Numerical Analysis
Notes for Math 575A

William G. Faris
Program in Applied Mathematics
University of Arizona

Fall 1992

Contents

æ

Chapter 1

Nonlinear equations

1.1 Introduction

This chapter deals with solving equations of the form $f(x) = 0$, where f is a continuous function.

The usual way in which we apply the notion of continuity is through sequences. If g is a continuous function, and c_n is a sequence such that $c_n \to c$ as $n \to \infty$, then $g(c_n) \to g(c)$ as $n \to \infty$.

Here is some terminology that we shall use. A number x is said to be *positive* if $x \geq 0$. It is *strictly positive* if $x > 0$. (Thus we avoid the mind-numbing term "non-negative.") A sequence a_n is said to be increasing if $a_n \leq a_{n+1}$ for all n. It is said to be strictly increasing if $a_n < a_{n+1}$ for all n. There is a similar definition for what it means for a function to be increasing or strictly increasing. (This avoids the clumsy locution "non-decreasing.")

Assume that a sequence a_n is increasing and bounded above by some $c < \infty$, so that $a_n \leq c$ for all n. Then it is always true that there is an $a \leq c$ such that $a_n \to a$ as $n \to \infty$.

1.2 Bisection

The bisection method is a simple and useful way of solving equations. It is a constructive implementation of the proof of the following theorem. This result is a form of the *intermediate value theorem.*

Theorem 1.2.1 *Let g be a continuous real function on a closed interval $[a, b]$ such that $g(a) \leq 0$ and $g(b) \geq 0$. Then there is a number r in the*

interval with $g(r) = 0$.

Proof: Construct a sequence of intervals by the following procedure. Take an interval $[a, b]$ on which g changes sign, say with $g(a) \leq 0$ and $g(b) \geq 0$. Let $m = (a + b)/2$ be the midpoint of the interval. If $g(m) \geq 0$, then replace $[a, b]$ by $[a, m]$. Otherwise, replace $[a, b]$ by $[m, b]$. In either case we obtain an interval of half the length on which g changes sign.

The sequence of left-hand points a_n of these intervals is an increasing sequence bounded above by the original right-hand end point. Therefore this sequence converges as $n \to \infty$. Similarly, the sequence of right-hand points b_n is a decreasing sequence bounded below by the original left-hand end point. Therefore it also converges. Since the length $b_n - a_n$ goes to zero as $n \to \infty$, it follows that the two sequences have the same limit r.

Since $g(a_n) \leq 0$ for all n, we have that $g(r) \leq 0$. Similarly, since $g(b_n) \geq 0$ for all n, we also have that $g(r) \geq 0$. Hence $g(r) = 0$. □

Note that if a and b are the original endpoints, then after n steps one is guaranteed to have an interval of length $(b - a)/2^n$ that contains a root.

In the computer implementation the inputs to the computation involve giving the endpoints `a` and `b` and the function `g`. One can only do a certain number of steps of the implementation. There are several ways of accomplishing this. One can give the computer a `tolerance` and stop when the the length of the interval does not exceed this value. Alternatively, one can give the computer a fixed number of steps `nsteps`. The output is the sequence of `a` and `b` values describing the bisected intervals.

```
void bisect(real tolerance, real a, real b, real (*g)(real) )
{
  real m ;
  while( b - a > tolerance )
  {
    m = (a+b) / 2 ;
    if( g(a) * g(m) <= 0.0 )
   b = m ;
    else
   a = m ;
    display(a, b) ;
  }
}
```

The construction here is the `while` loop, which is perhaps the most fundamental technique in programming.

Here is an alternative version in which the iterations are controlled by a counter `n`.

```
void bisect(int nsteps, real a, real b, real (*g)(real) )
{
  int n ;
  real m ;
  n = 0 ;
  while( n < nsteps )
  {
    m = (a+b) / 2 ;
    if( g(a) * g(m) <= 0.0 )
   b = m ;
    else
   a = m ;
    n = n + 1 ;
    display(n, a, b) ;
  }
}
```

Algorithms such as this are actual computer code in the programming language C. It should be rather easy to read even without a knowledge of *C*. The key word `void` indicates that the bisect function is a procedure and not a function that returns a meaningful value. The parameter declaration real `(*g)(real)` indicates that g is a variable that can point to functions that have been defined (real functions of real arguments). When reading the text of a function such as `bisect`, it is useful to read the sign `=` as "becomes."

To make a complete program, one must put this in a program that calls this procedure.

```
/* bisection */

#include <stdio.h>
#include <math.h>

typedef double real;
void bisect(int, real, real, real (*)(real));
real quadratic(real) ;
void fetch(int *, real *, real *);
void display(int, real, real) ;

int main()
{
   int nsteps;
```

```
   real a, b;
   real (*f)(real);

   f = quadratic;
   fetch(nsteps, a, b);
   display(nsteps,a,b);
   bisect(nsteps,a,b,f);
   return 0;
}
```

This program begins with declarations of a new type `real` and of four functions `bisect`, `quadratic`, `fetch`, and `display`. The *main* program uses these functions to accomplish its purpose; it returns the integer value `0` only to proclaim its satisfaction with its success.

One also needs to define the function `quadratic`.

```
real quadratic( real x)
{
   return ( x * x - 2.0 )  ;
}
```

This particular function has roots that are square roots of two. We shall not go into the dismal issues of input and output involved with `fetch` and `display`.

Another interesting question is that of uniqueness. If g is strictly increasing on $[a, b]$, then there is at most one solution of $g(x) = 0$.

The easiest way to check that g is strictly increasing on $[a, b]$ is to check that $g'(x) > 0$ on (a, b). Then for $a \leq p < q \leq b$ we have by the mean value theorem that $g(q) - g(p) = g'(c)(q - p) > 0$ for some c with $p < c < q$. Thus $p < q$ implies $g(p) < g(q)$.

One can use a similar idea to find maxima and minima. Let g be a continuous function on $[a, b]$. Then there is always a point r at which g assumes its maximum.

Assume that g is *unimodal*, that is, that there exists an r such that g is strictly increasing on $[a, r]$ and strictly decreasing on $[r, b]$. The computational problem is to locate the point r at which the maximuum is assumed.

The *trisection* method accomplishes this task. Divide $[a, b]$ into three equal intervals with end points $a < p < q < b$. If $g(p) \leq g(q)$, then r must be in the smaller interval $[p, b]$. Similarly, if $g(p) \geq g(q)$, then r must be in the smaller interval $[a, q]$. The method is to repeat this process until a sufficiently small interval is obtained.

Projects

1. Write a bisection program to find the square roots of two. Find them.
2. Use the program to solve $\sin x = x^2$ for $x > 0$.
3. Use the program to solve $\tan x = x$ with $\pi/2 < x < 3\pi/2$.
4. Write a trisection program to find maxima. Use it to find the minimum of $h(x) = x^3/3 + \cos x$ for $x \geq 0$.

Problems

1. Show that $\sin x = x^2$ has a solution with $x > 0$. Be explicit about the theorems that you use.
2. Show that $\sin x = x^2$ has at most one solution with $x > 0$.
3. Show that $\tan x = x$ has a solution with $\pi/2 < x < 3\pi/2$.
4. Show that $x^5 - x + 1/4 = 0$ has has at least two solutions between 0 and 1.
5. Show that $x^5 - x + 1/4 = 0$ has has at most two solutions between 0 and 1.
6. Prove a stronger form of the intermediate value theorem: if g is continuous on $[a, b]$, then g assumes every value in $[g(a), g(b)]$.
7. How many decimal places of accuracy does one gain at each bisection?
8. How many decimal places are obtained at each trisection?

1.3 Iteration

1.3.1 First order convergence

Recall the *intermediate value theorem*: If f is a continuous function on the interval $[a, b]$ and $f(a) \leq 0$ and $f(b) \geq 0$, then there is a solution of $f(x) = 0$ in this interval.

This has an easy consequence: the *fixed point theorem*. If g is a continuous function on $[a, b]$ and $g(a) \geq a$ and $g(b) \leq b$, then there is a solution of $g(x) = x$ in the interval.

Another approach to numerical root-finding is *iteration*. Assume that g is a continuous function. We seek a fixed point r with $g(r) = r$. We can attempt to find it by starting with an x_0 and forming a sequence of iterates using $x_{n+1} = g(x_n)$.

Theorem 1.3.1 *Let g be continuous and let x_n a sequence such that $x_{n+1} = g(x_n)$. Then if $x_n \to r$ as $n \to \infty$, then $g(r) = r$.*

This theorem shows that we need a way of getting sequences to converge. One such method is to use increasing or decreasing sequences.

Theorem 1.3.2 *Let g be a continuous function on $[r, b]$ such that $g(x) \leq x$ for all x in the interval. Let $g(r) = r$ and assume that $g'(x) \geq 0$ for $r < x < b$. Start with x_0 in the interval. Then the iterates defined by $x_{n+1} = g(x_n)$ converge to a fixed point.*

Proof: By the mean value theorem, for each x in the interval there is a c with $g(x) - r = g(x) - g(r) = g'(c)(x - r)$. It follows that $r \leq g(x) \leq x$ for $r \leq x \leq b$. In other words, the iterations decrease and are bounded below by r. □

Another approach is to have a bound on the derivative.

Theorem 1.3.3 *Assume that g is continuous on $[a, b]$ and that $g(a) \geq a$ and $g(b) \leq b$. Assume also that $|g'(x)| \leq K < 1$ for all x in the interval. Let x_0 be in the interval and iterate using $x_{n+1} = g(x_n)$. Then the iterates converge to a fixed point. Furthermore, this fixed point is unique.*

Proof: Let r be a fixed point in the interval. By the mean value theorem, for each x there is a c with $g(x) - r = g(x) - g(r) = g'(c)(x - r)$, and so $|g(x) - r| = |g'(c)||x - r| \leq K|x - r|$. In other words each iteration replacing x by $g(x)$ brings us closer to r. □

We say that r is a stable fixed point if $|g'(r)| < 1$. We expect convergence when the iterations are started near a stable fixed point.

If we want to use this to solve $f(x) = 0$, we can try to take $g(x) = x - kf(x)$ for some suitable k. If k is chosen so that $g'(x) = 1 - kf'(x)$ is small for x near r, then there should be a good chance of convergence.

It is not difficult to program fixed point iteration. Here is a version that displays all the iterates.

```
void iterate(int nsteps, real x, real (*g)(real) )
{
   n = 0 ;
   while( n < nsteps)
   {
      x = g(x) ;
      n = n + 1 ;
      display(n, x) ;
   }
}
```

1.3.2 Second order convergence

Since the speed of convergence in iteration with g is controlled by $g'(r)$, it follows that the situation when $g'(r) = 0$ is going to have special properties.

It is possible to arrange that this happens! Say that one wants to solve $f(x) = 0$. *Newton's method* is to take $g(x) = x - f(x)/f'(x)$. It is easy to check that $f(r) = 0$ and $f'(r) \neq 0$ imply that $g'(r) = 0$.

Newton's method is not guaranteed to be good if one begins far from the starting point. The damped Newton method is more conservative. One defines $g(x)$ as follows. Let $m = f(x)/f'(x)$ and let $y = x - m$. While $|f(y)| > |f(x)|$ replace m by $m/2$ and let $y = x - m$. Let $g(x)$ be the final value of y.

Projects

1. Implement Newton's method as a special case of the fixed point iterations. Use this to find the largest root of $\sin x - x^2 = 0$. Describe what happens if you start the iteration with .46.

2. Implement the damped Newton's method. Use this to find the largest root of $\sin x - x^2 = 0$. Describe what happens if you start the iteration with .46.

3. Find all roots of $2 \sin x - x = 0$ numericallly. Use some version of Newton's method.

Problems

1. Let $g(x) = (2/3)x + (7/3)(1/x^2)$. Show that for every initial point x_0 above the fixed point the iterations converge to the fixed point. What happens for initial points $x_0 > 0$ below the fixed point?

2. Assume that $x \leq g(x)$ and $g'(x) \geq 0$ for $a \leq x \leq r$. Show that it follows that $x \leq g(x) \leq r$ for $a \leq x \leq r$ and that the iterations increase to the root.

3. Prove the fixed point theorem from the intermediate value theorem.

4. In fixed point iteration with a g having continuous derivative and stable fixed point r, find the limit of $(x_{n+1} - r)/(x_n - r)$. Assume the iterations converge.

5. Perhaps one would prefer something that one could compute numerically. Find the limit of $(x_{n+1} - x_n)/(x_n - x_{n-1})$ as $n \to \infty$.

6. How many decimal places does one gain at each iteration?

7. In fixed point iteration with a g having derivative $g'(r) = 0$ and continuous second derivative, find the limit of $(x_{n+1} - r)/(x_n - r)^2$.

8. Describe what this does to the decimal place accuracy at each iteration.

9. Calculate $g'(x)$ in Newton's method.

10. Show that in Newton's method $f(r) = 0$ with $f'(r) \neq 0$ implies $g'(r) = 0$.

11. Calculate $g''(x)$ in Newton's method.

12. Consider Newton's method for $x^3 - 7 = 0$. Find the basin of attraction of the positive root. Be sure to find the entire basin and prove that your answer is correct. (The basin of attraction of a fixed point of an interation function g is the set of all initial points such that fixed point iteration starting with that initial point converges to the fixed point.)

13. Consider Newton's method to find the largest root of $\sin x - x^2 = 0$. What is the basin of attraction of this root? Give a mathematical argument that your result is correct.

14. Show that in Newton's method starting near the root one has either increase to the the root from the left or decrease to the root from the right. (Assume that $f'(x)$ and $f''(x)$ are non-zero near the root.) What determines which case holds?

15. Assume that $|x_{n+1} - r| \leq K|x_n - r|^2$ for all $n \geq 0$. Find a condition on x_0 that guarantees that $x_n \to r$ as $n \to \infty$.

16. Is the iteration function in the damped Newton's method well-defined? Or could the halving of the steps go on forever?

1.4 Some C notations

1.4.1 Introduction

This is an exposition of a fragment of C sufficient to express numerical algorithms involving only scalars. Data in C comes in various *types*. Here we consider arithmetic types and function types.

A C program consists of *declarations* and *function definitions*. The declarations reserve variables of various types. A function definition describes how to go from input values to an output value, all of specified types. It

may also define a procedure by having the side effect of changing the values of variables.

The working part of a function definition is formed of *statements*, which are commands to perform some action, usually changing the values of variables. The calculations are performed by evaluating *expressions* written in terms of constants, variables, and functions to obtain values of various types.

1.4.2 Types

Arithmetic

Now we go to the notation used to write a C program. The basic types include arithmetic types such as:

```
char
int
float
double
```

These represent character, integer, floating point, and double precision floating point values.

There is also a `void` type that has no values.

A *variable* of a certain type associates to each machine state a *value* of this type. In a computer implementation a variable is realized by a location in computer memory large enough to hold a value of the appropriate type.

Example: One might declare `n` to be an integer variable and `x` to be a float variable. In one machine state `n` might have the value 77 and `x` might have the value 3.41.

Function

Another kind of data object is a function. The type of a function depends on the types of the arguments and on the type of the value. The type of the value is written first, followed by a list of types of the arguments enclosed in parentheses.

Example: `float (int)` is the type of a function of an integer argument returning float. There might be a function `convert` of this type defined in the program.

Example: `float ( float (*)(float), float, int)` is the type of a function of three arguments of types `float (*)(float)`, `float`, and `int` returning `float`. The function `iterate` defined below is of this type.

A function is a constant object given by a function definition. A function is realized by the code in computer memory that defines the function.

Pointer to function

There are no variable functions, but there can be variables of type pointer to function. The values of such a variable indicate which of the function definitions is to be used.

Example: `float (*)(int)` is the type of a pointer to a function from `int` to `float`. There could be a variable `f` of this type. In some machine state it could point to the function `convert`.

The computer implementation of pointer to function values is as addresses of memory locations where the functions are stored.

A function is difficult to manipulate directly. Therefore in a C expression the value of a function is not the actual function, but the pointer associated with the function. This process is known as *pointer conversion*.

Example: It is legal to make the assignment `f = convert`.

1.4.3 Declarations

A declaration is a specification of variables or functions and of their types.

A *declaration* consists of a value type and a list of *declarators* and is terminated by a semicolon. These declarators associate identifiers with the corresponding types.

Example: `float x, y ;` declares the variables `x` and `y` to be of type float.

Example: `float (*g)(float) ;` declares `g` as a pointer to function from `float` to `float`.

Example: `float iterate( float (*)(float), float, int) ;` declares a function `iterate` of three arguments of types `float (*)(float)`, `float`, and `int` returning `float`.

1.4.4 Expressions

Variables

An expression of a certain type associates to each machine state a value of this type.

Primary expressions are the expressions that have the highest precedence. Constants and variables are primary expressions. An arbitrary expression can be converted to a primary expression by enclosing it in parentheses.

Usually the value of the variable is the data contained in the variable. However the value of a function is the pointer that corresponds to the function.

Example: After the declaration `float x, y ;` and subsequent assignments the variables `x` and `y` may have values which are float.

Example: After the declaration `float (*g)(float) ;` and subsequent assignments the variable `g` may have a pointer to function on float returning a float value.

Example: After the declaration `float iterate( float (*)(float), float, int) ;` and a function definition the function `iterate` is defined. Its value is the pointer value that corresponds to the function. Thus if `h` is a variable which can point to such a function, then the assignment `h = iterate ;` is legal.

Function calls

A *function call* is an expression formed from a pointer to function expression and an argument list of expressions. Its value is obtained by finding the pointer value, evaluating the arguments and copying their values, and using the function corresponding to the pointer value to calculate the result.

Example: Assume that `g` is a function pointer that has a pointer to some function as its value. Then this function uses the value of `x` to obtain a value for the function call `g(x)`.

Example: The function `iterate` is defined with the heading `iterate( float (*g)(float), float x, float n )`. A function call `iterate(square, z, 3)` uses the value of `iterate`, which is a function pointer, and the values of `square`, `x`, and `3`, which are function pointer, float, and integer. The values of the arguments `square`, `z`, and `3` are copied to the parameters `g`, `x`, and `n`. The computation described in the function definition is carried out, and a float is returned as the value of the function call.

Casts

A data type may be changed by a *cast* operator. This is indicated by enclosing the type name in parentheses.

Example: `7 / 2` evaluates to `3` while `(float)7 / 2` evaluates to `3.5`.

Arithmetic and logic

There are a number of ways of forming new expressions from old.

The *unary* operators `+` and `-` and the negation `!` can form new expressions.

Multiplicative expressions are formed by the binary operators `*`, `/` and `%`. The last represents the remainder in integer division.

Additive expressions are formed by the binary operators `+` and `-`.

Relational expressions are formed by the inequalities `<` and `<=` and `>` and `>=`.

Equality expressions are formed by the equality and negated equality `==` and `!==`.

Logical AND expression are formed by `&&`.

Logical OR expression are formed by `||`.

Assignments

Another kind of expression is the *assignment* expression. This is of the form *variable* `=` *expression*. It takes the expression on the right, evaluates it, and assigns the value to the variable on the left (and to the assignment expression). This changes the machine state.

An assignment is is read variable "becomes" expression.

Warning: This should be distinguished from an *equality* expression of the form *expression* `==` *expression*. This is read expression "equals" expression.

Example: `i = 0`

Example: `i = i + 1`

Example: `x = g(x)`

Example: `h = iterate`, where `h` is a function pointer variable and `iterate` is a function constant.

1.4.5 Statements

Expression statements

A statement is a command to perform some action changing the machine state.

Among the most important are statements formed from expressions (such as assignment expressions) of the form *expression* `;`

Example: `i = 0 ;`

Example: `i = i + 1 ;`

Example: `x = g(x) ;`

Example: `h = iterate ;`, where `h` is a function pointer variable and `iterate` is a function constant.

In the compound statement part of a function definition the statement `return` *expression* `;` stops execution of the function and returns the value of the expression.

Control statements

There are several ways of building up new statements from old ones. The most important are the following.

A *compound statement* is of the form:
{ *declaration-list statement-list* }
An *if-else statement* is of the form:
`if (` *expression* `)` *statement* `else` *statement*
A *while statement* is of the form:
`while (` *expression* `)` *statement*
The following pattern of statements often occurs:
expr1 `;` `while (` *expr2* `)` { *statement expr 3*; }
Abbreviation: The above pattern is abbreviated by the *for* statement of the form:
`for(` *expr1* `;` *expr2* `;` *expr3* `)` *statement*

1.4.6 Function definitions

A function definition begins with a heading that indicates the type of the output, the name of the function, and a parenthesized parameter list. Each element of the parameter list is a specification that identifies a parameter of a certain type. The body of a function is a single compound statement.

Example: The definition of `square` as the squaring function of type function of float returning float is

```
float square( float y )
{
   return y*y ;
}
```

The definition of `iterate` (with parameters `g`, `x`, `n` of types pointer to function of float returning float, float, and integer) returning float is

```
float iterate( float (*g)(float), float x, float n )
{
   int i ;
   i = 0 ;
   while ( i < n ) do
   {
      x = g(x) ;
      i = i + 1 ;
   }
   return x ;
}
```

Example: Consider the main program

```
main()
{
   float z, w ;
   z = 2.0 ;
   w = iterate(square, z, 3) ;
}
```

The function call `iterate(square, z, 3)` has argument expressions which are a function `square`, a float `z`, and an integer `3`. These arguments are evaluated and the values are copied to the parameters `g`, `x`, and `n`, which are pointer to function, float, and integer objects. In the course of evaluation the parameter `x` changes its value, but `z` does not change its value of 2.0. The value returned by `iterate(square,z,3)` is 256.0. The ultimate result of the program is to assign 2.0 to `z` and 256.0 to `w`.

