Keith On ...
Numerical Analysis

K.E. Schubert

Founder
Renaissance Research Labs

Associate Professor
Department of Electrical and Computer Engineering
School of Engineering and Computer Science
Baylor University

Contents

List of Figures

List of Tables

Part I

Classical Numerical Techniques

Chapter 1

Preliminaries

1.1 Taylor Polynomials

We want an easier way of calculating a difficult function, $f(x)$. To this end we want to find a function that is similar to our original that we can calculate. Taylor polynomials, $p_n(x)$[1], are one such type of functions with an easy calculation and intuition. To find the Taylor polynomials we match the derivatives of the two polynomials at a particular point. We are in essence enforcing a smoothness criterion at the point of interest, say $x = a$.

$$\begin{aligned} p_n'(a) &= f'(a) \\ p_n''(a) &= f''(a) \\ &\vdots \\ p_n^{(n)}(a) &= f^{(n)}(a) \end{aligned}$$

Thus the general expression for the Taylor series is

$$\begin{aligned} p_n(x) &= f(a) + (x-a)f'(a) + \frac{(x-a)^2}{2!}f''(a) + \ldots + \frac{(x-a)^n}{n!}f^{(n)}(a) \\ &= \sum_{k=0}^{n} \frac{(x-a)^k}{k!}f^{(k)}(a) \end{aligned}$$

Example

[1] The subscript n tells the highest power of the polynomial, i.e. x^n.

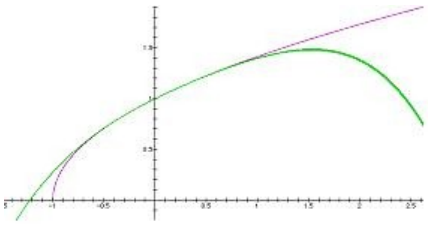


Figure 1.1: Taylor polynomial and $\sqrt{1+x}$

Problem 1.1-3(c)

$$\begin{aligned} f(x) &= \sqrt{1+x} \\ f(0) &= \sqrt{1+0} = 1 \\ f'(0) &= \frac{1}{2\sqrt{1+0}} = \frac{1}{2} \\ f''(0) &= \frac{-1}{4(\sqrt{1+0})^3} = \frac{-1}{4} \\ f''(0) &= \frac{3}{8(\sqrt{1+0})^5} = \frac{3}{8} \\ f^{(k)}(0) &= \frac{(-1)^{k-1}(2k-3)}{2^k} \end{aligned}$$

Example

Problem 1.1-8

$$\begin{aligned} f(x) &= \frac{\log(1+x)}{x} \\ &\approx \frac{\sum_{k=1}^{n} \frac{(-1)^{k-1}}{k} x^k}{x} \\ &= \sum_{k=1}^{n} \frac{(-1)^{k-1}}{k} x^{k-1} \\ f(0) &\approx 1 \end{aligned}$$

1.2 Remainder

The Taylor Series obviously has errors in its approximation. If the original function is in C_{n+1} on the interval $\alpha \leq x \leq \beta$ (with a in the interval) then the remainder (or error) is given by

$$\begin{aligned} R_n(x) &= f(x) - p_n(x) \\ &= \frac{(x-a)^{n+1}}{(n+1)!} f^{(n+1)}(c_x) \end{aligned}$$

with c_x between a and x. To get an error bound we assume that c_x is the worst possible.

Example

Problem 1.2-3(a) In this case $n = 1$ so the worst case would be if $\cos(c_x) = -1$ were -1.

$$R_n(x) = \frac{x^{2(1)+1}}{(2(1)+1)!} = \frac{x^3}{6} \leq \frac{\pi^3}{324} < 0.081$$

Example

Prove problem 8.

1.3 Evaluating Polynomials

Consider the polynomial

$$y = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$$

1.3.1 Straightforward

The obvious way is to calculate each term separately,

$$a_k x^k = a_k * x * x * ... * x$$

This takes k multiplications for a monomial of size k, so for a polynomial with monomials up to size n it would take $\frac{n(n+1)}{2}$ multiplications.

1.3.2 Storing

Calculate $x2 = x * x$, $x3 = x * x2$, etc. This takes $2n - 1$ multiplications. This is much better that the straightforward way. We can even do this without having to store each intermediate result by using a temporary variable. Still it is not the best.

