presents the main result giving three equivalent conditions that imply \( A^n \to 0 \) as \( n \to \infty \). Its Corollary is a simple test which can be used as a guide. Theorem 5 uses the concept of convergence to give an algorithm for inverting a particular kind of matrix. Its corollary gives upper and lower bounds on the size of this inverse.

1.1.2 Review questions

What conditions are required to know that a matrix has an eigenvector and eigenvalue pair? (This is required for the first step in the proof of Theorem 1, and the induction step.)

Is the first line of page 11 right? What does it mean to have ‘a complete set of orthonormal eigenvectors’? How is this property different from the condition just before, that every matrix has an eigenvector and eigenvalue pair?

1.1.3 Homework

(1-3) page 16, numbers 1,2,5.

1.1.4 Typos

On page 7, in the 5th line of the proof of Theorem 2, the word ‘ball’ should instead be ‘hypercube.’
1.2 Floating point arithmetic, Chapter 1, Section 2

The main point of the section is to provide a way to analyze algorithms which are executed in floating point arithmetic on a digital computer. The algorithms we consider involve real numbers at an abstract level, but we are forced to work with a finite approximation of them.

The important thing to realize is that we are going to prove theorems about the actual computations, not just about their theoretical counterparts executed using real numbers. The latter is of interest, but not sufficient to guarantee success of the approximated computations done using floating point arithmetic. This point has not yet been fully understood in the field, and there have been other models of numerical analysis proposed, e.g., by Smale et al.

Actual floating point arithmetic is quite complex and does not lend itself to a simple representation. There is a floating point hardware standard developed by the IEEE and most hardware follows this at the moment. However, these specifications simply provide bounds on behavior of floating point.

The model used in the book is the standard way of representing an upper bound of the inaccuracies of floating point. We emphasize that is just a model, and there could be other more accurate models possible. One flaw of this model is that, even if some error bound is shown to be worst case for the model, it may not be worst case for floating point arithmetic for any set of data.

It could be that \( f_\ell(a + b) = f_\ell(a) + f_\ell(b) \) exactly for some \( a \) and \( b \). However, it is also easy to see that this will fail in most cases. It might be an interesting exercise to compute the distribution of errors in floating point.

Cancellation is a major source of error. Note that \( f_\ell(a - b) = (a - b)(1 + \delta) \) does not mean that \( f_\ell(f_\ell(a + e) - b) \) is at all close to \( a + e - b \). Floating point arithmetic is not associative. Cancellation amplifies errors that have already occurred. It can easily be that \( f_\ell(a + e) = a \), and if \( b = a \) we get zero for the result instead of \( e \).

The idea of backwards error analysis is to show that the following result holds for a given algorithm: for any input data, the approximate algorithm using floating point will produce the same result as the exact algorithm using real numbers but with a slightly perturbed input data set. Once this result is established, it quantifies the potential effect of the use of floating point. Such a theorem typically would say how big the perturbation is in the initial data required to reconcile the exact and approximate algorithms. However, we still need to understand the error propagation of the algorithm: how does a small change in input data affect the output? This depends on the stability of the algorithm, or of the process being approximated.

Backwards error analysis is itself still too pessimistic in important cases. One can
give a fairly complete analysis in the case of computing a sum \( \sum_{i=1}^{n} a_i \). Let \( A_n \) denote the result obtained by floating point arithmetic. Then

\[
\sum_{i=1}^{n} a_i - A_n \approx \sum_{i=1}^{n} \sigma_i \delta_i + O(\delta^2)
\]  

(11)

where \( \sigma_j = \sum_{i=1}^{j} a_i \). Compare this with (9a) in the text which is much more pessimistic (but also a rigorous upper bound).

Doing the same level of analysis for more complex algorithms remains a topic of research.

1.2.1 Homework

(4) page 21, number 2. (5) Is there a sequence of numbers for which the backwards error estimate is sharp?

1.3 Well posed algorithms, Chapter 1, Section 3

This section attempts to provide a general framework for “well posed” algorithms. We won’t pursue the generality presented here but will rather just look at the particular example presented.

Suppose that we want to roots of a quadratic equation \( x^2 + 2bx + c = 0 \) where \( b < 0 \) and we chose the algorithm

\[
x \leftarrow -b - \sqrt{b^2 - c}
\]  

(12)

This fails if we have \( c = \epsilon^2 b^2 \) (it returns \( x = 0 \)) as soon as \( \epsilon^2 = c/b^2 \) is small enough that \( f(1 - \epsilon^2) = 1 \).

We can see the behavior in some simple codes. But first, let us simplify the problem further so that we just have one parameter to deal with. Suppose that the equation to be solved is of the form

\[
x^2 - 2bx + 1 = 0.
\]

That is, we switch \( b \) to \( -b \) and set \( c = 1 \). In this case, the two roots are multiplicative inverses of each other. Define \( x_{\pm} = b \pm \sqrt{b^2 - 1} \). Then \( x_- = 1/x_+ \).

There are various possible algorithms. We could use one of the two formulæ \( x_{\pm} = b \pm \sqrt{b^2 - 1} \) directly. More precisely, let us write \( \tilde{x}_{\pm} \approx b \pm \sqrt{b^2 - 1} \) to indicate that we implement this in floating point. Correspondingly, there is another pair of algorithms that start by computing \( \tilde{x}_{\pm} \) and then define, say, \( \hat{x}_- \approx 1/\tilde{x}_+ \). A similar algorithm could determine \( \hat{x}_+ \approx 1/\tilde{x}_- \).