Chapter 2

Linear Systems

This chapter is about solving systems of linear equations. This is an algebraic problem, and it provides a good place in which to explore matrix theory.

2.1 Shears

In this section we make a few remarks about the geometric significance of Gaussian elimination.

We begin with some notation. Let $\mathbf{z}$ be a vector and $\mathbf{w}$ be another vector. We think of these as column vectors. The *inner product* of $\mathbf{w}$ and $\mathbf{z}$ is $\mathbf{w}^T\mathbf{z}$ and is a scalar. The *outer product* of $\mathbf{z}$ and $\mathbf{w}$ is $\mathbf{z}\mathbf{w}^T$, and this is a matrix.

Assume that $\mathbf{w}^T\mathbf{z} = 0$. A *shear* is a matrix M of the form $I + \mathbf{z}\mathbf{w}^T$. It is easy to check that the inverse of M is another shear given by $I - \mathbf{z}\mathbf{w}^T$.

The idea of Gaussian elimination is to bring vectors to a simpler form by using shears. In particular one would like to make the vectors have many zero components. The vectors of concern are the column vectors of a matrix.

Here is the algorithm. We want to solve $A\mathbf{x} = \mathbf{b}$. If we can decompose $A = LU$, where L is lower triangular and U is upper triangular, then we are done. All that is required is to solve $L\mathbf{y} = \mathbf{b}$ and then solve $U\mathbf{x} = \mathbf{y}$.

In order to find the LU decomposition of A, one can begin by setting L to be the identity matrix and U to be the original matrix A. At each stage of the algorithm on replaces L by LM^{-1} and U by MU, where M is a suitably chosen shear matrix.

The choice of M at the jth stage is the following. We take $M = I + \mathbf{z}\mathbf{e}_j$,

where $\mathbf{e}_j$ is the jth unit basis vector in the standard basis. We take $\mathbf{z}$ to have non-zero coordinates z_i only for index values $i > j$. Then M and M^{-1} are lower triangular matrices.

The goal is to try to choose M so that U will eventually become an upper triangular matrix. Let us apply M to the jth column $\mathbf{u}_j$ of the current U. Then we want to make $M\mathbf{u}_j$ equal to zero for indices larger than j. That is, one must make $u_{ij} + z_i u_{jj} = 0$ for $i > j$. Clearly this can be done, provided that the diagonal element $u_{jj} = 0$.

This algorithm with shear transformations only works if all of the diagonal elements turn out to be non-zero. This is somewhat more restrictive than merely requiring that the matrix A be non-singular.

Here is a program that implements the algorithm.

```
/* lusolve */

#include <stdio.h>
#include <stdlib.h>

typedef double     real;
typedef real *     vector;
typedef real **    matrix;

vector vec(int);
matrix mat(int,int);

void triangle(matrix, matrix, int);
void column(int, matrix, matrix, int);
void shear(int, matrix, matrix, int);
void solveltr(matrix, vector, vector, int);
void solveutr(matrix, vector, vector, int);

void fetchdim(int*);
void fetchvec(vector, int);
void fetchmat(matrix, int, int);
void displayvec(vector, int);
void displaymat(matrix, int, int);

int main()
{
int n;
vector b, x, y;
matrix a, l;
```

```
fetchdim(&n);
a = mat(n,n);
b = vec(n);
x = vec(n);
y = vec(n);
l = mat(n,n);

fetchmat(a,n,n);
displaymat(a,n,n);
fetchvec(b,n);
displayvec(b,n);

triangle(a,l,n);
displaymat(a,n,n);
displaymat(l,n,n);

solveltr(l,b,y,n);
displayvec(y,n);

solveutr(a,y,x,n);
displayvec(x,n);

return 0;
}
```

In *C* the vector and matrix data types may be implemented by pointers. These pointers must be told to point to available storage regions for the vector and matrix entries. That is the purpose of the following functions.

```
vector vec(int n)
{
vector x;
x = (vector) calloc(n+1, sizeof(real) );
return x ;
}

matrix mat(int m, int n)
{
int i;
matrix a;
a = (matrix) calloc(m+1, sizeof(vector) );
```

```
for (i = 1; i <= m ; i = i + 1)
     a[i] = vec(n);
return a ;
}
```

The actual work in producing the upper triangular matrix is done by the following procedure. The matrix `a` is supposed to become upper triangular while the matrix `l` remains lower triangular.

```
void triangle(matrix a, matrix l, int n)
{
int j;
for ( j=1 ;j<=n; j= j+1 )
{
column(j,a,l,n);
shear(j, a, l,n);
}
}
```

The `column` procedure computes the proper shear and stores it in the lower triangular matrix `l`.

```
void column(int j, matrix a, matrix l, int n)


{
int i;
for( i = j; i <= n; i = i+1)
l[i][j] = a[i][j] / a[j][j];
}
```

The shear procedure applies the shear to bring `a` closer to upper triangular form.

```
void shear(int j, matrix a, matrix l, int n)
{
int k, i;
for( k=j; k<= n ; k = k+1)
   for( i = j+1; i <= n; i = i + 1)
      a[i][k] = a[i][k] -  l[i][j] * a[j][k];
}
```

The actual solving of lower and upper triangular systems is routine.

```
void solveltr(matrix l, vector b, vector y, int n)
{
int i, j;
real sum;
for(i = 1; i <= n; i=i+1)
{
sum = b[i];
for( j= 1; j< i; j = j+1)
sum = sum - l[i][j]*y[j];
y[i] = sum;
}
}

void solveutr(matrix u, vector y, vector x, int n)
{
int i, j;
double sum;
for(i = n; i >=1; i=i-1)
{
sum = y[i];
for( j= i+1; j<= n;j = j+1)
sum = sum - u[i][j]*x[j];
x[i] = sum / u[i][i];
}
}
```

It would be nicer to have an algorithm that worked for an arbitrary non-singular matrix. Indeed the problem with zero diagonal elements can be eliminated by complicating the algorithm.

The idea is to decompose $A = PLU$, where P is a permutation matrix (obtained by permuting the rows of the identity matrix). Then to solve $A\mathbf{x} = \mathbf{b}$, one solves $LU\mathbf{x} = P^{-1}\mathbf{b}$ by the same method as before.

One can begin by setting P to be the identity matrix and L to be the identity matrix and U to be the original matrix A. The algorithm uses shears as before, but it is also allowed to use permutations when it is useful to get rid of zero or small diagonal elements.

Let R be a permutation that interchanges two rows. Then we replace P by PR^{-1}, L by RLR^{-1}, and U by RU. Then P remains a permutation matrix, L remains lower triangular, and U is modified to obtain a non-zero diagonal element in the appropriate place.

Projects

1. Write a program to multiply a matrix A (not necessarily square) times

a vector $\mathbf{x}$ to get an output vector $\mathbf{b} = A\mathbf{x}$.

Problems

1. Check the formula for the inverse of a shear.
2. Show that a shear has determinant one.
3. Describe the geometric action of a shear in two dimensions. Why is it called a shear?
4. Consider a transformation of the form $M = I + \mathbf{z}\mathbf{w}^T$, but do not assume that $\mathbf{w}^T\mathbf{z} = 0$. When does this have an inverse? What is the formula for the inverse?

2.2 Reflections

Gaussian elimination with LU decomposition is not the only technique for solving equations. The QR method is also worth consideration.

The goal is to write an arbitrary matrix $A = QR$, where Q is an orthogonal matrix and R is an upper triangular matrix. Recall that an orthogonal matrix is a matrix Q with $Q^TQ = I$.

Thus to solve $A\mathbf{x} = \mathbf{b}$, one can take $\mathbf{y} = Q^T\mathbf{b}$ and solve $R\mathbf{x} = \mathbf{y}$.

We can define an *inner product* of vectors $\mathbf{x}$ and $\mathbf{y}$ by $\mathbf{x} \cdot \mathbf{y} = \mathbf{x}^T\mathbf{y}$. We say that $\mathbf{x}$ and $\mathbf{y}$ are *perpendicular* or *orthogonal* if $\mathbf{x} \cdot \mathbf{y} = 0$.

The Euclidean *length* (or *norm*) of a vector $\mathbf{x}$ is $|\mathbf{x}| = \sqrt{\mathbf{x} \cdot \mathbf{x}}$. A *unit vector* $\mathbf{u}$ is a vector with length one: $|\mathbf{u}| = 1$.

A *reflection* P is a linear transformation of the form $P = I - 2\mathbf{u}\mathbf{u}^T$, where $\mathbf{u}$ is a unit vector. The action of a reflection on a vector perpendicular to $\mathbf{u}$ is to leave it alone. However a $\mathbf{x} = c\mathbf{u}$ vector parallel to $\mathbf{u}$ is sent to its negative.

It is easy to check that a reflection is an orthogonal matrix. Furthermore, if P is a reflection, then $P^2 = I$, so P is its own inverse.

Consider the problem of finding a reflection that sends a given non-zero vector $\mathbf{a}$ to a multiple of another given unit vector $\mathbf{b}$. Since a reflection preserves lengths, the other vector must be $\pm|\mathbf{a}|\mathbf{b}$.

Take $\mathbf{u} = c\mathbf{w}$, where $\mathbf{w} = \mathbf{a} \pm |\mathbf{a}|\mathbf{b}$, and where c is chosen to make $\mathbf{u}$ a unit vector. Then $c^2\mathbf{w} \cdot \mathbf{w} = 1$. It is easy to check that $\mathbf{w} \cdot \mathbf{w} = 2\mathbf{w} \cdot \mathbf{a}$. Furthermore,

$$P\mathbf{a} = \mathbf{a} - 2c^2\mathbf{w}\mathbf{w}^T \cdot \mathbf{a} = \mathbf{a} - \mathbf{w} = \mp|\mathbf{a}|\mathbf{b}. \tag{2.1}$$

Which sign should we choose? We clearly want $\mathbf{w} \cdot \mathbf{w} > 0$, and to avoid having to choose a large value of c we should take it as large as possible.

However $\mathbf{w} \cdot \mathbf{w} = 2\mathbf{a} \cdot \mathbf{a} \pm 2|\mathbf{a}|\mathbf{b} \cdot \mathbf{a}$. So we may as well choose the sign so that $\pm\mathbf{b} \cdot \mathbf{a} \geq 0$.

Now the goal is to use successive reflections in such a way that $P_n \cdots P_1 A = R$. This gives the $A = QR$ decomposition with $Q = P_1 \cdots P_n$.

One simply proceeds through the columns of A. Fix the column j. Apply the reflection to send the vector a_{ij} for $j \leq i \leq n$ to a vector that is non-zero in the jj place and zero in the ij place for $j \leq i \leq n$.

We have assumed up to this point that our matrices were square matrices, so that there is some hope that the upper triangular matrix R can be inverted. However we can also get a useful result for systems where there are more equations than unknowns. This corresponds to the case when A is an m by n matrix with $m > n$. Take $\mathbf{b}$ to be an m dimensional vector. The goal is to solve the least-squares problem of minimizing $|A\mathbf{x} - \mathbf{b}|$ as a function of the n dimensional vector $\mathbf{x}$.

In that case we write $Q^T A = R$, where Q is m by m and R is m by n. We cannot solve $A\mathbf{x} = \mathbf{b}$. However if we look at the difference $A\mathbf{x} - \mathbf{b}$ we see that

$$|A\mathbf{x} - \mathbf{b}| = |R\mathbf{x} - \mathbf{y}|, \tag{2.2}$$

where $\mathbf{y} = Q^T\mathbf{x}$.

We can try to choose $\mathbf{x}$ to minimize this quantity. This can be done by solving an upper triangular system to make the first n components of $R\mathbf{x}-\mathbf{y}$ equal to zero. (Nothing can be done with the other $m-n$ components, since $R\mathbf{x}$ automatically has these components equal to zero.)

The computer implementation of the QR algorithm is not much more complicated than that for the LU algorithm. The work is done by a triangulation procedure. It goes throught the columns of the matrix `a` and finds the suitable unit vectors, which it stores in another matrix `h`. There is never any needed to actually compute the orthogonal part of the decomposition, since the unit vectors for all of the reflections carry the same information.

```
void triangle(matrix a, matrix h, int m, int n)
{
int j ;
for ( j=1;j<=n; j= j+1)
{
select(j,a,h,m) ;
reflm(j, a, h,m,n) ;
}
}
```

The select procedure does that calculation to determine the unit vector that is appropriate to the given column.

```
void select(int j, matrix a, matrix h, int m)
{
int i ;
real norm , sign;
norm = 0.0 ;
for( i = j; i <=m; i = i+1)
norm = norm + a[i][j] * a[i][j] ;
norm = sqrt(norm) ;
if ( a[j][j] >= 0.0)
sign = 1.0;
else
sign = -1.0;
h[j][j] = a[j][j] + sign * norm ;
for( i = j+1; i <= m; i = i+1)
h[i][j] = a[i][j] ;
norm = 2 * norm * ( norm + fabs( a[j][j] ) );
norm = sqrt(norm) ;
for( i = j; i <= m; i = i+1)
h[i][j] = h[i][j] / norm ;
}
```

The reflect matrix procedure applies the reflections to the appropriate column of the matrix.

```
void reflm(int j, matrix a, matrix h, int m, int n)
{
int k, i ;
real scalar ;
for( k=j; k<= n ; k = k+1)
{
scalar = 0.0 ;
for( i = j; i <= m; i = i + 1)
scalar = scalar + h[i][j] * a[i][k] ;
for( i = j; i <= m; i = i + 1)
a[i][k] = a[i][k] - 2 * h[i][j] * scalar ;
}
}
```

In order to use this to solve an equation one must apply the same reflections to the right hand side of the equation. Finally, one must solve the resulting triangular system.

Projects

1. Implement the QR algorithm for solving systems of equations and for solving least-squares problems.

2. Consider the 6 by 6 matrix with entries $a_{ij} = 1/(i + j^2)$. Use your QR program to find the first column of the inverse matrix.

3. Consider the problem of getting the best least-squares approximation of $1/(1 + x)$ by a linear combination of 1, x, and x^2 at the points 1, 2, 3, 4, 5, and 6. Solve this 6 by 3 least squares problem using your program.

Problems

1. Show that if A and B are invertible matrices, then $(AB)^{-1} = B^{-1}A^{-1}$.

2. Show that if A and B are matrices, then $(AB)^T = B^T A^T$.

3. Show that an orthogonal matrix preserves the inner product in the sense that $Q\mathbf{x} \cdot Q\mathbf{y} = \mathbf{x} \cdot \mathbf{y}$.

4. Show that an orthogonal matrix preserves length: $|Q\mathbf{x}| = |\mathbf{x}|$.

5. Show that the product of orthogonal matrices is an orthogonal matrix. Show that the inverse of an orthogonal matrix is an orthogonal matrix.

6. What are the possible values of the determinant of an orthogonal matrix? Justify your answer.

7. An orthogonal matrix with determinant one is a rotation. Show that the product of two reflections is a rotation.

8. How is the angle of rotation determined by the angle between the unit vectors determining the reflection?

2.3 Vectors and matrices in C

2.3.1 Pointers in C

Pointer types

The variables of a certain type T correspond to a linearly ordered set of *pointer to* T values. In a computer implementation the pointer to T values are realized as addresses of memory locations.

Each pointer value determines a unique variable of type T. In other words, pointer to T values correspond to variables of type T. Thus there are whole new families of pointer types.

Example: `float *` is the type pointer to float.

Example: `float **` is the type pointer to pointer to float.

One can have variables whose values are pointers.

Example: Consider a variable `x` of type float. One can have a variable `p` of type pointer to float. One possible value of `p` would be a pointer to `x`. In this case the corresponding variable is `x`.

Example: `float x, *p, **m ;` declares the variables `x`, `p`, and `m` to be of types float, pointer to float, and pointer to pointer to float.

2.3.2 Pointer Expressions

Indirection

The operator `&` takes a *variable* (or function) and returns its corresponding *pointer*. If the variable or function has type T, the result has type pointer to T.

The other direction is given by the *indirection* or *dereferencing* operator `*`. Applying `*` to an *pointer value* gives the *variable* (or function) corresponding to this value. This operator can only be applied to pointer types. If the value has type *pointer to* T, then the result has type T.

Example: Assume that `p` is a variable of type pointer to float and that its value is a pointer to `x`. Then `*p` is the same variable as `x`.

Example: Assume that `m` is a variable of type pointer to pointer to float. The expression `*m` can be a variable whose values are pointer to float. The expression `**m` can be a variable whose values are float.

Example: If `f` is a function pointer variable with some function pointer value, then `*f` is a function corresponding to this value. The value of this function is the pointer value, so `(*f)(x)` is the same as `f(x)`.

Pointer arithmetic

Let T be a type that is not a function type. For an integer `i` and a pointer value `p` we have another pointer value `p+i`. This is the pointer value associated with the `i`th variable of this type past the variable associated with the pointer value `p`.

Incrementing the pointer to T value by `i` corresponds to incrementing the address by `i` times the size of a T value.

The fact that variables of type T may correspond to a linearly ordered set of pointer to T values makes C useful for models where a linear strucure is important.

When a pointer value `p` is incremented by the integer amount `i`, then `p+i` is a new pointer value. We use `p[i]` as a synonym for `*(p+i)`. This is the variable pointed to by `p+i`.

Example: Assume that `v` has been declared `float *v`. If we think of `v` as pointing to an entry of a vector, then `v[i]` is the entry `i` units above it.

Example: Assume that `m` has been declared `float **m`. Think of `m` as pointing to a row pointer of a matrix, which in turn points to an entry of the matrix. Then `m[i]` points to an entry in the row `i` units above the original row in row index value. Furthermore `m[i][j]` is the entry `j` units above this entry in column index value.

Function calls

In C function calls it is always a value that is passed. If one wants to give a function access to a variable, one must pass the value of the pointer corresponding to the variable.

Example: A procedure to fetch a number from input is defined with the heading `void fetch(float *p)`. A call `fetch(&x)` copies the argument, which is the pointer value corresponding to `x`, onto the the parameter, which is the pointer variable `p`. Then `*p` and `x` are the same float variable, so an assignment to `*p` can change the value of `x`.

Example: A procedure to multiply a scalar `x` times a vector given by `w` and put the result back in the same vector is defined with the heading `void mult(float x, float *w)`. Then a call `mult(a,v)` copies the values of the arguments `a` and `v` onto the parameters `x` and `w`. Then `v` and `w` are pointers with the same value, and so `v[i]` and `w[i]` are the same float variables. Therefore an assignment statement `w[i] = x * w[i]` in the body of the procedure has the effect of changing the value of `v[i]`.

Memory allocation

There is a cleared memory allocation function named `calloc` that is very useful in working with pointers. It returns a pointer value corresponding to the first of a specified number of variables of a specified type.

The `calloc` function does not work with the actual type, but with the size of the type. In an implementation each data type (other than function) has a size. The size of a data type may be recovered by the `sizeof( )` operator. Thus `sizeof( float )` and `sizeof( float * )` give numbers that represent the amount of memory needed to store a float and the amount of memory need to store a pointer to float.

The function call `calloc( n, sizeof( float ) )` returns a pointer to `void` corresponding to the first of `n` possible float variables. The cast operator `(float *)` converts this to a pointer to `float`. If a pointer variable v has been declared with `float *v ;` then

```
v = (float *) calloc( n, sizeof (float) ) ;
```

assigns this pointer to `v`. After this assignment it is legitimate to use the variable `v[i]` of type float, for `i` between `0` and `n-1`.

Example: One can also create space for a matrix in this way. The assignment statement

```
m = (float **) calloc( m, sizeof (float *) ) ;
```

creates space for the row pointers and assigns the pointer to the first row pointer to `m`, while

```
m[i] = (float *) calloc( n, sizeof (float) ) ;
```

creates space for a row and assigns the pointer to the first entry in the row to `m[i]`. After these assignments we have float variables `m[i][j]` available.

Chapter 3

Eigenvalues

3.1 Introduction

A square matrix can be analyzed in terms of its eigenvectors and eigenvalues. In this chapter we review this theory and approach the problem of numerically computing eigenvalues.

If A is a square matrix, $\mathbf{x}$ is a vector not equal to the zero vector, and λ is a number, then the equation

$$A\mathbf{x} = \lambda\mathbf{x} \tag{3.1}$$

says that λ is an eigenvalue with eigenvector $\mathbf{x}$.

We can also identify the eigenvalues as the set of all λ such that $\lambda I - A$ is not invertible.

We now begin an abbreviated review of the relevant theory. We begin with the theory of general bases and similarity. We then treat the theory of orthonormal bases and orthogonal similarity.

3.2 Similarity

If we have an n by n matrix A and a basis consisting of n linearly independent vectors, then we may form another matrix S whose columns consist of the vectors in the basis. Let $\hat{A}$ be the matrix of A in the new basis. Then $AS = S\hat{A}$. In other words, $\hat{A} = S^{-1}AS$ is *similar* to A.