1.3.3 Nesting

Rewrite the polynomial as

$$\begin{aligned} y &= a_n x^n + a_{n-1}x^{n-1} + \ldots + a_1 x + a_0 \\ &= ((\ldots(((a_n)x + a_{n-1})x + a_{n-2})\ldots)x + a_1)x + a_0. \end{aligned}$$

This can be done as

$$\begin{aligned} b_n &= a_n \\ b_{n-1} &= b_n x + a_{n-1} \\ b_{n-2} &= b_{n-1}x + a_{n-2} \\ &\vdots \\ b_1 &= b_2 x + a_1 \\ b_0 &= b_1 x + a_0 \end{aligned}$$

Each step takes 1 multiply so this method takes only n multiplications. The real savings come when you have to calculate a large polynomial many times. Another interesting thing is this is a more accurate method, see Fig 1.2, where the dark blue line is the nested multiplication method. Note on the right side both methods are equally good (or bad in my opinion), but on the right the nested multiplication technique shows marked improvement in accuracy. Not bad for less work.

1.4 Binary

In any number system, the position of a digit relative to the decimal place specifies the integer power of the base we must multiple the digit by to get its value. We specify what base we are using by a subscript, if no subscript appears then the base is obvious(usually base 10, though sometimes it will be base 2 if we are calculating in base 2 for that section). So for base 10

$$101_{10} = 1 \times 10^2 + 0 \times 10^1 + 1 \times 10^0,$$

and for base 2

$$101_2 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 5_{10}.$$

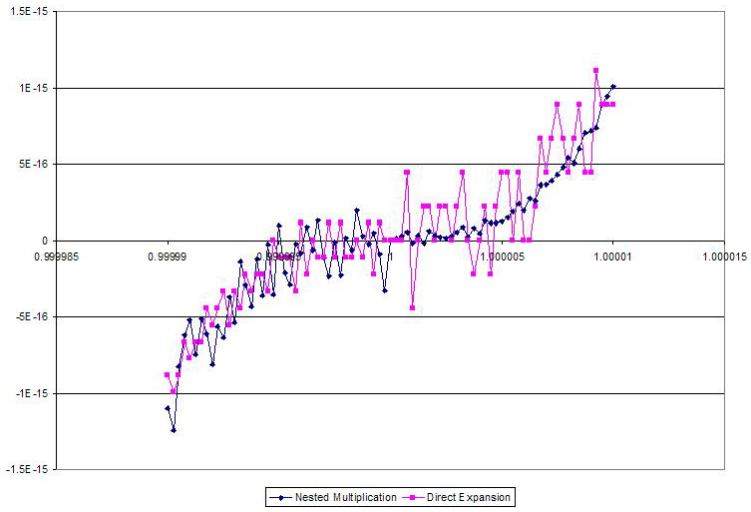


Figure 1.2: Close-up Look at Resulting Values of Two Evaluation Methods for $y = x^3 - 3x^2 + 3x - 1$

This gives us one way to convert numbers. For instance, we can convert binary to decimal by expanding the binary number in this way. Thus using the above to convert binary (10.01) to decimal we find,

Note that the "2" we are using is the base of binary in decimal form, and this is why we went from binary to decimal. In binary, its form would be "10" and ten would be "1010". Therefore, we could go to binary by, expanding this out with ten in binary. The problem with this method is it is clumsy to use since we do not do squaring, cubing, etc. easily in base 2. Another problem is that 0.1 is an infinitely repeating decimal in binary so it is a pain to deal with 10-1! Instead, we convert decimal to binary as follows.

1. Split your number into a.b
2. For the whole number part, a
   (a) Divide 2 into a and note the quotient and remainder as q1,r1 (a=2*q1+r1)
   (b) As long as the quotient from above is not zero, divide it by 2 and record the quotient and remainder as qi,ri (with i denoting the current step). Repeat.
   (c) The binary equivalent of a is rnrn-1...r2r1. Basically we have done our nested polynomial evaluation backwards with x=2, and the coefficients being the remainders.
3. For the fractional part (b)
   (a) Multiply 2*b, and record the unit value as a1. Denote b-a1=b1.

Table 1.1: Binary, Decimal, and Hexadecimal Equivalents

Bin	Hex	Dec	Bin	Hex	Dec
0000	0	0	1000	8	8
0001	1	1	1001	9	9
0010	2	2	1010	A	10
0011	3	3	1011	B	11
0100	4	4	1100	C	12
0101	5	5	1101	D	13
0110	6	6	1110	E	14
0111	7	7	1111	F	15

(b) If bi does not equal zero, multiply it by 2, denoting the units digit by ai+1 and the difference bi-ai+1=bi+1. Repeat until the difference is zero (this may never happen so be looking for patterns to get repeating fractions).

(c) The fractional part, b, is a1a2a3a4...

4. The full answer is thus rnrn-1...r2r1.a1a2a3a4...

1.5 Hexadecimal

This is often made to sound more intimidating than it is. Hexadecimal numbers are simply base 16, but this can be handled nicely since $2^4 = 16$. All you have to do is group binary digits into groups of 4 and use the conversion table

1.6 Fixed Point Numbers

Example:

Convert π to binary and hexadecimal. Assume you have four bits before the radix point and 8 bits after the radix point.