All four of these algorithms will have different behaviors. We expect that the behaviors of the algorithms for computing \( \tilde{x}_- \) and \( \hat{x}_- \) will be dual in some way to those for computing \( \tilde{x}_+ \) and \( \hat{x}_+ \), so we consider only the first pair.
To implement these algorithms, we choose Matlab’s scripting syntax. As a programming language, this has significant flaws, but its use is extremely widespread. In addition to the commercial interpreter provided by the Mathworks company, a public domain implementation called octave is available.

First of all, the function minus implements the $\bar{x}_-$ square root algorithm:

```matlab
function x=minus(b)
% solving = 1-2bx +x^2
x=b-sqrt(b^2-1);
```

To know if it is getting the right answer, we need another one to check the answer:

```matlab
function error=check(b,x)
error = 1-2*b*x +x^2;
```

To automate the process, we put the two together:

```matlab
function error=chekminus(b)
x=minus(b);
error=check(b,x)
```

For example, when $b = 10^6$, we find the error is $-7.6 \times 10^{-6}$.

Finally, the algorithm for $\hat{x}_-$ is given by

```matlab
function x=plusinv(b)
% solving = 1-2bx +x^2
y=b+sqrt(b^2-1);
x=1/y;
```

Similarly, we can check the accuracy of this computation by the code

```matlab
function error=chekplusinv(b)
x=plusinv(b);
error=check(b,x)
```

Now when $b = 10^6$, we find the error is $-2.2 \times 10^{-17}$. And the bigger $b$ becomes, the more accurate it becomes.

### 1.3.1 Homework

(6) Change the function minus for computing $\bar{x}_-$ and the function plusinv for computing $\hat{x}_-$ to functions for computing $\bar{x}_+$ (call that function plus) and $\hat{x}_+$ (call that function minusinv). Use the check function to see where they work well and where they fail. Compare that with the corresponding behavior for minus and plusinv.
2 Matrix inversion/system solution, Chapter 2

2.1 Matrix examples

Many problems are posed using matrix equations. We give two examples to indicate the scale of the problem.

2.1.1 Two-point boundary value problems

Many physical models are described by differential equations. For a two-point boundary value problem, a finite difference approximation produces a linear system where the $N \times N$ matrix in question is of the form

\[
A = \begin{pmatrix}
    a_1 & b_1 & 0 & \ldots & 0 & 0 \\
    b_1 & a_2 & b_2 & \ldots & 0 & 0 \\
    0 & b_2 & a_3 & \ldots & 0 & 0 \\
    \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
    0 & \ldots & 0 & b_{N-2} & a_{N-1} & b_{N-1} \\
    0 & \ldots & \ldots & 0 & b_{N-1} & a_N
\end{pmatrix}
\]  

(13)

where

\[
b_i = \frac{-1}{h_i}, \quad a_i = \frac{1}{h_i} + \frac{1}{h_{i+1}}
\]  

(14)

for a mesh $x_0 < x_1 < \cdots < x_N$ where $h_i = x_{i+1} - x_i$. Finding the deflection $X$ of a rod subject to a force $F$ would require solving $AX = F$.

Difference (or finite element) methods generate matrices with structure very similar to (13), with $N$ limited only by the size of memory. It is a significant simplification that this matrix is *banded*, meaning that it is zero outside a band (or strip) around the diagonal.

2.1.2 Implicit rankings

There are many situations in which it is desired to estimate rankings of related objects based on relationships among the objects. Suppose that we want to measure popularity. One way to do that is to see how many people keep your phone number in their personal digital assistants. If there are many such people, then you must be popular. But you are surely more popular if the people who keep your number are themselves popular. So a better way to rank things would be to keep track not only of how many people list you but also their rankings. For example, your ranking might be defined to be a simple multiple of the sum of all rankings of people who list you. However, this is a circular definition. How can you define your ranking until you know all other rankings? And how can you define them since presumably some of them must depend on your own?
The way out of this dilemma is to write the relationships using matrices and then observe that it leads to an eigenvalue problem, as follows. Let $\lambda > 0$ denote an unknown parameter for the moment which represents the constant of proportionality for rankings. Let $A = (a_{ij})$ denote the matrix with the property that $a_{ij} = 1$ if and only if the $j$-th person lists the $i$-th person’s phone number, and zero otherwise. Then the $i$-th ranking $x_i$ can be determined from other rankings ($x_j$) by the relationship

$$x_i = \frac{1}{\lambda} \sum_{\{j : a_{ij} \neq 0\}} x_j.$$  \hfill (15)

This just says that the $i$-th ranking is proportional to the sum of the rankings of the entities that point to it, with constant of proportionality given by $1/\lambda$. Using properties of the definition of the matrix $A = (a_{ij})$, we thus see that for all $i$

$$\lambda x_i = \sum_{\{j : a_{ij} \neq 0\}} x_j = \sum_j a_{ij} x_j = (AX)_i.$$  \hfill (16)

This says that $X$ and $\lambda$ are eigenpairs: $AX = \lambda X$. \footnote{If the matrix $A$ is irreducible [1], then there is a non-negative eigenpair, where $\lambda$ is the dominant eigenvalue. In this case, the power method can be used to approximate $X$ and $\lambda$.} Note that rankings, like eigenvalues, only can be meaningfully defined to within a constant factor. That is, absolute rankings make no sense; only rankings relative to others are useful.

Computing popularity of people based on phone number tabulation may be a silly example. But it is prototypical of other uses which are more realistic. The use of link popularity for search engines for the World Wide Web is an example. Instead of phone numbers, we record links to web pages. The above ideas carry over immediately and provide a model of rankings of web pages based on the links that point to them and their rankings. Current web search engines compute the corresponding eigenvalue problem for the entire web each day, with several billions of web pages ranked currently. For such a size of matrix, it is significant that the matrices are sparse, that is, they are mostly zeros. A typical web page might only link to a few dozen other web pages. However, the location of the nonzero entries is quite arbitrary in this case, not leading to a banded structure.

References