Similar matrices tend to have similar geometric properties. They always have the same eigenvalues. They also have the same determinant and trace. (Similar matrices are not always identical in their geometrical properties; similarity can distort length and angle.)

We would like to pick the basis to display the geometry. The way to do this is to use eigenvectors as basis vectors, whenever possible.

If the dimension n of the space of vectors is odd, then a matrix always has at least one real eigenvalue. If the dimension is even, then there may be no real eigenvalues. (Example: a rotation in the plane.) Thus it is often helpful to allow complex eigenvalues and eigenvectors. In that case the typical matrix will have a basis of eigenvectors.

If we can take the basis vectors to be eigenvectors, then the matrix $\hat{A}$ in this new basis is diagonal.

There are exceptional cases where the eigenvectors do not form a basis. (Example: a shear.) Even in these exceptional cases there will always be a new basis in which the matrix is triangular. The eigenvalues will appear (perhaps repeated) along the diagonal of the triangular matrix, and the determinant and trace will be the product and sum of these eigenvalues.

We now want to look more closely at the situation when a matrix has a basis of eigenvectors.

We say that a collection of vectors is *linearly dependent* if one of the vectors can be expressed as a linear combination of the others. Otherwise the collection is said to be linearly independent.

Proposition 3.2.1 *If $\mathbf{x}_i$ are eigenvectors of A corresponding to distinct eigenvalues λ_i, then the $\mathbf{x}_i$ are linearly independent.*

Proof: The proof is by induction on k, the number of vectors. The result is obvious when $k = 1$. Assume it is true for $k - 1$. Consider the case of k vectors. We must show that it is impossible to express one eigenvector as a linear combination of the others. Otherwise we would have $\mathbf{x}_j = \sum_{i \neq j} c_i \mathbf{x}_i$ for some j. If we apply $A - \lambda_j I$ to this equation, we obtain $0 = \sum_{i \neq j} c_i (\lambda_i - \lambda_j) \mathbf{x}_i$. If $c_i \neq 0$ for some $i \neq j$, then we could solve for $\mathbf{x}_i$ in terms of the other $k - 2$ vectors. This would contradict the result for $k - 1$ vectors. Therefore $c_i = 0$ for all $i \neq j$. Thus $\mathbf{x}_j = 0$, which is not allowed. □

If we have n independent eigenvectors, then we can put the eigenvectors as columns of a matrix X. Let Λ be the diagonal matrix whose entries are the corresponding eigenvalues. Then we may express the eigenvalue equation as

$$AX = X\Lambda. \tag{3.2}$$

Since X is an invertible matrix, we may write this equation as

$$X^{-1}AX = \Lambda. \tag{3.3}$$

This says that A is similar to a diagonal matrix.

Theorem 3.2.1 *Consider an n by n matrix with n distinct (possibly complex) eigenvalues λ_i. Then the corresponding (possibly complex) eigenvectors $\mathbf{x}_i$ form a basis. The matrix is thus similar to a diagonal matrix.*

Let Y be a matrix with column vectors $\mathbf{y}_i$ determined in such a way that $Y^T = X^{-1}$. Then $Y^T A X = \Lambda$ and so $A = X \Lambda Y^T$. This leads to the following *spectral representation.*

Let $\mathbf{y}_i$ be the dual basis defined by $\mathbf{y}_i^T \mathbf{x}_j = \delta_{ij}$. Then we may represent

$$A = \sum_i \lambda_i \mathbf{x}_i \mathbf{y}_i^T . \tag{3.4}$$

It is worth thinking a bit more about the meaning of the complex eigenvalues. It is clear that if A is a real matrix, then the eigenvalues that are not real occur in complex conjugate pairs. The reason is simply that the complex conjugate of the equation $A\mathbf{x} = \lambda \mathbf{x}$ is $A\bar{\mathbf{x}} = \bar{\lambda}\bar{\mathbf{x}}$. If λ is not real, then we have a pair $\lambda \neq \bar{\lambda}$ of complex conjugate eigenvalues.

We may write $\lambda = a + ib$ and $\mathbf{x} = \mathbf{u} + i\mathbf{v}$. Then the equation $A\mathbf{x} = \lambda \mathbf{x}$ becomes the two real equations $A\mathbf{u} = a\mathbf{u} - b\mathbf{v}$ and $A\mathbf{v} = b\mathbf{u} + a\mathbf{v}$. The vectors $\mathbf{u}$ and $\mathbf{v}$ are no longer eigenvectors, but they can be used as part of a real basis. In this case instead of two complex conjugate diagonal entries one obtains a two by two matrix that is a multiple of a rotation matrix.

Thus geometrically a typical real matrix is constructed from stretches, shrinks, and reversals (from the real eigenvalues) and from stretches, shrinks, and rotations (from the conjugate pair non-real eigenvalues).

Problems

1. Find the eigenvalues of the 2 by 2 matrix whose first row is 0, -3 and whose second row is -1, 2. Find eigenvectors. Find the similarity transformation and show that it takes the matrix to diagonal form.

2. Find the spectral representation for the matrix of the previous problem.

3. Consider a rotation by angle θ in the plane. Find its eigenvalues and eigenvectors.

4. Give an example of two matrices with the same eigenvalues that are not similar.

5. Show how to express the function $\mathrm{tr}(zI - A)^{-1}$ of complex z in terms of the numbers $\mathrm{tr}A^n$, $n = 1, 2, 3, \ldots$.

6. Show how to express the eigenvalues of A in terms of $\mathrm{tr}(zI - A)^{-1}$.

7. Let $\mathbf{z}\mathbf{w}^T$ and $\mathbf{z}'\mathbf{w}'^T$ be two one-dimensional projections. When is their product zero? If the product is zero in one order, must it be zero in the other order?

8. Show that for arbitrary square matrices $\mathrm{tr}AB = \mathrm{tr}BA$.

9. Show that $\mathrm{tr}(AB)^n = \mathrm{tr}(BA)^n$.

10. Show that if B is non-singular, then AB and BA are similar.

11. Show that if AB and BA always have the same eigenvalues, even if both of them are singular.

12. Give an example of square matrices A and B such that AB is not similar to BA.

3.3 Orthogonal similarity

3.3.1 Symmetric matrices

If we have an n by n matrix A and an orthonormal basis consisting of n orthogonal unit vectors, then as before we may form another matrix Q whose columns consist of the vectors in the basis. Let $\hat{A}$ be the matrix of A in the new basis. Then again $\hat{A} = Q^{-1}AQ$ is similar to A. However in this special situation Q is orthogonal, that is, $Q^{-1} = Q^T$. In this case we say that $\hat{A}$ is *orthogonal similar* to A.

The best of worlds is the case of a symmetric real matrix A.

Theorem 3.3.1 *For a symmetric real matrix A the eigenvalues are all real, and there is always a basis of eigenvectors. Furthermore, these eigenvectors may be taken to form an orthonormal basis. With this choice the matrix Q is orthogonal, and $\hat{A} = Q^{-1}AQ$ is diagonal.*

3.3.2 Singular values

It will be useful to have the observation that for a real matrix A the matrix A^TA is always a symmetric real matrix. It is easy to see that it must have positive eigenvalues $\sigma_i^2 \geq 0$. Consider the positive square roots $\sigma_i \geq 0$. These are called the *singular values* of the original matrix A. It is not difficult to see that two matrices that are orthogonally equivalent have the same singular values.

We may define the positive square root $\sqrt{A^TA}$ as the matrix with the same eigenvectors as A^TA but with eigenvalues σ_i. We may think of $\sqrt{A^TA}$ as a matrix that is in some sense the absolute value of A.

Of course one could also look at AA^T and its square root, and this would be different in general. We shall see, however, that these matrices are always orthogonal similar, so in particular the eigenvalues are the same.

To this end, we use the following *polar decomposition.*

Proposition 3.3.1 *Let A be a real square matrix. Then $A = Q\sqrt{A^T A}$, where Q is orthogonal.*

This amounts to writing the the matrix as the product of a part that has absolute value one with a part that represents its absolute value. Of course here the absolute value one part is an orthogonal matrix and the absolute value part is a symmetric matrix.

Here is how this can be done. We can decompose the space into the orthogonal sum of the range of A^T and the nullspace of A. This is the same as the orthogonal sum of the range of $\sqrt{A^T A}$ and the nullspace of $\sqrt{A^T A}$. The range of A^T is the part where the absolute value is non-zero. On this part the unit size part is determined; we must define Q on $\mathbf{x} = \sqrt{A^T A}\mathbf{y}$ in the range in such a way as to have $Q\mathbf{x} = Q\sqrt{A^T A}\mathbf{y} = A\mathbf{y}$. Then $|Q\mathbf{x}| = |A\mathbf{y}| = |\mathbf{x}|$, so Q sends the range of A^T to the range of A and preserves lengths on this part of the space. However on the nullspace of A the unit size part is arbitrary. But we can also decompose the space into the orthogonal sum of the range of A and the nullspace of A^T. Since the nullspaces of A and A^T have the same dimension, we can define Q on the nullspace of A to be an arbitrary orthogonal transformation that takes it to the nullspace of A^T.

We see from $A = Q\sqrt{A^T A}$ that $AA^T = QA^T AQ^T$. Thus AA^T is similar to $A^T A$ by the orthogonal matrix Q. The two possible notions of absolute value are geometrically equivalent, and the two possible notions of singular value coincide.

3.3.3 The Schur decomposition

We now consider a real matrix A that has only real eigenvalues. Then this matrix is similar to an upper triangular matrix, that is, $AX = X\hat{A}$, where $\hat{A}$ is upper triangular.

In the general situation the vectors $\mathbf{x}_i$ that constitute the columns of X may not be orthogonal. However we may produce a family $\mathbf{q}_i$ of orthogonal vectors, each of norm one, such that for each k the subspace spanned by $\mathbf{x}_1, \ldots, \mathbf{x}_k$ is the same as the subspace spanned by $\mathbf{q}_1, \ldots, \mathbf{q}_k$.

Let Q be the matrix with columns formed by the vectors $\mathbf{q}_i$. This

condition may be expressed by

$$X_{ik} = \sum_{j \le k} R_{jk} Q_{ij}. \tag{3.5}$$

In other words, $X = QR$, where Q is orthogonal and R is upper triangular.

From this equation we may conclude that $R^{-1}Q^{-1}AQR = \hat{A}$, or

$$Q^{-1}AQ = U, \tag{3.6}$$

where Q is orthogonal and $U = R\hat{A}R^{-1}$ is upper triangular. This is called the *Schur decomposition.*

Theorem 3.3.2 *Let A be a real matrix with only real eigenvalues. Then A is orthogonal similar to an upper triangular matrix U.*

The geometrical significance of the Schur decomposition may be seen as follows. Let V_r be the subspace spanned by column vectors that are non-zero only in their first r components. Then we have $AQV_r = QUV_r$. Since UV_r is contained in V_r, it follows that QV_r is an r-dimensional subspace invariant under the matrix A that is spanned by the first r column vectors of Q.

Problems

1. Consider the symmetric 2 by 2 matrix whose first row is 2, 1 and whose second row is 1, 2. Find its eigenvalues. Find the orthogonal similarity that makes it diagonal. Check that it works.

2. Find the spectral decomposition in this case.

3. Find the eigenvalues of the symmetric 3 by 3 matrix whose first row is 2, 1, 0 and whose second row is 1, 3, 1 and whose third row is 0, 1, 4. (Hint: One eigenvalue is an integer.) Find the eigenvectors and check orthogonality.

4. Find the singular values of the matrix whose first row is 0, -3 and whose second row is -1, 2.

5. Find a Schur decomposition of the matrix in the preceding problem.

6. Give an example of two matrices that are similar by an invertible matrix, but cannot be made similar by an orthogonal matrix.

7. Show that an arbitrary A may be written $A = Q_1 D Q_2$, where D is a diagonal matrix with positive entries and Q_1 and Q_2 are orthogonal matrices.

3.4 Vector and matrix norms

3.4.1 Vector norms

We shall use three vector norms. The first is the 1-norm

$$|\mathbf{x}|_1 = \sum_{i=1}^{n} |x_i|. \tag{3.7}$$

The second is the 2-norm

$$|\mathbf{x}|_2 = \sqrt{\sum_{i=1}^{n} |x_i|^2}. \tag{3.8}$$

The final one is the ∞-norm

$$|\mathbf{x}|_\infty = \max_{1\le i\le n} |x_i|. \tag{3.9}$$

They are related by the inequalities

$$|\mathbf{x}|_\infty \le |\mathbf{x}|_2 \le |\mathbf{x}|_1 \le n|\mathbf{x}|_\infty. \tag{3.10}$$

3.4.2 Associated matrix norms

There are three matrix norms associated with the three vector norms. These are given for $p = 1, 2, \infty$ by

$$\|A\|_p = \min\{M \mid |A\mathbf{x}|_p \le M|\mathbf{x}|_p\}. \tag{3.11}$$

Here are the explicit forms. The 1-norm is easy to compute.

$$\|A\|_1 = \max_{1\le j\le n} \sum_{i=1}^{m} |a_{ij}|. \tag{3.12}$$

The 2-norm is the difficult one.

$$\|A\|_2 = \max_{1\le i\le n} \sigma_i = \sigma_{\max}, \tag{3.13}$$

where $\sigma_i \ge 0$ are the singular values of A.

The ∞-norm is just as easy as the 1-norm.

$$\|A\|_\infty = \max_{1\le i\le m} \sum_{j=1}^{n} |a_{ij}|. \tag{3.14}$$

The ∞ and 1 norms are related by $\|A\|_\infty = \|A^T\|_1$. For the 2-norm we have the important relation $\|A\|_2 = \|A^T\|_2$.

There is a very useful *interpolation bound* relating the 2-norm to the other norms.

Proposition 3.4.1

$$\|A\|_2 \le \sqrt{\|A\|_1 \|A\|_\infty}. \tag{3.15}$$

3.4.3 Singular value norms

It is sometime useful to define other norms in terms of singular values. Here are three such norms defined in terms of the singular values $\sigma_i \ge 0$ of A (where the σ_i^2 are eigenvalues of $A^T A$.) We distinguish these from the previous definitions by the use of a square bracket. (This is not standard notation.)

The first is the *trace norm*

$$\|A\|_{[1]} = \mathrm{tr}(\sqrt{A^T A}) = \sum_{i=1}^{n} \sigma_i, \tag{3.16}$$

This is difficult to compute, because of the square root.

The second is the *Hilbert-Schmidt* norm

$$\|A\|_{[2]} = \sqrt{\mathrm{tr}(A^T A)} = \sqrt{\sum_{i=1}^{n} \sigma_i^2}. \tag{3.17}$$

This one is easy to compute.

The final one is the *uniform* norm

$$\|A\|_{[\infty]} = \max_{1 \le i \le n} \sigma_i. \tag{3.18}$$

This again is difficult to compute.

They are related by the inequalities

$$\|A\|_{[\infty]} \le \|A\|_{[2]} \le \|A\|_{[1]} \le n \|A\|_{[\infty]}. \tag{3.19}$$

Why are these norms useful? Maybe the main reason is that $\|A\|_{[\infty]} = \|A\|_2$, and so

$$\|A\|_2 \le \|A\|_{[2]}. \tag{3.20}$$

This gives a useful upper bound that complements the interpolation bound.

3.4.4 Eigenvalues and norms

From now on we deal with one of the norms $\|A\|_p$ and denote it by $\|A\|$. The fundamental relation between norms and eigenvalues is that every eigenvalue λ of A satisfies $|\lambda| \leq \|A\|$. This is an equality for symmetric matrices. However in general it is not such an accurate result. The following is often a much better bound.

Theorem 3.4.1 *Every eigenvalue λ of A satisfies the inequality*

$$|\lambda| \leq \|A^n\|^{\frac{1}{n}} \tag{3.21}$$

for every $n = 1, 2, 3, \ldots$.

3.4.5 Condition number

Let A be an invertible square matrix. Consider one of the p-norms $\|A\|_p$, where p is 1, 2, or ∞. In this section we shall abbreviate this as $\|A\|$. We are most interested in the case $p = 2$. Unfortuately, this is the case when it is most difficult to compute the norm.

We want to measure how far A is from being invertible. The standard measure is

$$\mathrm{cond}(A) = \|A\|\|A^{-1}\|. \tag{3.22}$$

When this is not too much larger than one, then the matrix is well-conditioned, in the sense that calculations with it are not too sensitive to perturbations (small errors). (When the number is very large, then the matrix may be ill-conditioned, that is, extremely sensitive to perturbations.)

In the case of the 2-norm this condition number has a simple interpretation. Let σ_i^2 be the eigenvalues of $A^T A$. Then

$$\mathrm{cond}(A) = \frac{\sigma_{\max}}{\sigma_{\min}}. \tag{3.23}$$

Problems

1. Evaluate each of the six matrix norms for the two-by-two matrix whose first row is 0, -3 and whose second row is -1, 2.

2. In the preceding problem, check the interpolation bound.

3. In the preceding problem, check the Hilbert-Schmidt bound.

4. In the preceding problem, check the bound on the eigenvalues for $n = 1, 2, 3$ and for each of the three p norms.

5. Give an example of a matrix A for which the eigenvalue λ of largest absolute value satisfies $|\lambda| < \|A\|$ but $|\lambda| = \|A^n\|^{1/n}$ for some n.

6. Prove the assertions about the concrete forms of the p-norms $\|A\|_p$, for $p = 1, 2, \infty$.

7. Prove that the 2-norm of a matrix is the 2-norm of its transpose.

3.5 Stability

3.5.1 Inverses

We next look at the stability of the inverse under perturbation. The fundamental result is the following.

Proposition 3.5.1 *Assume that the matrix A has an inverse A^{-1}. Let E be another matrix. Assume that E is small relative to A in the sense that $\|E\| < 1/\|A^{-1}\|$. Let $\hat{A} = A - E$. Then $\hat{A}$ has an inverse $\hat{A}^{-1}$, and*

$$A^{-1} - \hat{A}^{-1} = -A^{-1}E\hat{A}^{-1}. \tag{3.24}$$

Proof: Assume that $(A - E)\mathbf{x} = 0$. Then $\mathbf{x} = A^{-1}A\mathbf{x} = A^{-1}E\mathbf{x}$. Hence $|\mathbf{x}| \leq \|A^{-1}\|\|E\||\mathbf{x}|$. Thus $|\mathbf{x}| = 0$, so $\mathbf{x}$ is the zero vector. This proves that $\hat{A} = A - E$ is invertible. The identity relating A^{-1} and $\hat{A}^{-1}$ follows by algebraic manipulation. □

We may write the hypothesis of the theorem in terms of the relative size of the perturbation as $\|E\|/\|A\| < 1/\mathrm{cond}(A)$. Thus for an ill-conditioned matrix, one can only take very small relative perturbations.

Furthermore, we may deduce that

$$\|A^{-1} - \hat{A}^{-1}\| \leq \|A^{-1}\||E\|\|\hat{A}^{-1}\| \tag{3.25}$$

which says that

$$\|A^{-1} - \hat{A}^{-1}\|/\|\hat{A}^{-1}\| \leq \mathrm{cond}(A)\|E\|/\|A\|. \tag{3.26}$$

Relative changes in matrices are controlled by condition numbers.

3.5.2 Iteration

Sometimes one wants to solve the equation $A\mathbf{x} = \mathbf{b}$ by iteration. A natural choice of fixed point function is

$$\mathbf{g}(\mathbf{x}) = \mathbf{x} + C(\mathbf{b} - A\mathbf{x}). \tag{3.27}$$

Here C can be an arbitrary non-singular matrix, and the fixed point will be a solution. However for convergence we would like C to be a reasonable guess of or approximation to A^{-1}.

When this is satisfied we may write

$$\mathbf{g}(\mathbf{x}) - \mathbf{g}(\mathbf{y}) = (I - CA)(\mathbf{x} - \mathbf{y}) = (A^{-1} - C)A(\mathbf{x} - \mathbf{y}). \tag{3.28}$$

Then if $\|A^{-1} - C\|\|A\| < 1$, the iteration function is guaranteed to shrink the iterates together to a fixed point.

If we write the above condition in terms of relative error, it becomes $\|A^{-1} - C\|/\|A^{-1}\| < 1/\mathrm{cond}(A)$. Again we see that for an ill-conditioned matrix one must make a good guess of the inverse.

3.5.3 Eigenvalue location

Let A be a square matrix, and let D be the diagonal matrix with the same entries as the diagonal entries of A. If all these entries are non-zero, then D is invertible. We would like to conclude that A is invertible.

Write $A = DD^{-1}A$. The matrix $D^{-1}A$ has matrix entries a_{ij}/a_{ii} so it has ones on the diagonal. Thus we may treat it as a perturbation of the identity matrix. Thus we may write $D^{-1}A = I - (I - D^{-1}A)$, where $I - D^{-1}A$ has zeros on the diagonal and entries $-a_{ij}/a_{ii}$ elsewhere. We know from our perturbation lemma that if $I - D^{-1}A$ has norm strictly less than one, then $D^{-1}A$ is invertible, and so A is invertible.