Sol:

before the decimal we have $3 = 0011$

after the decimal

$0.1415926\ldots$	
0.2831852	0
0.5663704	0
1.1327408	1
0.2654816	0
0.5309632	0
1.0619264	1
0.1238528	0
0.2477056	0

combining gives 0011.00100100

To convert to hexadecimal we group the digits together in groups of four starting at the radix point, thus we are forcing the hexadecimal digits to represent either integer or fractional portions.

0011	0010	0100
3	2	4

Thus the answer is $0x3.24$.

Example:

Convert 25.6875 to binary.

25	/2	.	*2	.6875
12	1		1	.375
6	0		0	.75
3	0		1	.5
1	1		1	0
0	1			

11001.1011

1.7 Floating point numbers

While the book discusses single precision numbers, they are essentially never used, as double precision is so much better and readily available. We will assume IEEE double precision floating point representation, as it is the standard. IEEE floating point numbers have the form

$$(-1)^s 2^E (b_0.b_{-1}b_{-2}\ldots b_{1-p}),$$

where

$$\begin{aligned} s &\in \{0,1\} \\ s &\in \{0,1\} \\ b_0 &= 1\,(implicitly) \\ b_{-j} &\in \{0,1\} \forall j \in \{1,2,\ldots p-1\} \end{aligned}$$

	Precision	
	Single	Double
P	24	53
E_{min}	-126	-1022
E_{max}	127	1023
$Bias$	127	1023

Thus IEEE is represented in memory as a sign bit, exponent bits (8 or 11), and mantissa bits (23 or 52). The mantissa is composed of all the b_{-j}. A few things to note about IEEE arithmetic.

1. The exponent stored is $E = e - Bias$
2. ± 0 is encoded by $E_{min} - 1$ and $f = 0$
3. Denormalized numbers are encoded by $E_{min} - 1$ and $f \neq 0$
4. $\pm\infty$ is encoded by $E_{max} + 1$ and $f = 0$
5. NAN is encoded by $E_{max} + 1$ and $f \neq 0$

1.7.1 Approximating the Reals

To approximate the real number x, we define the function fl(x) as, 0 when x=0, and the nearest element in floating point to x otherwise. Finding nearest elements requires a rounding scheme (rounding or "chopping"/truncating) and a tie breaker procedure (usually round away from zero).

1.7.2 Bounding Errors

To bound the error in approximating the real number x, we need to consider the floating point number, $fl(x)$, used to approximate x. First we note that a real number x, is written in binary as

$$x = \sigma \cdot f_r \cdot 2^e,$$

where σ is the sign, f_r has as many digits as needed, and e is any integer. Note that e will be different for IEEE, which normalizes to $1 \leq f_r < 2$ with an implicit 1 at the start; than the non-standard forms, which normalize to $0.5 \leq f_r < 1$ with no assumed leading 1. We will assume that e is within the permitted bounds for simplicity. The floating-point representation is

$$fl(x) = \sigma \cdot f \cdot 2^e.$$

We can now write the difference as

$$x - fl(x) = \sigma \cdot (f_r - f) \cdot 2^e.$$

For the moment, we will consider the difference $f_r - f$. Note that we are dealing with normalized numbers with n bits of accuracy and an implicit leading 1 (IEEE arithmetic), while the book deals with numbers normalized between a half and one, with no implicit 1, so for us

$$\begin{aligned} f_r &= 1.\beta_{-1}\beta_{-2}\beta_{-3}\ldots\beta_{1-n}\beta_n\ldots \\ f_r &= 1.\bar{\beta}_{-1}\bar{\beta}_{-2}\bar{\beta}_{-3}\ldots\bar{\beta}_{1-n}. \end{aligned}$$

Note that the digit to the left of the decimal in f is assumed to be 1, the only exception is when $f_r = 1.1111...$ which would have $f = 10.000...0$[2]. Technically it would actually have

[2]We are forcing the exponent to be the same, otherwise it would always be true for non-zero numbers

f=1.000...0 and the exponent would be (e+1) but since we are keeping the exponent e we keep the simplification. Note that this is equivalent to rounding 9.5 to 10. Anyway, our real concern is the worst case of the difference, which is in all cases given by

$$f_r - f = \pm 0.000 \ldots 01.$$

Note that the 1 is in the n^th place after the decimal[3]. We rewrite this using floating point notation as . We now stick this back into the expression for the difference between x and fl(x) and obtain an upper bound by taking absolute value . Similarly to get a lower bound we take the negative of the absolute value, and find

Now we note that the size of x is . For the book's form of the mantissa, we would have . The relative error is thus .