The norm that is most convenient to use is the ∞ norm. The condition for $I - D^{-1}A$ to have ∞ norm strictly less than one is that $\max_i \sum_{j \neq i} \frac{|a_{ij}|}{|a_{ii}|} < 1$. We have proved the following result on *diagonal dominance*.

Proposition 3.5.2 *If a matrix A satisfies for each i*

$$\sum_{j \neq i} |a_{ij}| < |a_{ii}|, \tag{3.29}$$

then A is invertible.

Let B be an arbitrary matrix and let λ be a number. Apply this result to the matrix $\lambda I - B$. Then λ is an eigenvalue of B precisely when $\lambda I - B$ is not invertible. This gives the following conclusion.

Corollary 3.5.1 *If λ is an eigenvalue of B, then for some i the eigenvalue λ satisfies*

$$|\lambda - b_{ii}| \leq \sum_{j \neq i} |b_{ij}|. \tag{3.30}$$

The intervals about b_{ii} in the corollary are known as *Gershgorin's disks.*

Problems

1. Assume $A\mathbf{x} = \mathbf{b}$. Assume that there is a computed solution $\hat{\mathbf{x}} = \mathbf{x} - \mathbf{e}$, where $\mathbf{e}$ is an error vector. Let $A\hat{\mathbf{x}} = \hat{\mathbf{b}}$, and define the residual vector $\mathbf{r}$ by $\hat{\mathbf{b}} = \mathbf{b} - \mathbf{r}$. Show that $|\mathbf{e}|/|\mathbf{x}| \leq \text{cond}(A)|\mathbf{r}|/|\mathbf{b}|$.

2. Assume $A\mathbf{x} = \mathbf{b}$. Assume that there is an error in the matrix, so that the matrix used for the computation is $\hat{A} = A - E$. Take the computed solution as $\hat{\mathbf{x}}$ defined by $\hat{A}\hat{\mathbf{x}} = \mathbf{b}$. Show that $|\mathbf{e}|/|\hat{\mathbf{x}}| \leq \text{cond}(A)\|E\|/\|A\|$.

3. Find the Gershgorin disks for the three-by-three matrix whose first row is 1, 2, -1, whose second row is 2, 7, 0, and whose third row is -1, 0, -5.

3.6 Power method

We turn to the computational problem of finding eigenvalues of the square matrix A. We assume that A has distinct real eigenvalues. The power method is a method of computing the *dominant eigenvalue* (the eigenvalue with largest absolute value).

The method is to take a more or less arbitrary starting vector $\mathbf{u}$ and compute $A^k\mathbf{u}$ for large k. The result should be approximately the eigenvector corresponding to the dominant eigenvalue.

Why does this work? Let us assume that there is a dominant eigenvalue and call it λ_1. Let $\mathbf{u}$ be a non-zero vector. Expand $\mathbf{u} = \sum_i c_i \mathbf{x}_i$ in the eigenvectors of A. Assume that $c_1 \neq 0$. Then

$$A^k\mathbf{u} = \sum_i c_i \lambda_i^k \mathbf{x}_i. \tag{3.31}$$

When k is large, the term $c_1\lambda_1^k\mathbf{x}_i$ is so much larger than the other terms that $A^k\mathbf{u}$ is a good approximation to a multiple of $\mathbf{x}_1$.

[We can write this another way in terms of the spectral representation. Let $\mathbf{u}$ be a non-zero vector such that $\mathbf{y}_1^T\mathbf{u} \neq 0$. Then

$$A^k\mathbf{u} = \lambda_1^k\mathbf{x}_1\mathbf{y}_1^T\mathbf{u} + \sum_{i\neq 1} \lambda_i^k \mathbf{x}_i\mathbf{y}_i^T\mathbf{u}. \tag{3.32}$$

When k is large the first term will be much larger than the other terms. Therefore $A^k\mathbf{u}$ will be approximately λ_1^k times a multiple of the eigenvector $\mathbf{x}_1$.]

In practice we take $\mathbf{u}$ to be some convenient vector, such as the first coordinate basis vector, and we just hope that the condition is satisfied. We compute $A^k\mathbf{u}$ by successive multiplication of the matrix A times the previous vector. In order to extract the eigenvalue we can compute the result for $k+1$ and for k and divide the vectors component by component. Each quotient should be close to λ_1.

Problems

1. Take the matrix whose rows are 0, -3 and -1, 2. Apply the matrix four times to the starting vector. How close is this to an eigenvector.

2. Consider the power method for finding eigenvalues of a real matrix. Describe what happens when the matrix is symmetric and the eigenvalue of largest absolute value has multiplicity two.

3. Also describe what happens when the matrix is not symmetric and the eigenvalues of largest absolute value are a complex conjugate pair.

3.7 Inverse power method

The inverse power method is just the power method applied to the matrix $(A - \mu I)^{-1}$. We choose μ as an intelligent guess for a number that is near but not equal to an eigenvalue λ_j. The matrix has eigenvalues $(\lambda_i - \mu)^{-1}$. If μ is closer to λ_j than to any other λ_i, then the dominant eigenvalue of $(A - \mu I)^{-1}$ will be $(\lambda_j - \mu)^{-1}$. Thus we can calculate $(\lambda_j - \mu)^{-1}$ by the power method. From this we can calculate λ_j.

The inverse power method can be used to search for all the eigenvalues of A. At first it might appear that it is computationally expensive, but in fact all that one has to do is to compute an LU or QR decomposition of $A - \mu I$. Then it is easy to do a calculation in which we start with an arbitrary vector $\mathbf{u}$ and at each stage replace the vector $\mathbf{v}$ obtained at that stage with the result of solving $(A - \mu I)\mathbf{x} = \mathbf{v}$ for $\mathbf{x}$ using this decomposition.

Projects

1. Write a program to find the dominant eigenvalue of a matrix by the inverse power method.

2. Find the eigenvalues of the symmetric matrix with rows 16, 4, 1, 1 and 4, 9, 1, 1 and 1, 1, 4, 1 and 1, 1, 1, 1.

3. Change the first 1 in the last row to a 2, and find the eigenvalues of the resulting non-symmetric matrix.

3.8 Power method for subspaces

The power method for subspaces is very simple. One computes A^k for large k. Then one performs a decomposition $A^k = QR$. Finally one computes $Q^{-1}AQ$. Miracle: the result is upper triangular with the eigenvalues on the diagonal!

Here is why this works. Take $\mathbf{e}_1, \ldots, \mathbf{e}_r$ to be the first r unit basis vectors. Then $\mathbf{e}_i = \sum_j c_{ij}\mathbf{x}_j$, where the $\mathbf{x}_j$ are the eigenvectors of A corresponding to the eigenvalues ordered in decreasing absolute value. Thus for the powers we have

$$A^k \mathbf{e}_i = \sum_j c_{ij} \lambda_j^k \mathbf{x}_j. \tag{3.33}$$

To a good approximation, the first r terms of this sum are much larger than the remaining terms. Thus to a good approximation the $A^k\mathbf{e}_i$ for $1 \le i \le r$ are just linear combinations of the first r eigenvectors.

We may replace the $A^k\mathbf{e}_i$ by linear combinations that are orthonormal. This is what is accomplished by the QR decomposition. The first r columns

of Q are an orthonormal basis consisting of linear combinations of the $A^k\mathbf{e}_i$ for $1 \leq i \leq r$.

It follows that the first r columns of Q are approximately linear combinations of the first r eigenvectors. If this were exact, then $Q^{-1}AQ$ would be the exact Schur decomposition. However in any case it should be a good approximation.

[This can be considered in terms of subspaces as an attempt to apply the power method to find the subspace spanned by the first r eigenvectors, for each r. The idea is the following. Let V_r be a subspace of dimension r chosen in some convenient way. Then, in the typical situation, the first r eigenvectors will have components in V_r. It follows that for large k the matrix A^k applied to V_r should be approximately the subspace spanned by the first r eigenvectors.

However we may compute the subspace given by A^k applied to V_r by using the QR decomposition. Let

$$A^k = \tilde{Q}_k \tilde{R}_k \tag{3.34}$$

be the QR decomposition of A^k. Let V_r be the subspace of column vectors which are non-zero only in their first r components. Then $\tilde{R}_k$ leaves V_r invariant. Thus the image of this V_r by $\tilde{Q}_k$ is the desired subspace.

We expect from this that $\tilde{Q}_k$ is fairly close to mapping the space V_r into the span of the first r eigenvectors. In other words, if we define U_{k+1} by

$$U_{k+1} = \tilde{Q}_k^{-1} A \tilde{Q}_k, \tag{3.35}$$

then this is an approximation to a Schur decomposition. Thus one should be able to read off all the eigenvalues from the diagonal.]

This method is certainly simple. One simply calculates a large power of A and finds the QR decomposition of the result. The resulting orthogonal matrix give the Schur decomposition of the original A, and hence the eigenvalues.

What is wrong with this? The obvious problem is that A^k is an ill-conditioned matrix for large k, and so computing the QR decomposition is numerically unstable. Still, the idea is appealing in its simplicity.

Problems

1. Take the matrix whose rows are 0, -3 and -1, 2. Take the eigenvector corresponding to the largest eigenvalue. Find an orthogonal vector and form an orthogonal basis with these two vectors. Use the matrix with this basis to perform a similarity transformation of the original matrix. How close is the result to an upper triangular matrix?

2. Take the matrix whose rows are 0, -3 and -1, 2. Apply the matrix four times to the starting vector. Find an orthogonal vector and form an orthogonal basis with these two vectors. Use the matrix with this basis to perform a similarity transformation of the original matrix. How close is the result to an upper triangular matrix?

3.9 QR method

The famous QR method is just another variant on the power method for subspaces of the last section. However it eliminates the calculational difficulties.

Here is the algorithm. We want to find approximate the Schur decomposition of the matrix A.

Start with $U_1 = A$. Then iterate as follows. Having defined U_k, write

$$U_k = Q_k R_k, \tag{3.36}$$

where Q_k is orthogonal and R_k is upper triangular. Let

$$U_{k+1} = R_k Q_k. \tag{3.37}$$

(Note the reverse order). Then for large k the matrix U_{k+1} should be a good approximation to the upper triangular matrix in the Schur decomposition.

Why does this work?

First note that $U_{k+1} = R_k Q_k = Q_k^{-1} U_k Q_k$, so U_{k+1} is orthogonal similar to U_k.

Let $\tilde{Q}_k = Q_1 \cdots Q_k$ and $\tilde{R}_k = R_k \cdots R_1$. Then it is easy to see that

$$U_{k+1} = \tilde{Q}_k^{-1} A \tilde{Q}_k. \tag{3.38}$$

Thus U_{k+1} is similar to the original A.

Furthermore, $\tilde{Q}_k \tilde{R}_k = \tilde{Q}_{k-1} U_k \tilde{R}_{k-1} = A \tilde{Q}_{k-1} \tilde{R}_{k-1}$. Thus the kth stage decomposition is produced from the previous stage by multiplying by A.

Finally, we deduce from this that

$$\tilde{Q}_k \tilde{R}_k = A^k. \tag{3.39}$$

In other words, the $\tilde{Q}_k$ that sets up the similarity of U_{k+1} with A is the same $\tilde{Q}_k$ that arises from the QR decompositon of the power A^k. But we have seen that this should give an approximation to the Schur decomposition of A. Thus the U_{k+1} should be approximately upper triangular.

Projects

1. Implement the QR method for finding eigenvalues.

2. Use the program to find the eigenvalues of the symmetric matrix with rows 1, 1, 0, 1 and 1, 4, 1, 1 and 0, 1, 9, 5 and 1, 1, 5, 16.

3. Change the last 1 in the first row to a 3, and find the eigenvalues of the resulting non-symmetric matrix.

3.10 Finding eigenvalues

The most convenient method of finding all the eigenvalues is the QR method. Once the eigenvalues are found, then the inverse power method gives an easy determination of eigenvectors.

There are some refinements of the QR method that give greater efficiency, especially for very large matrices.

The trick is to work with *Hessenberg matrices*, which are matrices with zeros below the diagonal below the main diagonal.

The idea is to do the eigenvalue determination in two stages. The first stage is to transform A to $Q^{-1}AQ = \hat{A}$, where $\hat{A}$ is a Hessenberg matrix. This is an orthogonal similarity transformation, so this gives a matrix $\hat{A}$ with the same eigenvalues.

This turns out to be an easy task. The idea is much the same as the idea for the QR decomposition, except that the reflections must be applied on both sides, to make it an orthogonal similarity transformation. No limiting process is involved.

One builds the matrix Q out of reflection matrices, $Q = P_n \cdots P_1$. At the jth stage the matrix is $P_j \cdots P_1 A P_1 \cdots P_j$. The unit vector determining the reflection P_j is taken to be zero in the first j components. Furthermore it is chosen so that application of P_j on the left will zero out the components in the jth column below the entry below the diagonal entry. The entry just below the diagonal entry does not become zero. However the advantage is that the application of P_j on the right does not change the jth column or any of the preceding columns.

Now that the matrix is in Hessenberg form, we note that the QR algorithm preserves Hessenberg form. We take the first $U_1 = \hat{A}$, which is in Hessenberg form. Then we may easily compute that

$$U_{k+1} = R_k U_k R_k^{-1}. \tag{3.40}$$

Thus each U_k is in Hessenberg form, since Hessenberg form is preserved by multiplication by upper triangular matrices.

This is very advantageous, since at each stage we must decompose $U_k = Q_k R_k$ and then multiply out $R_k Q_k$. Since U_k is in Hessenberg form, the

reflection vectors used in the decomposition are each vectors that have only two non-zero components. The arithmetic is much reduced.

æ

Chapter 4

Nonlinear systems

4.1 Introduction

This chapter deals with solving equations of the form $\mathbf{f}(\mathbf{x}) = 0$, where $\mathbf{f}$ is a continuous function from $\mathbf{R}^n$ to $\mathbf{R}^n$. It also treats questions of roundoff error and its amplification in the course of a numerical calculation.

In much of what we do the derivative $\mathbf{f}'$ of such a function $\mathbf{f}$ will play an essential role. This is defined in such as way that

$$\mathbf{f}(\mathbf{x}+\mathbf{h}) - \mathbf{f}(\mathbf{x}) = \mathbf{f}'(\mathbf{x})\mathbf{h} + \mathbf{r}, \tag{4.1}$$

where the remainder is of higher than first order in the vector $\mathbf{h}$. Thus the derivative $\mathbf{f}'(\mathbf{x})$ is a matrix. If we write this in variables with $\mathbf{y} = \mathbf{f}(\mathbf{x})$, then the derivative formula is

$$\Delta\mathbf{y} \approx \mathbf{f}'(\mathbf{x})\Delta\mathbf{x}. \tag{4.2}$$

If we write these relations in components, we get

$$f_i(\mathbf{x}+\mathbf{h}) = f_i(\mathbf{x}) + \sum_{j=1}^{n} \frac{\partial f_i(\mathbf{x})}{\partial x_j} h_j + r_i. \tag{4.3}$$

Thus the derivative matrix is the matrix of partial derivatives. Using variables one writes

$$\Delta y_i \approx \sum_{j=1}^{n} \frac{\partial y_i}{\partial x_j} \Delta x_j. \tag{4.4}$$

The same idea is often expressed in differential notation

$$dy_i = \sum_{j=1}^{n} \frac{\partial y_i}{\partial x_j} dx_j. \tag{4.5}$$

There are several interpretations of such a function. Two of the most important are the interpretation as a *transformation* and the interpretation as a *vector field*.

When we think of a function $\mathbf{g}$ as a transformation, we may think of $\mathbf{x}$ as being a point in one space and $\mathbf{y} = \mathbf{g}(\mathbf{x})$ as being a point in some other space. For each $\mathbf{x}$ there is a corresponding $\mathbf{y}$. It is illuminating to look at the consequence of a change of coordinate system. Say that we have $\mathbf{z}$ coordinates that are functions of the $\mathbf{x}$ coordinates, and we have $\mathbf{w}$ coordinates that are functions of the $\mathbf{y}$ coordinates. In that case we have

$$\frac{\partial w_i}{\partial z_j} = \sum_k \frac{\partial w_i}{\partial y_k} \sum_r \frac{\partial y_k}{\partial x_r} \frac{\partial x_r}{\partial z_j}. \tag{4.6}$$

This says that the new derivative matrix is obtained from the original matrix by multiplying on each side by matrices representing the effect of the coordinate transformations.

A variant is when we think of the function as a transformation from a space to the same space. In that case we may write $\hat{\mathbf{x}} = \mathbf{g}(\mathbf{x})$ and think of $\mathbf{x}$ as the coordinates of the original point and $\hat{\mathbf{x}}$ as the coordinates of the new point.

In this case there is only the change from $\mathbf{x}$ to $\mathbf{z}$ coordinates, so the change of variable formula becomes

$$\frac{\partial \hat{z}_i}{\partial z_j} = \sum_k \frac{\partial \hat{z}_i}{\partial \hat{x}_k} \sum_r \frac{\partial \hat{x}_k}{\partial x_r} \frac{\partial x_r}{\partial z_j}. \tag{4.7}$$

We shall see in the problems that there is a special situation when this change is a familiar operation of linear algebra.

When we think of a function $\mathbf{f}$ as a vector field, then we think of $\mathbf{x}$ as being a point in some space and $\mathbf{y} = \mathbf{f}(\mathbf{x})$ as being the components of a vector attached to the point $\mathbf{x}$.

Let us look at the effect of a change of coordinates on the vector field itself. We change from $\mathbf{x}$ to $\mathbf{z}$ coordinates. Let us call the new components of the vector field $\bar{\mathbf{y}}$. Then if we look at a curve tangent to the vector field, we see that along the curve

$$\bar{y}_i = \frac{d\bar{z}_i}{dt} = \sum_k \frac{\partial z_i}{\partial x_k} \frac{dx_k}{dt} = \sum_k \frac{\partial z_i}{\partial x_k} y_k. \tag{4.8}$$

So the vector field is changed by multiplication on the left by a matrix:

$$\bar{y}_i = \sum_k \frac{\partial z_i}{\partial x_k} y_k. \tag{4.9}$$

How about the partial derivatives of the vector field? Here the situation is ugly. A routine computation gives

$$\frac{\partial \bar{y}_i}{\partial z_j} = \sum_r \sum_k [\frac{\partial^2 z_i}{\partial x_r \partial x_k} y_k + \frac{\partial z_i}{\partial x_k}\frac{\partial y_k}{\partial x_r}]\frac{\partial x_r}{\partial z_j}. \tag{4.10}$$

This does not even look like matrix multiplication. We shall see in the problems that there is a special situation where this difficulty does not occur and where we get instead some nice linear algebra.

Problems

1. Consider a transformation $\hat{\mathbf{x}} = \mathbf{g}(\mathbf{x})$ of a space to itself. Show that at a fixed point the effect of a change of coordinates on the derivative matrix is a similarity transformation.

2. Consider a vector field with components $\mathbf{y} = \mathbf{f}(\mathbf{x})$ in the $\mathbf{x}$ coordinate system. Show that at a zero of the vector field the effect of a change of coordinates on the derivative matrix is a similarity transformation.

4.2 Degree

The intermediate value theorem was a fundamental result in solving equations in one dimension. It is natural to ask whether there is an analog of this theorem for systems. There is such an analog; one version of it is the following *topological degree* theorem.

We say that a vector $\mathbf{y}$ is opposite to another vector $\mathbf{x}$ if there exists $c \geq 0$ with $\mathbf{y} = -c\mathbf{x}$.

Theorem 4.2.1 *Let $\mathbf{f}$ be a continuous function on the closed unit ball B in $\mathbf{R}^n$ with values in $\mathbf{R}^n$. Let ∂B be the sphere that is the boundary of B. Assume that for each $\mathbf{x}$ in ∂B the vector $\mathbf{f}(\mathbf{x})$ is not opposite to $\mathbf{x}$. Then there exists a point $\mathbf{r}$ in B such that $\mathbf{f}(\mathbf{r}) = 0$.*

This is called a topolological degree theorem because there is an invariant called the degree which under the hypotheses of this theorem has the value one. In general, if the degree is non-zero, then there is a fixed point.

The proof of this theorem is much more difficult than the proof of the intermediate value theorem, and it will not be attempted here.

When $n = 1$ we are essentially in the situation of the intermediate value theorem. The unit ball B is the closed interval $[-1, 1]$, and the boundary ∂B consists of the two points -1 and 1. The computational problem thus consists of checking the value of a function on these two end points.

Unfortunately, the theorem does not lead to a particularly efficient computer implementation when $n > 1$. (However perhaps something can be done when $n = 2$.) The problem is that the boundary ∂B is an infinite set, and one has to do a calculation involving the whole boundary to check the presence of a root.

Problems

1. Show how to derive the intermediate value theorem as a corollary of the degree theorem.

2. Find a example in two dimensions where the degree theorem applies to guarantee the existence of the root, but where the root cannot be calculated by elementary means.

4.2.1 Brouwer fixed point theorem

The degree theorem has implications for the existence of fixed points. The most famous such result is the Brouwer fixed point theorem.

Theorem 4.2.2 *Let* $\mathbf{g}$ *be a continuous function on the closed unit ball* B *in* $\mathbf{R}^n$ *with values in* $\mathbf{R}^n$*. Let* ∂B *be the sphere that is the boundary of* B*. Assume* $\mathbf{g}$ *sends* ∂B *into* B*. Then* $\mathbf{g}$ *has a fixed point.*

Proof: Let $\mathbf{f}(\mathbf{x}) = \mathbf{x} - \mathbf{g}(\mathbf{x})$ defined on the closed unit ball B. We want to show that for some $\mathbf{x}$ in B the vector $\mathbf{f}(\mathbf{x}) = 0$.

Suppose that for all $\mathbf{x}$ in B the vector $\mathbf{f}(\mathbf{x}) \neq 0$. Then $\mathbf{g}$ maps ∂B into the interior of the unit ball. Consider $\mathbf{x}$ in ∂B. If $\mathbf{f}(\mathbf{x})$ is opposite to $\mathbf{x}$, then $\mathbf{g}(\mathbf{x}) = (1 + c)\mathbf{x}$ with $c \geq 0$, which is impossible. Therefore $\mathbf{f}(\mathbf{x})$ is never opposite to $\mathbf{x}$. But then the degree theorem leads to a contradiction. □

Unfortunately, this proof only reduces the Brouwer theorem to the degree theorem, and does not provide a self-contained proof. Again the apparatus of algebraic topology is necessary for a satisfactory treatment.

Two subsets $\mathbf{R}^n$ are *homeomorphic* if there is a one-to-one correspondence between them that is continuous in both directions. The Brouwer fixed point theorem applies to a subset that is homeomorphic to the closed unit ball. Thus it applies to a closed ball of any size, or to a closed cube.

4.3 Iteration

4.3.1 First order convergence

Another approach to numerical root-finding is *iteration*. Assume that $\mathbf{g}$ is a continuous function. We seek a fixed point $\mathbf{r}$ with $\mathbf{g}(\mathbf{r}) = \mathbf{r}$. We can

attempt to find it by starting with an $\mathbf{x}_0$ and forming a sequence of iterates using $\mathbf{x}_{n+1} = \mathbf{g}(\mathbf{x}_n)$.

Theorem 4.3.1 *Let* $\mathbf{g}$ *be continuous and let* $\mathbf{x}_n$ *a sequence such that* $\mathbf{x}_{n+1} = \mathbf{g}(\mathbf{x}_n)$. *Then if* $\mathbf{x}_n \to \mathbf{r}$ *as* $n \to \infty$, *then* $\mathbf{g}(r) = \mathbf{r}$.

This theorem shows that we need a way of getting sequences to converge. In higher dimensions the most convenient approach is to have a bound on the derivative.

How do we use such a bound? We need a replacement for the mean value theorem. Here is the version that works.

Lemma 4.3.1 *Let* $\mathbf{g}$ *be a function with continuous derivative* $\mathbf{g}'$. *Then*

$$|\mathbf{g}(\mathbf{y}) - \mathbf{g}(\mathbf{x})| \leq \max_{0 \leq t \leq 1} \|\mathbf{g}'(\mathbf{z}_t)\| \, |\mathbf{y} - \mathbf{x}|, \tag{4.11}$$

where $\mathbf{z}_t = (1-t)\mathbf{x} + t\mathbf{y}$ *lies on the segment between* $\mathbf{x}$ *and* $\mathbf{y}$.

The way to prove the lemma is to use the identity

$$\mathbf{g}(\mathbf{y}) - \mathbf{g}(\mathbf{x}) = \int_0^1 \mathbf{g}'(\mathbf{z}_t)(\mathbf{y} - \mathbf{x}) \, dt. \tag{4.12}$$

Theorem 4.3.2 *Let* B *be a set that is homeomorphic to a closed ball. Assume that* $\mathbf{g}$ *is continuous on* B *and that* $\mathbf{g}$ *maps* B *into itself. Then* $\mathbf{g}$ *has a fixed point. Assume also that* B *is convex and that* $\|\mathbf{g}'(\mathbf{z})\| \leq K < 1$ *for all* $\mathbf{z}$ *in* B. *Let* $\mathbf{x}_0$ *be in* B *and iterate using* $\mathbf{x}_{n+1} = \mathbf{g}(\mathbf{x}_n)$. *Then the iterates converge to the fixed point. Furthermore, this fixed point is the unique fixed point in* B.

Proof: The Brouwer theorem guarantees that there is a fixed point $\mathbf{r}$ in the interval. By the mean value theorem, for each $\mathbf{x}$ we have $\mathbf{x}_{n+1} - \mathbf{r} = \mathbf{g}(\mathbf{x}_n) - \mathbf{g}(\mathbf{r})$. By the lemma the norm of this is bounded by K times the norm of $\mathbf{x} - \mathbf{r}$. In other words each iteration replacing $\mathbf{x}$ by $\mathbf{g}(\mathbf{x})$ brings us closer to $\mathbf{r}$. □

We see that that $\mathbf{r}$ is a stable fixed point if $\|\mathbf{g}'(\mathbf{r})\| < 1$. However there is a stronger result. Recall that the power of a matrix has a norm that gives a much better bound on the eigenvalues. We may compute the derivative of the mth iterate $\mathbf{g}^m(\mathbf{x})$ and we get $\mathbf{g}'(\mathbf{g}^{m-1}(\mathbf{x}))\mathbf{g}'(\mathbf{g}^{m-2}(\mathbf{x})) \cdots \mathbf{g}'(\mathbf{g}(\mathbf{x}))\mathbf{g}'(\mathbf{x})$. At the fixed point with $\mathbf{g}(\mathbf{r}) = \mathbf{r}$ this is just the power $\mathbf{g}'(\mathbf{r})^m$. So near the fixed point we expect that

$$\mathbf{x}_{n+m} - \mathbf{r} = \mathbf{g}^m(\mathbf{x}_n) - \mathbf{g}^m(\mathbf{r}) \approx \mathbf{g}'(\mathbf{r})^m(\mathbf{x}_n - \mathbf{r}). \tag{4.13}$$

We see that if $\|\mathbf{g}(\mathbf{r})^m\| < 1$ then the fixed point $\mathbf{r}$ is *stable*.

If we want to use this to solve $\mathbf{f}(\mathbf{x}) = 0$, we can try to take $\mathbf{g}(\mathbf{x}) = \mathbf{x} - C\mathbf{f}(\mathbf{x})$ for some suitable matrix K. If C is chosen so that $\mathbf{g}'(\mathbf{x}) = 1 - C\mathbf{f}'(\mathbf{x})$ is small for $\mathbf{x}$ near $\mathbf{r}$, then there should be a good chance of convergence.

4.3.2 Second order convergence

Since the speed of convergence in iteration with $\mathbf{g}$ is controlled by $\mathbf{g}'(\mathbf{r})$, it follows that the situation when $\mathbf{g}'(\mathbf{r}) = 0$ is going to have special properties.

It is possible to arrange that this happens. Say that one wants to solve $\mathbf{f}(\mathbf{x}) = 0$. *Newton's method* is to take $\mathbf{g}(\mathbf{x}) = \mathbf{x} - \mathbf{f}'(\mathbf{x})^{-1}\mathbf{f}(\mathbf{x})$. It is easy to check that $\mathbf{f}(\mathbf{r}) = 0$ and $\mathbf{f}'(\mathbf{r})$ non-singular imply that $\mathbf{g}'(\mathbf{r}) = 0$.

Newton's method is not guaranteed to be good if one begins far from the starting point. The damped Newton method is more conservative. One defines $\mathbf{g}(\mathbf{x})$ as follows. Let $\mathbf{m} = \mathbf{f}'(\mathbf{x})^{-1}\mathbf{f}(\mathbf{x})$ and let $\mathbf{y} = \mathbf{x} - \mathbf{m}$. While $|\mathbf{f}(\mathbf{y})| \geq |\mathbf{f}(\mathbf{x})|$ replace $\mathbf{m}$ by $\mathbf{m}/2$ and let $\mathbf{y} = \mathbf{x} - \mathbf{m}$. Let $\mathbf{g}(\mathbf{x})$ be the final value of $\mathbf{y}$.

Projects

1. Newton's method for systems has the disadvantage that one must compute many partial derivatives. Steffensen's method provides an alternative. The method is to iterate with $\mathbf{g}(\mathbf{x}) = \mathbf{x} - \mathbf{w}$, where $\mathbf{w}$ is the solution of $J(\mathbf{x})\mathbf{w} = \mathbf{f}(\mathbf{x})$. For Newton's method $J(\mathbf{x}) = \mathbf{f}'(\mathbf{x})$, but for Steffensen's method we approximate the matrix of partial derivatives by a matrix of difference quotients. Thus the i, j entry of $J(\mathbf{x})$ is $(f_i(\mathbf{x} + h_j\mathbf{e}_j) - f_i(\mathbf{x}))/h_j)$, where $h_j = \alpha_j(\mathbf{f}(\mathbf{x}))$. Here α is a function that vanishes at zero. Thus as $\mathbf{f}(\mathbf{x})$ approaches zero, these difference quotients automatically approach the partial derivatives.

   There are various possible choices for the function α. One popular choice is the identity $\alpha_j(\mathbf{z}) = z_j$, so that $h_j = f_j(\mathbf{x})$. The disadvantage of this choice is that h_j can be zero away from the root.

   Another method is to take each component of α to be the length, so that $\alpha_j(\mathbf{z}) = |\mathbf{z}|$ and $h_j = |\mathbf{f}(\mathbf{x})|$. This choice of α is not differentiable at the origin, but in this case this is not a problem.

   Perhaps an even better method is to take $\alpha_j(\mathbf{z})$ to be the minimum of $|\mathbf{z}|$ and some small number, such as 0.01. This will make the difference matrix somewhat resemble the derivative matrix even far from the solution.

   The project is to write a program for solving a system of non-linear equations by Steffensen's method. Try out the program on a simple system for which you know the solution.

2. Use the program to solve the following system.

$$\begin{aligned} x^3 - 3xy^2 - 6z^3 + 18zw^2 - 1 &= 0 \\ 3x^2y - y^3 - 18z^2w + 6w^3 &= 0 \end{aligned}$$

$$\begin{aligned} xz - yw - 1 &= 0 \\ yz + xw &= 0 \end{aligned}$$

Find a solution near the point where (x, y, z, w) is $(0.6, 1.1, 0.4, -0.7)$.

Problems

1. Let B be the ball of radius r centered at $\mathbf{c}$. Assume that $\|\mathbf{g}'(\mathbf{z})\| \leq K < 1$ for all $\mathbf{z}$ in B. Suppose that $\mathbf{a}$ is in B and that $\mathbf{a}$ and $\mathbf{g}(\mathbf{a})$ satisfy $Kr + K|\mathbf{a} - \mathbf{c}| + |\mathbf{g}(\mathbf{a}) - \mathbf{c}| \leq r$. Show that $\mathbf{g}$ maps B into B.

2. Let $\mathbf{g}(\mathbf{x}) = \mathbf{x} - C\mathbf{f}(\mathbf{x})$. Show that if C approximates $\mathbf{f}'(\mathbf{x})^{-1}$ in the sense that $\|C - \mathbf{f}'(\mathbf{x})^{-1}\|\|\mathbf{f}'(\mathbf{x})\|$ is bounded below one near the fixed point, then iteration with $\mathbf{g}$ starting near the fixed point converges to the fixed point.

3. Calculate $\mathbf{g}'(\mathbf{x})$ in Newton's method.

4. Show that in Newton's method $\mathbf{f}(\mathbf{r}) = 0$ with $\mathbf{f}'(\mathbf{r})$ invertible implies $\mathbf{g}'(\mathbf{r}) = 0$.

5. The following problems deal with the theory of Steffensen's method. For simplicity we deal with the scalar case. Thus the method consists of iteration with $g(x) = x - f(x)/m(x)$, where $m(x) = (f(x + \alpha(f(x))) - f(x))/\alpha(f(x))$. The function α is chosen so that $\alpha(0) = 0$. Let r be a root of f, so that $f(r) = 0$, and assume that $f'(r) \neq 0$. The first problem is to compute $m(r)$ and $m'(r)$.

6. Show that $g'(r) = 0$ and that $g''(r) = (2m'(r) - f''(r))/f'(r)$.

7. Let $x_{n+1} = g(x_n)$ and assume $x_n \to r$ as n to ∞. Evaluate the limit of $(x_{n+1} - r)/(x_n - r)^2$ as $n \to \infty$.

8. What if $\alpha(z) = |z|$, so that α is not differentiable at 0. Is it still true that $g'(r) = 0$.

9. Another interesting problem is to examine the situation when the iterations give increasing or decreasing sequences of vectors. Show that if the matrix $\mathbf{f}'(\mathbf{x})^{-1}$ has positive entries and $\mathbf{f}(\mathbf{x})$ has positive entries, then $\mathbf{g}(\mathbf{x}) \leq \mathbf{x}$.

10. This leads to the problem of finding when a matrix A has the property that A^{-1} has positive entries. Show that if $A = D - N$, where D is diagonal and N is off-diagonal and $D \geq 0$ and $N \geq 0$ and $\|D^{-1}N\|_\infty < 1$, then A^{-1} has only positive entries.

11. Show that if in this situation A^{-1} exists but we have only $\|D^{-1}N\|_\infty \leq 1$, then the same conclusion holds.

12. Assume that $\mathbf{x} \geq \mathbf{r}$ implies $\mathbf{f}(\mathbf{x}) \geq 0$. We want to see when $\mathbf{x} \geq \mathbf{r}$ implies $\mathbf{g}(\mathbf{x}) \geq \mathbf{r}$. Evaluate $\mathbf{g}'(\mathbf{z})$ terms of $\mathbf{f}''(\mathbf{z})$ and show that it is sufficient that all the second partial deriviatives in this expression are positive.

4.4 Power series

Let C be a matrix with $\|C\| < 1$. Then $(1-C)^{-1}$ exists. We may most easily see this by expanding C in a power series. By summing this series we see that $\|(1-C)^{-1}\| \leq 1/(1-\|C\|)$.

Recall that in general $\|C^k\| \leq \|C\|^k$. Thus there is an interesting generalization to the case when $\|C^k\| < 1$ for some k. We observe that

$$(1-C)^{-1} = (1 + C + C^2 + \cdots + C^{k-1})(1-C^k)^{-1}. \tag{4.14}$$

It follows that also in this situation $(1-C)^{-1}$ exists.

Problems

1. Find a bound for $(1+C)^{-1}$ in terms of $\|C\|$ and $\|C^k\|$.

2. Show that if A^{-1} exists and $\|(EA^{-1})^k\| < 1$ for some k, then $(A-E)^{-1}$ exists.

4.5 The spectral radius

The *spectral radius* $\rho(A)$ of a square matrix A is the maximum absolute value of an eigenvalue. Even if the matrix is real, it is important in this definition that all complex eigenvalues are considered. The reason for this importance is that with this definition we have the following theorem.

Theorem 4.5.1 *The norms of powers A^n and the spectral radius of A are related by*

$$\lim_{n\to\infty} \|A^n\|^{\frac{1}{n}} = \rho(A). \tag{4.15}$$

Proof:

It is easy to see that for all n it is true that $\rho(A) \leq \|A^n\|^{\frac{1}{n}}$. The problem is to show that for large n the right hand side is not much larger than the left hand side.

The first thing to do is to check that $|z| < 1/\rho(A)$ implies that $(I-zA)^{-1}$ exists. (Otherwise $1/z$ would be an eigenvalue of A outside of the spectral radius.)

The essential observation is that $(I - zA)^{-1}$ is an analytic function of z for $|z| < 1/\rho(A)$. It follows that the power series expansion converges in this disk. Thus for $|z|$ with $|z| < 1/\rho(A)$ there is a constant c with $\|(zA)^n\| = |z|^n\|A^n\| \leq c$.

We have shown that for every $r < 1/\rho(A)$ there is a c with $\|A^n\|^{\frac{1}{n}} \leq c^{\frac{1}{n}}/r$. Take $1/r$ to be larger than but very close to $\rho(A)$. Take n so large that $c^{\frac{1}{n}}/r$ is still close to $\rho(A)$. Then $\|A^n\|$ must be larger than but close to $\rho(A)$. □

Let us look at this proof in more detail. The essential point is the convergence of the power series. Why must this happen? It is a miracle of complex variable: the Cauchy integral formula reduces the convergence of an arbitrary power series inside its radius of convergence to the convergence of a geometric series.

Look at the Cauchy integral formula

$$(I - wA)^{-1} = \frac{1}{2\pi i}\int (1 - zA)^{-1} 1/(z - w)\, dz, \tag{4.16}$$

where w is inside the circle of integration $|z| = r$ and $r < 1/\rho(A)$. We may expand in a geometric series in powers of w/z. From this we see that the coefficients of the expansion in powers of w are

$$A^n = \frac{1}{2\pi i}\int (1 - zA)^{-1} 1/z^{n+1}\, dz. \tag{4.17}$$

This proves that $\|A^n\| \leq c/r^n$, where

$$c = \frac{1}{2\pi r}\int \|(I - zA)^{-1}\|\, d|z| \tag{4.18}$$

over $|z| = r$.

Notice that as r approaches $1/\rho(A)$ the bound c will become larger, due to the contribution to the integral from the singularity at $z = 1/\lambda$.

Problems

1. Show that it is false in general that $\rho(A + B) \leq \rho(A) + \rho(B)$. Hint: Find 2 by 2 matrices for which the right hand side is zero.

4.6 Linear algebra review

We briefly review the various similarity representations of matrices. We are interested in the case of a real square matrix A.

There is always a (possibly complex) matrix S and a (possibly complex) upper triangular matrix J such that $S^{-1}AS = J$. Thus A has the Jordan representation $A = SJS^{-1}$. The eigenvalues of A occur on the diagonal of J.

Assume that A has distinct eigenvalues. Then the eigenvectors of A form a basis. Then there is always a (possibly complex) matrix S and a (possibly complex) diagonal matrix D such that $S^{-1}AS = D$. Thus A has the spectral representation $A = SDS^{-1}$. The eigenvalues of A occur on the diagonal of D. The eigenvectors form the columns of the matrix S.

Assume that A has real eigenvalues (not necessarily distinct). Then there is always an orthogonal matrix Q and an upper triangular matrix U such that $Q^{-1}AQ = U$. Thus A has the Schur representation $A = QUQ^{-1}$ where $Q^{-1} = Q^T$. The eigenvalues of A occur on the diagonal of U.

Assume that A is symmetric, so $A = A^T$. Then A has real eigenvalues (not necessarily distinct). The eigenvectors of A may be chosen so as to form an orthonormal basis. There is always an orthogonal matrix Q and a diagonal matrix D such that $Q^{-1}AQ = D$. Thus A has the spectral representation $A = QDQ^{-1}$ where $Q^{-1} = Q^T$. The eigenvalues of A occur on the diagonal of D. The eigenvectors may be chosen to form the columns of the matrix Q.

4.7 Error analysis

There are several sources of error in numerical analysis. There are some that are obvious and inescapable, such as the *input error* in data from the outside world, and the inherent *representation error* in the precision that is available in the output.

4.7.1 Approximation error and roundoff error

However there are two main sources of error. One is the *approximation error*. This is the error that is due to approximating limits by finite arithmetical operators. It is sometimes called truncation error and sometimes called discretization error, depending on the context.

The most common tool for treating approximation error is Taylor's formula with remainder. The idea is that a function such as $f(x) = f(a) + f'(a)(x-a) + 1/2f''(a)(x-a)^2 + r$ is approximated by a polynomical $p(x) = f(a) + f'(a)(x-a) + 1/2f''(a)(x-a)^2$. The advantage is that the polynomial can be computed by addition and multiplication, which are operations supported by the computer. The disadvantage is that there is an error r. One hopes that the error r is small, and that one can prove that it is small.

There are other forms of approximation error, but in every case the challenge is to find algorithms for which the approximation error is small. This is a general mathematical problem, perhaps the central problem in analysis. In this respect numerical analysis appears as nothing more than a subset of analysis.

Of course some parts of numerical analysis have no approximation error. For instance, the formulas for inverting a matrix using the LU or QR decomposition are exact.

Examples that we have encountered where there is approximation error are root-finding and eigenvalue computation. In each of these cases the computation is only exact when one performs infinitely many iterations, which is impossible on the computer.

The other main source of error is *roundoff error*. This is due to the fact that computer does not do exact arithmetic with real numbers, but only floating point arithmetic. This source of error is peculiar to numerical analysis.

4.7.2 Amplification of absolute error

We want an algorithm to have the property that it does not magnify roundoff error needlessly. For a start we do an analysis of *absolute error*.

For the purposes of analysis, imagine that we want to compute a function $\mathbf{y} = \mathbf{f}(\mathbf{x})$. Then $d\mathbf{y} = \mathbf{f}'(\mathbf{x})d\mathbf{x}$, so small input error is amplified by the entries in the matrix $\mathbf{f}'(\mathbf{x})$. If this matrix has large entries, then the problem in inherently ill-conditioned with respect to absolute error. Nothing can be done to remedy this.

Example: Consider the problem of solving a polynomial equation $p(z) = 0$. The roots are functions of the coefficients. Let a be one of the coefficients, say the coefficient of z^m. Then the root y satisfies an equation $p(y) = P(a, y) = 0$. Differentiate this equation with respect to a. We obtain $y^m + p'(y)dy/da = 0$, where $p'(z)$ is the derivative of the polynomial with respect to z. This shows that the problem of finding the root y as a function of the coefficient a is not well-posed when $p'(y) = 0$, that is, when y is a multiple root.

However, even if the problem is well-conditioned, a poor choice of algorithm can give trouble. Say that our numerical algorithm is to compute $\mathbf{z} = \mathbf{h}(\mathbf{x})$ and then $\mathbf{y} = \mathbf{g}(\mathbf{z})$. Then we have an intermediate stage at which we can introduce roundoff errors. We have $d\mathbf{y} = \mathbf{g}'(\mathbf{z})d\mathbf{z}$, so if $\mathbf{g}'(\mathbf{z})$ is large, which can happen, then we can have these intermediate errors amplified.

We can examine this amplification effect rather crudely in terms of the norm of the matrix $\mathbf{g}'(\mathbf{z})$. Or we can write out the entries explicitly as

$$dy_i = \sum_k \frac{\partial y_i}{\partial z_k}\, dz_k \tag{4.19}$$

and perform a detailed analysis.

From the chain rule, we always have $\mathbf{f}'(\mathbf{x}) = \mathbf{g}'(\mathbf{z})\mathbf{h}'(\mathbf{x})$. So, roughly speaking, for a fixed problem of computing $\mathbf{f}(\mathbf{x})$, we want to choose an algorithm so as to take $\mathbf{g}'(\mathbf{z})$ to have a reasonably small norm, not greatly exceeding the norm of $\mathbf{f}'(\mathbf{x})$. This keeps the norm of $\mathbf{h}'(\mathbf{x})$ from being small, but this norm does not matter. (The whole point is not to needlessly amplify roundoff errors from intermediate stages.)

We say that an algorithm is *numerically stable* if the errors that are introduced from intermediate stages are not much larger than the inherent roundoff errors that arize from the input error and the representation error.

A problem can be well-conditioned, but we can make the mistake of choosing a numerically unstable algorithm. This leads to wrong answers!

Example: Here is a classic example. Say that one wants to compute the function $y = f(x) = \sqrt{x+1} - \sqrt{x}$ for large x. One does this in stages. The first stage is to compute $z = \sqrt{x+1}$ and $w = \sqrt{x}$. (Note that $z^2 - w^2 = 1$.) The second stage may be performed in two ways. The obvious but undesirable way is to compute $y = z - w$. The better way is to compute $y = 1/(z + w)$.

The derivative dy/dx is very small when x is large. Errors in x are damped. However in the undesirable method $\partial y/\partial z = 1$ and $\partial y/\partial w = -1$, which are not small. So errors in z and w are not damped.

With the better method, the partial derivatives are $\partial y/\partial z = -1/(z+w)^2$ and $\partial y/\partial w = -1/(z+w)^2$. Errors in z and w are damped. Clearly this method is preferable.

Example: Fix $0 < a < 1$. Let $b = a + 1/a$. Consider the problem of solving the recurrence relation $x_{n+1} = bx_n - x_{n-1}$ with $x_1 = a$ and $x_0 = 1$. It is easy to see that the solution is $x_k = a^k$. Clearly $dx_k/da = ka^{k-1}$ is small. The problem is well-posed.

However the recurrence relation is a terrible way to compute the answer. The reason is that this is essentially computing the matrix power in

$$\begin{pmatrix} x_{k+1} \\ x_k \end{pmatrix} = \begin{pmatrix} b & -1 \\ 1 & 0 \end{pmatrix}^k \begin{pmatrix} a \\ 1 \end{pmatrix}. \tag{4.20}$$

The largest eigenvalue of the matrix is $1/a > 1$, so for large k this has very large norm. Errors (from earlier stages of taking the power) are very much amplified!

4.7.3 Amplification of relative error

The arithmetic used in most computers is floating point arithmetic, so the roundoff error is more or less independent of the size of the number. It should be thought of as a relative error. Thus the amplification effect of relative error is what is actually important.

The formula for the dependence of relative error in the output on relative error in an intermediate stage is

$$\frac{dy_i}{y_i} = \sum_k \frac{z_k}{y_i} \frac{\partial y_i}{\partial z_k} \frac{dz_k}{z_k}. \tag{4.21}$$

It is of course possible to think of this as the formula for the absolute error amplifiation for logarithms

$$d \log |y_i| = \sum_k \frac{\partial \log |y_i|}{\partial \log |z_k|} \, d \log |z_k|. \tag{4.22}$$

In this sense it falls under the scope of the analysis of the previous section.

Ultimately, however, the result is that the amplification of relative error is given by a matrix with entries $(z_k/y_i)(\partial y_i/\partial z_k)$. We want this to be a small matrix. We have no control over the y_i, since this is simply the output value. But the algorithm controls the size of the intermediate number z_k.

The conclusion is that, unless there is a compensating small derivative, we prefer small sizes of the numbers z_k that occur in intermediate steps of the computation.

Example: Say that one wants to compute $y = f(x) = x + 2$ at $x = 2$. A bad algorithm is to compute $z = 1000001x$ and $w = 1000000x$ and compute $y = z - w + 2$. The reason is that a relative error of a millionth in z or w will give a completely wrong answer. Big numbers have been introduced needlessly.

Example: Say that one wants to compute $y = f(x) = x + 2$ at $x = 2$. One algorithm is to compute $z = 1000000x$ and then $y = 1/1000000z + 2$. This introduces a big number z, but in this case it does no harm. The reason is that the small dy/dz compensates the large z.

Example: Let us repeat the example of the function $f(x) = \sqrt{x+1} - \sqrt{x}$, this time making the analysis using relative error. The first stage is to compute $z = \sqrt{x+1}$ and $w = \sqrt{x}$. (Note that $z^2 - w^2 = 1$.) The second stage may be performed in two ways. The obvious but undesirable way is to compute $y = z - w$. The better way is to compute $y = 1/(z + w)$.

The relative error amplification inherent in the problem is $(x/y)dy/dx$ which evaluates to $-x/(zw)$, a quantity bounded by one. The problem is well-conditioned.

However in the undesirable method we have $(z/y)\partial y/\partial z = z(z+w)$ and also $(w/y)\partial y/\partial w = -w(z + w)$, which are very large. So relative errors in z and w are amplified.

With the better method, the partial derivatives are given by the expressions $(z/y)\partial y/\partial z = -z/(z+w)$ and $(w/y)\partial y/\partial w = -w/(z+w)$. This does not amplify relative error, and so this method is numerically stable. Note that in this particular example the advantage of the better method is the same in the absolute error analysis and the relative error analysis. This is because the two algorithms deal with the precisely the same numbers in the intermediate stage.

Problems

1. Say that $f(a, y) = 0$ defines y as a function of a. When is this problem of determining y from a well-posed?

2. Consider the problem of finding $y = e^{-x}$ for large values of x. One strategy is to use the nth partial sum of the Taylor series expansion about zero. How large must n be chosen so that the approximation error is smaller than some small $\epsilon > 0$?

3. Methods using Taylor series can have problems with roundoff errror. Consider the problem of finding $y = e^{-x}$ for large x. Here are two

methods. Neither is particularly good, but one is considerably better than the other. The problem is to give a relative error analysis.

The first method is to compute $y = z + w$, where z is the nth partial sum of the Taylor series of y is powers of x about zero. The other term w is the remainder (which we assume can be well-approximated).

The second method is to compute $y = 1/u$. Here $u = z + w$, where z is the nth partial sum of the Taylor series of u in powers of x about zero. The other term is the remainder (which we assume again can be well-approximated).

4.8 Numerical differentiation

Let us apply our analysis of error to the example of numerical differentiation. We want to start from x and compute $y' = f'(x)$. The relative error amplification is $(x/y')(dy'/dx)$ and so it is controlled by the second derivative $f''(x)$. So the problem is well-conditioned in most circumstances.

The method of numerical differentiation is to compute $z = f(x+h)$ and $w = f(x)$ and approximate y' by $(z - w)/h$. This is supposed to be a good approximation when h is very small.

The relative amplification factor from z is given by $(z/y')\partial y'/\partial z$. In this approximation this is $(z/y')(1/h)$. This is ordinarily huge when h is very small. Therefore numerical differentiation is not a numerically stable way of computing derivatives.

There is one exceptional case when numerical differentiation is not so bad. This is one $f(x)$ is itself small, comparable to h. Then factors such as z/h are of order one, which is acceptable.

Newton's method for solving equations is to iterate using $g(x) = x - f(x)/y'$, where $y' = f'(x)$. Steffensen's method is to apply numerical differentiation with $h = f(x)$ to avoid having to compute a formula for the derivative. Why is this justified, if numerical differentiation is not a numerically stable process?

The answer is first that this is precisely the case when numerical differentiation is not so bad. Furthermore, the numerical differentiation is not the final stage. We are actually interested in computing $g(x)$, and the relative error amplification quantity is $(y'/g(x))\partial g(x)/\partial y' = -1/(y'g(x))f(x)$. As we approach the solution the factor $f(x)$ gets very small, so in this final stage the relative error is damped by a huge amount. Thus the iteration is very stable with respect to errors in the numerical derivative.

Problems

1. Take $f(x) = \sqrt{x}$ and consider the problem of computing the difference quotient $(f(x+h) - f(x))/h$ for small h. Discuss numerical stability of various algorithms.

2. Take $f(x) = 1/x$ and consider the problem of computing the difference quotient $(f(x+h) - f(x))/h$ for small h. Discuss numerical stability of various algorithms.

3. Consider the problem of computing the derivative $f'(x)$. One may compute either $(f(x+h) - f(x))/h$, or one may compute $(f(x+h) - f(x-h))/(2h)$. Compare these from the point of view of approximation error.

4. Say that one takes Steffensen's method with $h = f(x)^2$ instead of $h = f(x)$. What is the situation with numerical stability?

5. How about with $h = f(x)^3$?

æ

Chapter 5

Ordinary Differential Equations

5.1 Introduction

This chapter is on the numerical solution of ordinary differential equations. There is no attempt to cover a broad spectrum of methods. In fact, we stick with the simplest methods to implement, the Runge-Kutta methods.

Our main purpose is to point out that there are two different problems with the approximation of ordinary differential equations. The first is to get an accurate representation of the solution for moderate time intervals by using a small enough step size and an accurate enough approximation method. The second is to get the right asymptotic behavior of the solution for large time.

5.2 Numerical methods for scalar equations

We consider the equation

$$\frac{dy}{dt} = g(t, y) \tag{5.1}$$

with the initial condition $y = y_0$ when $t = t_0$.

The simplest numerical method is *Euler's method*, which is the first-order Runge-Kutta method. Fix a step size $h > 0$. Set $t_n = t_0 + nh$. The algorithm is

$$y_{n+1} = y_n + hg(t_n, y_n). \tag{5.2}$$

Here is a program in ISETL (Interactive Set Language).

```
g := :t,y -> y + exp(t) * cos(t):  ;
t := 0.0 ; y := 0.0 ;
dt := 0.01 ;
while t < 3.14 do
   dy := g(t,y) * dt ;
   y := y + dy ;
   t := t + dt ;
   writeln t, y , exp(t) * sin(t) ;
end ;
```

Another method is the implicit or *backward Euler's method*. This is given by

$$y_{n+1} = y_n + hg(t_{n+1}, y_{n+1}). \tag{5.3}$$

It would seem that this would never be used, because one has to solve an equation for y_{n+1} at each step. However we shall see that this method has important stability properties that can be crucial in some situations.

Euler's method is not very accurate, since the slopes are computed only at the beginning of the time step. A better method would take the average of the slopes at the beginning and at the end. This is the *implicit trapezoid method*. This is given by

$$y_{n+1} = y_n + h\frac{1}{2}[g(t_n, y_n) + g(t_{n+1}, y_{n+1})]. \tag{5.4}$$

Again this requires solving an equation for y_{n+1} at each stage.

The trouble is that we don't know the slope at the end. The solution is to use the Euler method to estimate the slope at the end. This gives an *explicit trapezoid method* which is a *second order Runge-Kutta method*. The formula is

$$y_{n+1} = y_n + h\frac{1}{2}[g(t_n, y_n) + g(t_{n+1}, y_n + hg(t_n, y_n))]. \tag{5.5}$$

Here is a program.

```
g := :t,y -> y + exp(t) * cos(t):  ;
t := 0.0 ; y := 0.0 ;
dt := 0.01 ;
while t < 3.14 do
   dye := g(t,y) * dt ;
   dy := (1/2) * ( g(t,y) + g(t+dt,y+dye) ) * dt ;
   y := y + dy ;
   t := t + dt ;
   writeln t, y , exp(t) * sin(t) ;
end ;
```

Problems

1. Find the solution of
$$\frac{dy}{dt} = y + e^t \cos t \tag{5.6}$$
with $y = 0$ when $t = 0$. What is the value of the solution at $y = \pi$.

2. Solve this numerically with Euler's method and compare.

3. Solve this numerically with the trapezoid second order Runge Kutta and compare.

4. Compare the Euler and trapezoid second order Runge Kutta method with the (left endpoint) Riemann sum and trapezoid rule methods for numerical integration.

5. Another method is to use midpoints: $dy = g(t + dt/2, y + dy_e/2)dt$ where $dy_e = g(t, y)dt$. This is another second-order Runge-Kutta method. Program this and solve the equation numerically. How does this compare in accuracy with the other methods?

5.3 Theory of scalar equations

5.3.1 Linear equations

Linear equations are easy to solve. The general homogeneous linear equation is
$$\frac{du}{dt} = a(t)u. \tag{5.7}$$
It may be solved by separation of variables $du/u = a(t)\,dt$.

It is then easy to find the solutions of the linear equation
$$\frac{dy}{dt} = a(t)y + s(t) \tag{5.8}$$
The trick is to let $u(t)$ be a solution of the corresponding homogeneous equation and try $y = c(t)u(t)$. Then it is easy to solve for $c(t)$ by integration of $dc(t)/dt = s(t)/u(t)$.

5.3.2 Autonomous equations

The general *autonomous* equation is
$$\frac{dy}{dt} = f(y). \tag{5.9}$$

An *equilibrium point* is a solution of $f(r) = 0$. For each equilibrium point we have a solution $y = r$.

Near an equilibrium point $f(y) \approx f'(r)(y - r)$. An equilibrium point r is attractive if $f'(r) < 0$ and repulsive if $f'(r) > 0$.

One can attempt to find the general solution of the equation by integrating

$$\int \frac{1}{f(y)}\, dy = \int dt. \tag{5.10}$$

Problems

1. If a population grows by $dp/dt = .05p$, how long does it take to double in size?

2. The velocity of a falling body (in the downward direction) is given by $dv/dt = g - kv$, where $g = 32$ and $k = 1/4$. If $v = 0$ when $t = 0$, what is the limiting velocity as $t \to \infty$?

3. Consider $dy/dt = ay + b$ where $y = y_0$ when $t = 0$. Solve for the case when $a \neq 0$. Fix t and find the limit of the solution y as $a \to 0$.

4. A population grows by $dp/dt = ap - bp^2$. Here $a > 0$, $b > 0$, and $0 < p < a/b$. Find the solution with $p = p_0$ at $t = 0$. Do this by letting $u = 1/p$ and solving the resulting differential equation for u.

5. Do the same problem by integrating $1/(ap - bp^2)\, dp = dt$. Use partial fractions.

6. In the same problem, find the limiting population as $t \to \infty$.

Projects

1. Write a program to solve an ordinary differential equation via the explicit trapezoid method.

2. Use your program to explore the solutions of $dx/dt = x - x^3$. Try many different initial conditions. What pattern emerges? Discuss the limit of x as $t \to \infty$ as a function of the initial condition x_0.

5.3.3 Existence

We want to explore several questions. When do solutions exist? When are they uniquely specified by the initial condition? How does one approximate them numerically?

We begin with existence. Consider the equation

$$\frac{dy}{dt} = g(t, y) \tag{5.11}$$

with initial condition $y = y_0$ when $t = t_0$. Assume that g is continuous. Then the solution always exists, at least for a short time interval near t_0. One proof of this is based on using Euler's method with step sizes $h > 0$ to generate approximate solutions. One then takes the limit $h \to 0$ and uses a compactness argument to show that these approximate a solution.

In general, however, we have only local existence. An example is given in the problems.

Problems

1. Consider the differential equation

$$\frac{dy}{dt} = y^2 \tag{5.12}$$

with initial condition $y = y_0$ when $t = 0$. Find the solution. For which t does the solution blow up?

2. Sketch the vector field in phase space (with $dx/dt = 1$). Sketch a solution that blows up.

3. Can this sort of blow up happen for linear equations? Discuss.

5.3.4 Uniqueness

Assume in addition that g has continuous derivatives. Then the solution with the given initial condition is unique. This fact is usually proved using a fixed point iteration method.

Uniqueness can fail when g is continuous but when $g(t, y)$ has infinite slope as a function of y.

Problems

1. Plot the function $g(y) = \text{sign}(y)\sqrt{|y|}$. Prove that it is continuous.

2. Plot its derivative and prove that it is not continuous.

3. Solve the differential equation

$$\frac{dy}{dt} = \text{sign}(y)\sqrt{|y|} \tag{5.13}$$

with the initial condition $y = 0$ when $t = 0$. Find all solutions for $t \geq 0$.

4. Substitute the solutions back into the equation and check that they are in fact solutions.

5. Sketch the vector field in phase space (with $dx/dt = 1$).

6. Consider the backward Euler's method for this example. What ambiguity is there in the numerical solution?

5.3.5 Forced oscillations

We consider the non-linear equation

$$\frac{dy}{dt} = g(t, y). \tag{5.14}$$

Assume that $g(t, y)$ has period T, that is, $g(t + T, y) = g(t, y)$. It will not necessarily be the case that all solutions have period T. However there may be a special steady-state solution that has period T.

Here is the outline of the argument. Assume that $a < b$ and that $g(t, a) \geq 0$ for all t and $g(t, b) \leq 0$ for all t. Then no solution can leave the interval $[a, b]$. Thus if $y = \phi(t, y_0)$ is the solution with $y = y_0$ at $t = 0$, then $h(y_0) = \phi(T, y_0)$ is a continuous function from $[a, b]$ to itself. It follows that h has a fixed point. But then if we take the initial condition to be this fixed point we get a periodic solution.

We can sometimes get to this fixed point by iterations. Let y' be $\partial y/\partial y_0$. Then

$$\frac{dy'}{dt} = \frac{\partial g(t, y)}{\partial y} y'. \tag{5.15}$$

Also $y' = 1$ at $t = 0$ and $y' = h'(y_0)$ at $t = T$. It follows that $h'(y_0) > 0$.

Assume that $\partial g(t, y)/\partial y < 0$. Then $h'(y_0) < 1$ and so we can hope that fixed point iterations of h converge. This would say that every solution in the interval converges to the periodic solution.

Problems

1. Consider the equation

$$\frac{dy}{dt} = g(y) + s(t) \tag{5.16}$$

with periodic forcing function $s(t)$. Find conditions that guarantee that this has a periodic solution.

2. Apply this to the equation

$$\frac{dy}{dt} = ay - by^2 + c\sin(\omega t). \tag{5.17}$$

3. Experiment with numerical solutions. Which solutions converge to the periodic solution?

5.4 Theory of numerical methods

5.4.1 Fixed time, small step size

We want to approximate the solution of the differential equation

$$\frac{dy}{dt} = f(t, y) \tag{5.18}$$

with initial condition $y = y_0$ at $t = 0$. For convenience we denote the solution of the equation at a point where $t = a$ by $y(a)$.

The general method is to find a function $\phi(t, y, h)$ and compute

$$y_{n+1} = y_n + h\phi(t_n, y_n, h). \tag{5.19}$$

Here $h > 0$ is the *step size* and $t_n = nh$.

Example: Let $\phi(t, y, h) = f(t, y)$. This is *Euler's method*.

Example: Let $\phi(t, y, h) = (1/2)[f(t, y) + f(t + h, y + hf(t, y))]$. This is a second-order Runge-Kutta method which we will call the *explicit trapezoid method*.

Example: Let $\phi(t, y, h) = f(t + h/2, y + hf(t, y)/2)$. This is another second-order Runge-Kutta method, the *explicit midpoint method*.

We will always pick the method so that $\phi(t, y, 0) = f(t, y)$. Such a method is said to be *consistent*.

Why is one method better than other. One criterion is obtained by looking at the exact solution $y(t)$. If

$$y(t_{n+1}) = y(t_n) + h\phi(t_n, y(t_n), h) + T_{n+1}, \tag{5.20}$$

then the remainder term T_n is called the *local truncation error*.

The local truncation error may be computed as a Taylor series about t_n in powers of h. If the local truncation error only contains only powers of order $p + 1$ or more, then the method is said to have order at least p.

Example: Euler's method has order one.

Example: The explicit trapezoid method described above has order two.

Remark: A consistent method has order at least one.

Let $\epsilon_n = y(t_n) - y_n$. This is the *error* at step n. We see that

$$\epsilon_{n+1} - \epsilon_n = h[\phi(t_n, y(t_n), h) - \phi(t_n, y_n, h)] + T_{n+1} \tag{5.21}$$

Assume that we have the inequality

$$|\phi(t, y_1, h) - \phi(t, y_2, h)| \leq L|y_1 - y_2| \tag{5.22}$$

for some constant L. This would follow from a bound on the y partial derivative of ϕ. We call this the *slope bound*.

Assume also that we have a bound $T_{n+1} \leq Kh^{p+1}$. We call this the *local truncation error bound*.

Theorem 5.4.1 *Assume that the one-step numerical method for solving the ordinary differential equation $dy/dt = f(t, y)$ satisfies the slope bound and the local truncation error bound. Then the error satisfies the* global truncation error bound

$$|\epsilon_n| \leq Kh^p \frac{e^{Lt_n} - 1}{L}. \tag{5.23}$$

This bound is a worst case analysis, and the error may not be nearly as large as the bound. But there are cases when it can be this bad. Notice that for fixed time t_n the bound gets better and better as $h \to 0$. In fact, when the order p is large, the improvement is dramatic as h becomes very small.

On the other hand, for fixed h, even very small, this bound becomes very large as $t_n \to \infty$.

Obviously, it is often desirable to take p to be large. It is possible to classify the Runge-Kutta methods of order p, at least when p is not too large. The usual situation is to use a method of order 2 for rough calculation and a method of order three or four for more accurate calculation.

We begin by classifying the explicit Runge-Kutta methods of order 2. We take

$$\phi(t, y, h) = (1 - c)f(t, y) + cf(t + ah, y + ahf(t, y)). \tag{5.24}$$

We see that every such method is consistent, and hence of order one. The condition that the method be of order 2 works out to be that $ac = 1/2$.

There is a similar classification of methods of order 3 and 4. In each case there is a two-parameter family of Runge-Kutta methods. By far the most commonly used method is a particular method of order 4. (There are methods of higher order, but they tend to be cumbersome.)

Problems

1. Prove the bound on the global truncation error.

2. Carry out the classification of methods of order 2.

5.4.2 Fixed step size, long time

In order to study the long time behavior, it is useful to begin with the *autonomous equation*

$$\frac{dy}{dt} = f(y). \tag{5.25}$$

If r is such that $f(r) = 0$, then r is a *stationary* or *equilibrium* point. If also $f'(r) < 0$, then r is *stable*.

The numerical method is now of the form

$$y_{n+1} = y_n + h\phi(y_n, h). \tag{5.26}$$

It is natural to require that the method satisfies the condition that $f(y) = 0$ implies $\phi(y, h) = 0$.

This is an iteration with the iteration function $g(y) = y + h\phi(y, h)$. Under the above requirement the equilibrium point r is a *fixed point* with $g(r) = r$. Such a fixed point r is *stable* if $g'(r) = 1 + h\phi'(r, h)$ is strictly less then one in absolute value.

If $f'(y) < 0$ for y near r, then for h small one might expect that $\phi'(r, h) < 0$ and hence $g'(r) < 1$. Furthermore, for h small enough we would have $g'(s) > -1$. So a stable equilibrium of the equation should imply a stable fixed point of the numerical scheme, at least for small values of h.

How small? We want $g'(r) = 1 + h\phi'(r, h) > -1$, or $h\phi'(r, h) > -2$. Thus a rough criterion would be $hf'(r) > -2$.

In some problems, as we shall see, there are two time scales. One time scale suggests taking a comparatively large value of time step h for the integration. The other time scale is determined by the reciprocal of the magnitude of $f'(r)$, and this can be very small. If these scales are so different that the criterion is violated, then the problem is said to be *stiff*. (We shall give a more precise definition later.)

A class of problems where stiffness is involved is for equations of the form

$$\frac{dy}{dt} = f(t, y), \tag{5.27}$$

where there is a function $y = r(t)$ with $f(t, r(t)) = 0$ and with $\partial f(t, r(t))/\partial y < 0$ and very negative. The dependence of $r(t)$ on t is slow compared with the fast decay of the solution y to $y = r(t)$. Thus one might want to take a moderate sized time step h suitable for tracking $y = r(t)$. However this would be too large for tracking the decay in detail, which we might not even care to do. Thus we need a numerical method that gives the decay without worrying about the details of how it happens.

What can we do for a stiff problem? One solution is to use implicit methods.

The general implicit method is to find a function $\phi(t, y, z, h)$ and compute

$$y_{n+1} = y_n + h\phi(t_n, y_n, y_{n+1}, h). \tag{5.28}$$

Here $h > 0$ is the *step size* and $t_n = nh$.

Example: Let $\phi(t, y, z, h) = f(t, z)$. This is *backward Euler's method*.

Example: Let $\phi(t, y, z, h) = (1/2)[f(t, y) + f(t + h, z)]$. This is a second-order implicit Runge-Kutta method often called the *implicit trapezoid method*.

Example: Let $\phi(t, y, z, h) = f(t + h/2, (y + z)/2)$. This is another second order implicit Runge-Kutta method, known as the *implicit midpoint method*.

The difficulty with an implicit method is that at each stage one must solve an equation for y_{n+1}. However an implicit method may be more useful for stiff problems.

Consider the special case of an autonomous equation. The implicit method amounts to iteration by an iteration function defined implicitly by

$$g(y) = y + h\phi(y, g(y), h). \tag{5.29}$$

The derivative is given by

$$g'(y) = \frac{1 + h\phi_1(y, g(y), h)}{1 - h\phi_2(y, g(y), h)}. \tag{5.30}$$

Here $\phi_1(y, z, h)$ and $\phi_2(y, z, h)$ denote the partial derivatives of $\phi(y, z, h)$ with respect to y and z, respectively. At a fixed point r with $g(r) = r$ this is

$$g'(r) = \frac{1 + h\phi_1(r, r, h)}{1 - h\phi_2(r, r, h)}. \tag{5.31}$$

The condition that $|g'(r)| < 1|$ translates to

$$|1 + h\phi_1(r, r, h)| < |1 - h\phi_2(r, r, h)|. \tag{5.32}$$

If the two partial derivatives are both strictly negative, then this condition is guaranteed for all $h > 0$, no matter how large, by the inequality

$$-\phi_1(r, r, h) \le -\phi_2(r, r, h). \tag{5.33}$$

This says that the implicit method must have at least as much dependence on the future as on the past. Thus a stiff problem requires implicitness.

The implicit methods require solving equations at each step. In some cases this may be done by algebraic manipulation. In other cases an iteration method must be used.

The most obvious iteration method is to use the iteration function

$$s(z) = y_n + h\phi(t_n, y_n, z, h). \tag{5.34}$$

One could start this iteration with y_n. The fixed point of this function is the desired y_{n+1}. The trouble with this method is that $s'(z) = h\phi_2(t_n, y_n, z, h)$ and for a stiff problem this is very large. So the iteration will presumably not converge.

How about Newton's method? This is iteration with

$$t(z) = z - \frac{z - y_n - h\phi(t, y_n, z, h)}{1 - h\phi_2(t, y_n, z, h)}. \tag{5.35}$$

This should work, but the irritation is that one must compute a partial derivative.

Problems

1. Consider a problem of the form $dy/dt = -a(y - r(t))$ where $a > 0$ is large but $r(t)$ is slowly varying. Find the explicit solution of the initial value problem when $y = y_0$ at $t = 0$. Find the limit of the solution at time t as $a \to \infty$.

2. Take $a = 10000$ and $r(t) = \sin t$. Use $y_0 = 1$. The problem is to experiment with explicit methods with different step sizes. Solve the problem on the interval from $t = 0$ to $t = \pi$. Use Euler's method with step sizes $h = .01, .001, .0001, .00001$. Describe your computational experience and relate it to theory.

3. The next problem is to do the same experiment with stable implicit methods. Use the backward Euler's method with step sizes $h = .01, .001$.

4. Consider the general implicit method of the form

   $$\phi(t, y, z, h) = c_1 f(t + a_1 h, y + a_1(z - y)) + c_2 f(t + a_2 h, y + a_2(z - y)) \tag{5.36}$$

   Find the condition that ensures that this is first order.

5. Find the condition that ensures that it is second order.

6. Consider an autonomous equation and assume that $f'(y) < 0$ for all y in some interval in the region of interest. Find the condition for stability of the method at a fixed point for a stiff problem.

7. Are there second order methods for stiff problems that are stable? Discuss.

8. We have required that every zero of $f(y)$ also be a zero of $\phi(y, h)$. When is this satisfied for Runge-Kutta methods?

9. Consider a Taylor method of the form $\phi(y, h) = f(y)+(1/2)f'(y)f(y)h$. When is the requirement satisfied for such Taylor methods?

5.5 Systems

5.5.1 Introduction

We now turn to systems of ordinary differential equations. For simplicity, we concentrate on autonomous systems consisting of two equations. The general form is

$$\frac{dx}{dt} = f(x, y) \tag{5.37}$$

$$\frac{dy}{dt} = g(x, y). \tag{5.38}$$

Notice that this includes as a special case the equation $dy/dt = g(t, y)$. Thus may be written as a system by writing it as

$$\frac{dx}{dt} = 1 \tag{5.39}$$

$$\frac{dy}{dt} = g(x, y). \tag{5.40}$$

In general, if we have a system in n variables with explicit time dependence, then we may use the same trick to get an autonomous system in $n + 1$ variables.

We may think of an autonomous system as being given by a vector field. In the case we are considering this is a vector field in the plane with components $f(x, y)$ and $g(x, y)$. If we change coordinates in the plane, then we change these coordinates.

In general, the matrix of partial derivatives of this vector field transforms in a complicated way under change of coordinates in the plane. However at a zero of the vector field the matrix undergoes a similarity transformation. Hence linear algebra is relevant!

In particular, two eigenvalues (with negative real parts) of the linearization at a stable fixed point determine two rates of approach to equilibrium. In the case when these rates are very different we have a stiff problem.

5.5.2 Linear constant coefficient equations

The homogeneous linear constant coefficient system is of the form

$$\frac{dx}{dt} = ax + by \tag{5.41}$$

$$\frac{dy}{dt} = cx + dy. \tag{5.42}$$

Try a solution of the form

$$x = ve^{\lambda t} \tag{5.43}$$
$$y = we^{\lambda t}. \tag{5.44}$$

We obtain the *eigenvalue* equation

$$av + bw = \lambda v \tag{5.45}$$
$$cv + dw = \lambda w. \tag{5.46}$$

This has a non-zero solution only when λ satisfies $\lambda^2-(a+d)\lambda+ad-bc=0$.

We can express the same ideas in matrix notation. The equation is

$$\frac{d\mathbf{x}}{dt} = A\mathbf{x}. \tag{5.47}$$

The trial solution is

$$\mathbf{x} = \mathbf{v}e^{\lambda t}. \tag{5.48}$$

The eigenvalue equation is

$$A\mathbf{v} = \lambda\mathbf{v}. \tag{5.49}$$

This has a non-zero solution only when $\det(\lambda I - A) = 0$.

Growth and Decay

The first case is real and unequal eigenvalues $\lambda_1 \neq \lambda_2$. This takes place when $(a-d)^2 + 4bc > 0$. There are two solutions corresponding to two independent eigenvectors. The general solution is a linear combination of these two. In matrix notation this is

$$\mathbf{x} = c_1\mathbf{v}_1e^{\lambda_1 t} + c_2\mathbf{v}_2e^{\lambda_2 t}. \tag{5.50}$$

When the two eigenvalues are both positive or both negative, the equilibrium is called a *node*. When one eigenvalue is positive and one is negative, it is called a *saddle*. An attractive node corresponds to an *overdamped* oscillator.

Oscillation

The second case is complex conjugate unequal eigenvalues $\lambda = \alpha + i\omega$ and $\bar{\lambda} = \alpha - i\omega$ with $\alpha = (a+d)/2$ and $\omega > 0$. This takes place when $(a-d)^2+4bc<0$. There are two independent complex conjugate solutions. These are expressed in terms of $e^{\lambda t} = e^{\alpha t}e^{i\omega t}$ and $e^{\bar{\lambda} t} = e^{\alpha t}e^{-i\omega t}$. Their real

and imaginary parts are independent real solutions. These are expressed in terms of $e^{\alpha t}\cos(\omega t)$ and $e^{\alpha t}\sin(\omega t)$.

In matrix notation we have complex eigenvectors $\mathbf{u}\pm i\mathbf{v}$ and the solutions are

$$x = (c_1 \pm ic_2)e^{\alpha t}e^{\pm i\omega t}(\mathbf{u} \pm i\mathbf{v}). \tag{5.51}$$

Taking the real part gives

$$x = c_1 e^{\alpha t}(\cos(\omega t)\mathbf{u} - \sin(\omega t)\mathbf{v}) - c_2 e^{\alpha t}(\sin(\omega t)\mathbf{u} + \cos(\omega t)\mathbf{v}). \tag{5.52}$$

If we write $c_1 \pm ic_2 = ce^{\pm i\theta}$, these take the alternate forms

$$x = ce^{\alpha t}e^{\pm i(\omega t+\theta)}(\mathbf{u} \pm i\mathbf{v}). \tag{5.53}$$

and

$$x = ce^{\alpha t}(\cos(\omega t + \theta)\mathbf{u} - \sin(\omega t + \theta)\mathbf{v}). \tag{5.54}$$

From this we see that the solution is characterized by an amplitude c and a phase θ. When the two conjugate eigenvalues are pure imaginary, the equilibrium is called a *center*. When the two conjugate eigenvalues have a non-zero real part, it is called a *spiral* (or a *focus*). An center corresponds to an *undamped* oscillator. An attractive spiral corresponds to an *underdamped* oscillator.

Shearing

The remaining case is when there is only one eigenvalue $\lambda = (a+d)/2$. This takes place when $(a-d)^2 + 4bc = 0$. In this case we neeed to try a solution of the form

$$\begin{aligned} x &= pe^{\lambda t} + vte^{\lambda t} && (5.55)\\ y &= qe^{\lambda t} + wte^{\lambda t}. && (5.56) \end{aligned}$$

We obtain the same eigenvalue equation together with the equation

$$\begin{aligned} ap + bq &= \lambda p + v && (5.57)\\ cp + dq &= \lambda q + w. && (5.58) \end{aligned}$$

In practice we do not need to solve for the eigenvector: we merely take p, q determined by the initial conditions and use the last equation to solve for v, w.

Im matrix notation this becomes

$$\mathbf{x} = \mathbf{p}e^{\lambda t} + \mathbf{v}te^{\lambda t} \tag{5.59}$$

with

$$A\mathbf{p} = \lambda\mathbf{p} + \mathbf{v}. \tag{5.60}$$

Inhomogeneous equations

The general linear constant coefficient equation is

$$\frac{d\mathbf{x}}{dt} = A\mathbf{x} + \mathbf{r}. \tag{5.61}$$

When A is non-singular we may rewrite this as

$$\frac{d\mathbf{x}}{dt} = A(\mathbf{x} - \mathbf{s}), \tag{5.62}$$

where $\mathbf{s} = -A^{-1}\mathbf{r}$ is constant. Thus $\mathbf{x} = \mathbf{s}$ is a particular solution. The general solution is the sum of this particular solution with the general solution of the homogeneous equation.

Problems

1. Find the general solution of the system

$$\begin{aligned} \frac{dx}{dt} &= x + 3y \\ \frac{dy}{dt} &= 5x + 3y. \end{aligned}$$

2. Find the solution of this equation with the initial condition $x = 1$ and $y = 3$ when $t = 0$.

3. Sketch the vector field in the above problem. Sketch the given solution in the x, y phase space. Experiment to find a solution that passes very close to the origin, and sketch it.

4. Write the Taylor series of e^z about $z = 0$. Plug in $z = i\theta$, where $i^2 = -1$. Show that $e^{i\theta} = \cos\theta + i\sin\theta$.

5. Find the general solution of the system

$$\begin{aligned} \frac{dx}{dt} &= x + 5y \\ \frac{dy}{dt} &= -x - 3y. \end{aligned}$$

6. Find the solution of this equation with the initial condition $x = 5$ and $y = 4$ when $t = 0$.

7. Sketch the vector field in the above problem. Find the orbit of the given solution in phase space. Also plot x versus t and y versus t.

8. A frictionless spring has mass $m > 0$ and spring constant $k > 0$. Its displacement and velocity x and y satisfy

$$\begin{aligned} \frac{dx}{dt} &= y \\ m\frac{dy}{dt} &= -kx. \end{aligned}$$

Describe the motion.

9. A spring has mass $m > 0$ and spring constant $k > 0$ and friction constant $f > 0$. Its displacement and velocity x and y satisfy

$$\begin{aligned} \frac{dx}{dt} &= y \\ m\frac{dy}{dt} &= -kx - fy. \end{aligned}$$

Describe the motion in the case $f^2 - 4k < 0$ (underdamped).

10. Take $m = 1$ and $k = 1$ and $f = 0.1$. Sketch the vector field and the solution in the phase plane. Also sketch x as a function of t.

11. In the preceding problem, describe the motion in the case $f^2 - 4k > 0$ (overdamped). Is it possible for the oscillator displacement x to overshoot the origin? If so, how many times?

12. An object has mass $m > 0$ and its displacement and velocity x and y satisfy

$$\begin{aligned} \frac{dx}{dt} &= y \\ m\frac{dy}{dt} &= 0. \end{aligned}$$

Describe the motion.

13. Solve the above equation with many initial condition with $x = 0$ and with varying value of y. Run the solution with these initial conditions for a short time interval. Why can this be described as "shear"?

5.5.3 Stiff systems

Stiff systems are ones where the eigenvalues near an equilibrium point have real parts describing very different decay rates. This situation may be illustrated by simple homogeneous constant coefficient systems such as an oscillator.

Problems

1. Consider the system

$$\frac{dx}{dt} = v \tag{5.63}$$
$$m\frac{dv}{dt} = -kx - cv, \tag{5.64}$$

where $m > 0$ is the mass, $k > 0$ is the spring constant, and $c > 0$ is the friction constant. We will be interested in the highly damped situations, when m is small relative to k and c. Take k and c each 10 times the size of m. Find the eigenvalues and find approximate numerical expressions for them. Find approximate numerical expressions for the eigenvectors. Describe the corresponding solutions.

2. In this preceding problem, which eigenvalue describes very rapid motion in phase space, and which eigenvalue describes slow motion in phase space? Describe the solution starting from an arbitrary initial condition. There are two stages to the motion. The first takes place until t is comparable to m/c. The second takes place until t is comparable to c/k. Describe the two stages in terms of motion in phase space. Which variable or variables (displacement x and velocity v) are making the main change in each of these two stages?

3. Produce pictures of the solutions in phase space. Do this with enough initial conditions to confirm the analysis in the last problem. Sketch the results. Confirm them by x versus t and v versus t pictures.

5.5.4 Autonomous Systems

The general *autonomous* system is

$$\frac{dx}{dt} = f(x, y) \tag{5.65}$$
$$\frac{dy}{dt} = g(x, y). \tag{5.66}$$

An *equilibrium point* is a solution of $f(r, s) = 0$ and $g(r, s) = 0$. For each equilibrium point we have a solution $x = r$ and $y = s$.

Near an equilibrium point

$$f(x, y) \approx a(x - r) + b(y - s) \tag{5.67}$$
$$g(x, y) \approx c(x - r) + d(y - s), \tag{5.68}$$

where $a = \partial f(x,y)/\partial x$, $b = \partial f(x,y)/\partial y$, $c = \partial g(x,y)/\partial x$, and $d = \partial g(x,y)/\partial y$, all evaluated at $x = r$ and $y = s$. So near the equilibrium point the equation looks like a linear equation.

Assume that the eigenvalues of the linear equation are real. Then the equilibrium point is attractive if they are both negative. On the other hand, assume that the eigenvalues of the linear equation are complex conjugates. Then the equilibrium point is attractive if the real part is negative. In general the equilibrium point is classified by the behavior of the linearized equation at that point.

A first example is the non-linear pendulum equation. This is

$$\frac{dx}{dt} = y \tag{5.69}$$

$$ml\frac{dy}{dt} = -mg\sin(x) - cy. \tag{5.70}$$

Here x is the angle and y is the angular velocity. The parameters are the mass $m > 0$, the length $l > 0$, and the gravitational acceleration $g > 0$. There may also be a friction coefficient $c \geq 0$. The first equation is the definition of angular velocity. The second equation is Newton's law of motion: mass times acceleration equals force.

There are two interesting equilibrium situations. One is where $x = 0$ and $y = 0$. In the case we use $\sin(x) \approx x$ to find the linear approximation The other interesting situation is when $x - \pi = 0$ and $y = 0$. In this case we use $\sin(x) \approx -(x - \pi)$. The minus sign makes a crucial difference.

A second example is the predator-prey system. This is

$$\frac{dx}{dt} = (a - by - mx)x \tag{5.71}$$

$$\frac{dy}{dt} = (cx - d - ny)y. \tag{5.72}$$

Here x is the prey and y is the predator. The prey equation says that the prey has a natural growth rate a, are eaten by the predators at rate by, and compete with themselves with rate mx. The predator equation says that the predators have a growth rate $cx - d$ at food level x and compete with themselves at rate ny. The parameters are strictly positive, except that we allow the special case $m = 0$ and $n = 0$ with no internal competition. We are only interested in the situation $x \geq 0$ and $y \geq 0$.

There are several equilibria. One corresponds to total extinction. Also when $m > 0$ one can have a situation when the predator is extinct and where $x = a/m$ is the natural prey carrying capacity. Whem $m = 0$, on the other hand, there is there is no natural limit to the size of the prey population: we interpret $a/m = +\infty$.

The most interesting equilibrium takes place when the natural predator growth rate $cx - d$ with $x = a/m$ at the prey carrying capacity is positive. This says that the predator can live off the land.

Problems

1. For the pendulum problem with no friction, find the linearization at $x = 0$, $y = 0$. Discuss the nature of the equilibrium.

2. Consider the pendulum problem. Find oscillatory solutions that are near the zero solution, but not too near. How large can the solutions be before the pendulum can no longer be used as a clock?

3. For the pendulum problem with no friction, find the linearization at $x = \pi$, $y = 0$. Discuss the nature of the equilibrium.

4. Find at least two different kinds of oscillatory solutions that pass near $x = \pi$, $y = 0$. Sketch plots that illustrate these different kinds of solutions.

5. For the pendulum problem, describe the nature of the two equilibria when there is friction.

6. Consider the predator-prey equations with internal competition. Find the nature of the equilibrium corresponding to total extinction.

7. Find the nature of the equilibrium corresponding to extinction of the predators. There are two situations, depending on the sign of the predator natural growth rate.

8. Find the nature of the equilibrium corresponding to coexistence. Discuss its stability.

9. Sketch representative solutions.

10. Find the nature of the equilibrium corresponding to coexistence when there is no internal competition.

11. Sketch representative solutions.

5.5.5 Limit cycles

Now we come to an essentially non-linear effect: oscillations that are stabilized by the non-linearity.

The classic example is

$$\frac{dx}{dt} = v \tag{5.73}$$

$$\frac{dv}{dt} = -kx - g(x)v. \tag{5.74}$$

This is an oscillator in which the friction coefficient $g(x)$ is a function of position. There is a constant $r > 0$ such that $g(x) < 0$ for $|x| < r$ and $g(x) > 0$ for $|x| > r$. Thus when $|x|$ is small the oscillator gets a boost. A standard example is $g(x) = c(x^2 - r^2)$.

Change variables to $y = v + G(x)$, where $G'(x) = g(x)$. Then this same oscillator becomes

$$\frac{dx}{dt} = y - G(x) \tag{5.75}$$

$$\frac{dy}{dt} = -kx. \tag{5.76}$$

The equation is often studied in this form.

Problems

1. Take the van der Pol oscillator in x, y space with $G(x) = x^3 - ax$. Investigate the Hopf bifurcation. Sketch your results.

2. Take the non-linear van der Pol oscillator in x, v space with $g(x) = a(x^2 - 1)$. Take $a > 0$ increasingly large. The result is a relaxation oscillator. Make plots in the x, v plane. Also make x versus t and v versus t plots and interpret them.

æ

Chapter 6

Fourier transforms

6.1 Groups

We want to consider several variants of the Fourier transform at once. The unifying idea is that the Fourier transform deals with complex functions defined on commutative groups. (Recall that a commutative group is a set with operations of addition and subtraction that satisfy the usual properties.) Here are the groups that we shall consider.

The first is the group of all real numbers.

The second is the group of all integer multiples of $n\Delta x$, where $\Delta x > 0$ is a fixed real number. This is a subgroup of the real numbers, since the sum or difference of any two $n\Delta x$ is also of the same form.

The third is the group of all real numbers mod L, where $L > 0$ is fixed. This is the circle group, where the circle has circumference L. Two real numbers determine the same element if they differ by an integer multiple of L. Thus the circle group is a quotient group of the real numbers.

The final group is the group of all integer multiples $n\Delta x$ mod $L = N\Delta x$. This is a subgroup of the circle group. It is also a quotient group of the integer group. It is finite with precisely N elements.

6.2 Integers mod N

We first consider the Fourier transform on the group G of integers mod N. This is a finite group with elements $\{0, 1, \ldots, N-1\}$ extended periodically.

It is helpful to think of the integers as being embedded in the reals at a spacing Δx. Then the integers mod N may be thought of as embedded in the reals mod $L = N\Delta x$.

We consider a complex function f on the group G. This may also be thought of as a function on the integers spaced by Δx with period $L = N\Delta x$.

The Fourier transform is another complex function defined on the dual group $\hat{G}$. This is another group of integers mod N, but it is regarded as embedded in the real line with spacing Δk, where $\Delta k = 2\pi/(N\Delta x) = 2\pi/L$. Thus the Fourier transform is periodic with period $N\Delta k = 2\pi/\Delta x$.

We think of frequencies from 0 to $N\Delta k/2$ as positive frequencies. We think of freqencies from $N\Delta k/2$ to $N\Delta k$ as negative frequencies. The frequency of least oscillation is 0, which is identified with $N\Delta k$. The frequency of maximum oscillation is $N\Delta k/2$.

The *Fourier transform* is

$$\hat{f}(k) = \sum_{x \in G} e^{-ikx} f(x)\, \Delta x, \tag{6.1}$$

where the sum is over N consecutive points spaced at interval Δx.

This may be written more explicitly as

$$\hat{f}(m\Delta k) = \sum_{n=0}^{N-1} e^{\frac{-i2\pi mn}{N}} f(n\Delta x)\, \Delta x. \tag{6.2}$$

The *inversion formula* is then

$$f(x) = \sum_{k \in \hat{G}} e^{ikx} \hat{f}(k)\, \Delta k/(2\pi), \tag{6.3}$$

where the sum is over N consecutive points k with spacing Δk.

This may be written more explicitly as

$$f(n\Delta x) = \sum_{m=0}^{N-1} e^{\frac{i2\pi mn}{N}} \hat{f}(m\Delta k)\, \Delta k/(2\pi). \tag{6.4}$$

Here is a proof of the inversion formula. We wish to show that

$$f(x) = \frac{1}{N} \sum_k e^{ikx} \sum_y e^{-iky} f(y) = \frac{1}{N} \sum_y \sum_k e^{ik(x-y)} f(y). \tag{6.5}$$

But it is easy to sum the geometric series

$$\frac{1}{N} \sum_k e^{ikz} \tag{6.6}$$

and see that it is zero unless $z = 0 \bmod L$, in which case it is one.

6.3 The circle

We now consider the Fourier transform on the circle group G, thought of as the reals mod L.

The dual group in this case is $\hat{G}$, thought of as the integers spaced with interval $\Delta k = 2\pi/L$.

The Fourier transform is

$$\hat{f}(k) = \int_0^L e^{-ikx} f(x)\, dx. \tag{6.7}$$

The inversion formula is

$$f(x) = \frac{1}{L} \sum_k e^{ikx} \hat{f}(k). \tag{6.8}$$

These formulas may be obtained from the finite case. Take the sum over k to run from $-N\Delta k/2 = -\pi/\Delta x$ to $N\Delta k/2 = \pi/\Delta x$, counting the end point at most once. Then let $N \to \infty$ and $\Delta x \to 0$ keeping $L = N\Delta x$ fixed.

6.4 The integers

We now consider the Fourier transform on the integers, thought of as spaced at interval Δx.

The dual group in this case is $\hat{G}$, thought of as the circle of circumference $2\pi/\Delta x$.

The Fourier transform is

$$\hat{f}(k) = \sum e^{-ikx} f(x)\, \Delta x. \tag{6.9}$$

The inversion formula is

$$f(x) = \int_0^{2\pi/\Delta x} e^{ikx} \hat{f}(k)\, dk/(2\pi). \tag{6.10}$$

These formulas may be obtained from the finite case by taking the sum on x to range from $-N\Delta x/2$ to $N\Delta x/2$ (counting end points only once) and then letting $N \to \infty$ with fixed Δx.

6.5 The reals

We now consider the Fourier transform on the group G of reals.

The dual group in this case is $\hat{G}$, again the reals.

The Fourier transform is

$$\hat{f}(k) = \int_{-\infty}^{\infty} e^{-ikx} f(x)\, dx. \tag{6.11}$$

The inversion formula is

$$f(x) = \int_{-\infty}^{\infty} e^{ikx} \hat{f}(k)\, dk/(2\pi). \tag{6.12}$$

These formulas may be obtained from the circle case integrating x from $-L/2$ to $L/2$ and letting $L \to \infty$ or from the integer case by integrating k from $-\pi/\Delta x$ to $\pi/\Delta x$ and letting $\Delta x \to 0$.

The notation has been chosen to suggest that x is position and k is wave number (spatial frequency). It is also common to find another notation in which t is time and ω is (temporal) frequency. For the record, here are the formulas in the other notation.

The Fourier transform is

$$\hat{f}(\omega) = \int_{-\infty}^{\infty} e^{-i\omega t} f(t)\, dt. \tag{6.13}$$

The inversion formula is

$$f(t) = \int_{-\infty}^{\infty} e^{i\omega t} \hat{f}(\omega)\, d\omega/(2\pi). \tag{6.14}$$

6.6 Translation Invariant Operators

We now want to explore the uses of the Fourier transform. It is convenient to look at the transform in a unified framework. Thus we write in all cases the Fourier transform as

$$\hat{f}(k) = \int_{G} e^{-ikx} f(x)\, dx. \tag{6.15}$$

The inversion formula is

$$f(x) = \int_{\hat{G}} e^{ikx} \hat{f}(k)\, dk/(2\pi). \tag{6.16}$$

The fundamental observation is that the Fourier transform does good things with respect to translation. Define $U_y f(x) = f(x - y)$. Then we have that the Fourier transform of $U_y f$ at k is $e^{-iky}\hat{f}(k)$. In other words, translation is converted to a multiplicative factor.

Here is a generalization. Define the convolution $g * f$ of two functions as the function defined by

$$(g * f)(x) = \int_G g(y) f(x - y)\, dy. \tag{6.17}$$

Thus convolution is weighted translation. The Fourier transform of $g * f$ evaluated at k is then $\hat{g}(k)\hat{f}(k)$. Again this is a multiplicative factor.

Here are some special cases. Let

$$\delta_+ f(x) = \frac{1}{\Delta x}(f(x + \Delta x) - f(x)). \tag{6.18}$$

This has Fourier transform $(1/\Delta x)(e^{ik\Delta x} - 1)\hat{f}(k)$. Let

$$\delta_- f(x) = \frac{1}{\Delta x}(f(x) - f(x - \Delta x)). \tag{6.19}$$

This has Fourier transform $(1/\Delta x)(1 - e^{-ik\Delta x})\hat{f}(k)$. Let

$$\delta_0 f(x) = \frac{1}{2}(\delta_+ + \delta_-)f(x) = \frac{1}{2\Delta x}(f(x + \Delta x) - f(x - \Delta x)). \tag{6.20}$$

This has Fourier transform $(i/\Delta x)\sin(k\Delta x)\hat{f}(k)$. We may take $\Delta x \to 0$ in these formulas and conclude that the first derivative is represented in the Fourier transform by multiplication by ik.

We may also represent the second derivative in the same way. The most useful formula is

$$\delta^2 f(x) = \delta_+\delta_- f(x) = \frac{1}{(\Delta x)^2}(f(x + \Delta x) - 2f(x) + f(x - \Delta x)). \tag{6.21}$$

This has Fourier transform $-(2/\Delta x)^2 \sin^2(k\Delta x/2)\hat{f}(k)$. In the limit $\Delta x \to 0$ this gives $-k^2\hat{f}(k)$.

It is not hard to check that the Fourier transform of the reversed conjugate $\overline{f(-x)}$ is the conjugate $\hat{f}(k)$. It follows that the Fourier transform of the "correlation" $\int_G \overline{f(y - x)}g(y)\, dy$ is $\overline{\hat{f}(k)}\hat{g}(k)$. From the inversion formula it follows that

$$\int_G \overline{f(y - x)}g(y)\, dy = \int_{\hat{G}} e^{ikx}\overline{\hat{f}(k)}\hat{g}(k)\, dk/(2\pi). \tag{6.22}$$

A very important special case is obtained by taking $f = g$ and $x = 0$. This gives the Plancherel theorem

$$\int_G |f(y)|^2\, dy = \int_{\hat{G}} |f(k)|^2\, dk/(2\pi). \tag{6.23}$$

6.7 Subgroups

We now want to consider a more complicated situation. Let G be a group. Let H be a discrete subgroup. Thus for some $\Delta y > 0$ the group H consists of the multiples $n\Delta y$ of Δy. We think of this subgroup as consisting of uniformly spaced sampling points. Let Q be the quotient group, where we identify multiples of Δy with zero.

The group G has a dual group $\hat{G}$. The elements of $\hat{G}$ that are multiples of $2\pi/\Delta y$ form the dual group $\hat{Q}$, which is a subgroup of $\hat{G}$. The quotient group, where we identify multiples of $2\pi/\Delta y$ with zero, turns out to be $\hat{H}$. These dual groups may all be thought of as consisting of angular frequencies.

We can summarize this situation in diagrams

$$H \longrightarrow G \longrightarrow Q \tag{6.24}$$

and

$$\hat{Q} \longrightarrow \hat{G} \longrightarrow \hat{H}. \tag{6.25}$$

The arrow between two groups means that elements of one group uniquely determine elements of the next group. Furthermore, an element of the group G or $\hat{G}$ that is determined by an element of the group on the left itself determines the element 0 of the group on the right.

The first main example is when G is the reals, H is the subgroup of integer multiples of Δy, and Q is the circle of circumference Δy. Then $\hat{G}$ is the reals (considered as angular frequencies), $\hat{Q}$ is the subgroup of multiples of $\Delta r = 2\pi/\Delta y$, and $\hat{H}$ is the circle of circumference Δr.

The other main example is when G is the circle of circumference $L = N\Delta y$, H is the subgroup of order N consisting of integer multiples of Δy, and Q is the circle of circumference Δy. Then $\hat{G}$ is the integers spaced by $\Delta k = 2\pi/L$, $\hat{Q}$ is the subgroup of multiples of $\Delta r = 2\pi/\Delta y = N\Delta k$, and $\hat{H}$ is the group of order N consisting of multiples of Δk mod N. In this example integrals over $\hat{G}$ and $\hat{H}$ are replaced by sums.

We begin with f defined on G. Its Fourier transform is

$$\int_G e^{-ikx} f(x)\, dx = \hat{f}(k) \tag{6.26}$$

defined for k in $\hat{G}$. The inversion formula then gives

$$f(x) = \int_{\hat{G}} e^{ikx} \hat{f}(k)\, \frac{dk}{2\pi}. \tag{6.27}$$

We now restrict the inversion formula to the discretely sampled points

y in the subgroup H and obtain

$$f(y) = \int_{\hat{G}} e^{iky} \hat{f}(k) \frac{dk}{2\pi} = \int_{\hat{H}} \sum_{r \in \hat{Q}} e^{i(k+r)y} \hat{f}(k+r) \frac{dk}{2\pi}. \tag{6.28}$$

We observe that there is an aliasing effect. The act of discrete sampling implies that all frequencies $k + r$ that differ by a multiple of Δr from k show up under the alias of k. The reason for this is simply that

$$e^{i(k+r)y} = e^{ikr} e^{iry} = e^{ikr} \tag{6.29}$$

when y is in H (one of the discretely sampled points). This is because ry is a multiple of $\Delta r \Delta y = 2\pi$.

Thus we obtain that for y in H

$$f(y) = \int_{\hat{H}} e^{iky} \sum_{r \in \hat{Q}} \hat{f}(k+r) \frac{dk}{2\pi}. \tag{6.30}$$

This is of the form of an inversion formula for the group H. Therefore we have identified the Fourier transform of f restricted to H. The proves the following fundamental result.

The *Poisson summation formula* says that the restriction of f to H has Fourier transform

$$\sum_{y \in H} e^{-iky} f(y)\, \Delta y = \sum_{r \in \hat{Q}} \hat{f}(k+r). \tag{6.31}$$

Here H consists of multiples of Δy and $\hat{Q}$ consists of multiples of $2\pi/\Delta y$.

This formula says that replacing an integral by a Riemann sum has the effect of replacing the Fourier transform by a sum of the transforms over aliased frequencies.

Thus, for instance, when we want to take the Fourier transform of a function on the circle of length L, and we approximate the transform by a Riemann sum with length Δy, then the aliased frequencies r are spaced by $2\pi/\Delta y$. Thus we want to take the Δy sufficiently small so that the $\hat{f}(k+r)$ are close to zero except when $r = 0$.

An immediate consequence is the following somewhat more general *shifted Poisson summation formula.*

$$\sum_{y \in H} e^{-ik(x+y)} f(x+y) \Delta y = \sum_{r \in \hat{Q}} e^{irx} \hat{f}(k+r). \tag{6.32}$$

This is obtained by applying the Poisson summation formula to $g(z) = f(x+z)$ and noting that $\hat{g}(k) = e^{ikx} \hat{f}(k)$.

An important special case of the Poisson summation formula is obtained if we take $k = 0$:

$$\sum_{y \in H} f(y)\Delta y = \sum_{r \in \hat{Q}} \hat{f}(r). \tag{6.33}$$

Even this form leads to remarkable identities.

6.8 The sampling theorem

We may ask to what extent f restricted to the sampling points in H determines f on the other points. The Fourier transform of f restricted to H is $\sum_r f(k+r)$ restricted to the frequency band $\hat{H}$. (Often $\hat{H}$ is thought of as the frequency band running from $-\Delta r/2$ to $\Delta r/2$; however any band of length Δr would do.)

We can try to define a function $f_H(x)$ on the entire group G from this Fourier transform by the formula

$$f_H(x) = \int_{\hat{H}} e^{ikx} \sum_{r \in \hat{Q}} \hat{f}(k+r) \, \frac{dk}{2\pi}. \tag{6.34}$$

We can change variable and write

$$f_H(x) = \sum_{r \in \hat{Q}} e^{-irx} \int_{\hat{H}+r} e^{iux} \hat{f}(u) \, \frac{du}{2\pi}. \tag{6.35}$$

Thus $f_H(x)$ has contributions from all frequency bands, but with a confusing exponential factor in front.

However note that when y is in H, then $f_H(y) = f(y)$. Thus f_H interpolates f at the sampling points.

Another way of writing $f_H(x)$ is as

$$f_H(x) = \int_{\hat{H}} e^{ikx} \sum_{y \in H} e^{-iky} f(y) \, \Delta y \, \frac{dk}{2\pi} = \sum_{y \in H} K_H(x-y) f(y) \Delta y \tag{6.36}$$

where

$$K_H(x) = \int_{\hat{H}} e^{ikx} \, \frac{dk}{2\pi}. \tag{6.37}$$

This formula expresses $f_H(x)$ directly in terms of the values $f(y)$ at the sampling points y.

Now assume in addition that the original Fourier transform $\hat{f}$ is band-limited, that is, it vanishes outside of $\hat{H}$. In that case it is easy to see that $f_H(x) = f(x)$ for all x in G. This is the sampling theorem: A band-limited function is so smooth that it is determined by its values on the sampling points.

6.9 FFT

It is clear that the obvious implementation of the Fourier transform on the cyclic group of order N amounts to multiplying a matrix times a vector and hence has order N^2 operations. The Fast Fourier Transform is another way of doing the computation that only requires order $N \log_2 N$ operations.

Again the setup is a group G and a subgroup H. Again we take Q to be the quotient group. We have

$$\hat{f}(k) = \int_Q e^{-ikx} [\sum_{y \in H} e^{-iky} f(x+y) \Delta y] \, \frac{dx}{\Delta y}. \tag{6.38}$$

The Fast Fourier Transform is a special case. We take G to be a discrete group given by multiples of Δx, and we take H to be the subgroup of even elements. Then Q is a two element group consisting of 0 and Δx. The formula becomes

$$\hat{f}(k) = \frac{1}{2} [\sum_{y \in H} e^{-iky} f(y) \, \Delta y + e^{-ik\Delta x} \sum_{y \in H} e^{-iky} f(y + \Delta x) \, \Delta y], \tag{6.39}$$

where the sum is over y which are multiples of $\Delta y = 2\Delta x$. Here k is a multiple of $\Delta k = 2\pi/(N\Delta x)$.

Again we may write this more explicitly as

$$\hat{f}(m\Delta k) = [\sum_{n=0}^{N/2-1} e^{-i2\pi mn/(N/2)} f(2n\Delta x) + e^{-i2\pi m/N} \sum_{n=0}^{N/2-1} e^{-i2\pi mn/(N/2)} f((2n+1)\Delta x)] \, \Delta x \tag{6.40}$$

If the order of G is N, an even number, then this expresses the Fourier transform on G as the sum of two Fourier transforms on H, a group of order $N/2$. This allows a recursive computation of the Fourier transform.

The number of operations required is of order $C_N = N \log_2 N$. One can see this as follows. For a group of order 1, no computation is required, so $C_1 = 0$. For a group of order N, one must have already computed two transforms of order $N/2$, which took $C_{N/2}$ operations. Then one has to compute the N values, so $C_N = 2C_{N/2} + N$. This determines C_N.

A typical application of the FFT is to compute a convolution. A convolution of two functions on a group of order N is a straightforward order N^2 operation. However the indirect computation by Fourier transforms is much more rapid. The Fourier transform of each function is of order $N \log N$. The multiplication of the Fourier transforms is order N. The inverse Fourier transformation of the product is order $N \log N$. So the whole computation is order $N \log N$, which is much faster.

æ