1.8 Floating Point Dangers

I came up with the following program in my doctoral work at UCSB.

```
#include <iostream>
#include <iomanip>
#include <cmath>

using namespace std;

int main(){
    double pi, e, result;
    int i;

    e=exp(1);

    pi=atan(1)*4;

    result=pi;

    for(i=1;i<53;i++){
        result=sqrt(result);
    }

    for(i=1;i<53;i++){
        result=result*result;
    }

    cout << setiosflags(ios::showpoint | ios::fixed) << setprecision(16);
```

[3]Explain

```
    cout << "Pi     = " << pi << endl;
    cout << "Result = " << result << endl;
    cout << "e      = " << e << endl;

    return 0;
}
```

The results are

```
Pi     = 3.1415926535897931
Result = 2.7182818081824731
e      = 2.7182818284590451
Press any key to continue
```

Notice that Result is e to 7 significant digits, but it should be π. This underscores the importance of being numerically aware when writing programs.

1.9 IEEE 754

Floating point numbers are based off scientific notation. Consider a typical number in base 10 scientific notation,

$$-1.23 \times 10^3.$$

The number is composed of five pieces of information,

1. sign of the number (-),
2. significant or mantissa (1.23),
3. base (10),
4. sign of the exponent (+),
5. magnitude of the exponent (3).

There are two basic number formats called out in IEEE 754, single precision (float in c/c++), and double precision (double in c/c++). In addition there are two extended formats, which are only used as intermediate results while calculating.

For this discussion, the notation $fl(x)$ will be used to mean the number x as it is represented in floating point on a computer.

$$(-1)^s \cdot 1.f \times 2^{e-127}$$

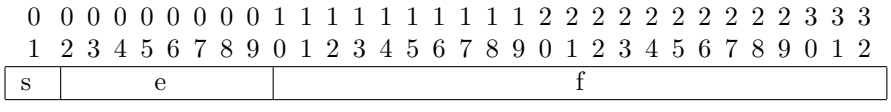


Table 1.2: IEEE Single Precision Floating Point

e	f	Category	Interpretation
$1\ldots11$	$1\ldots11$ $\vdots$ $0\ldots01$	NaN	See Codes
$1\ldots11$	$0\ldots00$	$\pm\infty$	$\pm\infty$
$1\ldots10$ $\vdots$ $0\ldots01$	$1\ldots11$ $\vdots$ $0\ldots00$	Numbers	$(-1)^s \times 1.f \times 2^{(e-127)}$
$0\ldots00$	$1\ldots11$ $\vdots$ $0\ldots00$	Denormals	$(-1)^s \times 0.f \times 2^{(-126)}$
$0\ldots00$	$0\ldots00$	±0	±0

Table 1.3: IEEE Floating Point NaN Codes

Dec	Meaning	Example
1	invalid square root	$\sqrt{-1}$
2	invalid addition	$\infty + -\infty$
4	invalid division	$\frac{0}{0}$
8	invalid multiplication	$0 \times \infty$
9	invalid modulo	$xmod0$

This is equivalent to saying

$$(-1)^s \cdot 1.f \times 2^E$$

0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2	2	3	3	3
1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2
s	e=E+127																	f													

They are the same because $e - 127 = E$ is the same equation as $e = E + 127$. I think the latter is easier to use because you read E from the number and want e. The first form (standard for most texts) involves you guessing what number produced what you are seeing (rather than calculating it). It is like trying to solve $y = mx + b$ for y given x but using the form $\frac{(y-b)}{m} = x$ to do it. It works, just not well. In any case, consider some examples.

Example:

Convert 7.892 to single precision IEEE.

Step 1: Convert 7.892 to binary

$7.892 = 111.11100100001011010000111$

Step 2: Normalize and note sign

$7.892 = (-1)^0 1.11111001000101101000011 1 \times 2^2$

Step 3: Calculate Excess 127 code for exponent

$e = 2 + 127 = 129 = 10000001$

Step 4:Round $1.f$ to 24 digits

$fl(1.11111001000101101000011 1) = 1.11111001000101101000100$

Step 5: Assemble

0	1 0 0 0 0 0 0 1	1 1 1 1 1 0 0 1 0 0 0 1 0 1 1 0 1 0 0 0 1 0 0

Example:

Calculate 3.75×29.625 in IEEE-754 single precision floating point.

Convert:

$3.75 = 11.11 = 1.111 \times 2^1$

$29.625 = 11101.101 = 1.1101101 \times 2^4$

Multiply Significants:

	1.	1	1	0	1	1	0	1			
$\times$	1.	1	1	1							
	1.	1	1	0	1	1	0	1			
	0.	1	1	1	0	1	1	0	1		
	0.	0	1	1	1	0	1	1	0	1	
	0.	0	0	1	1	1	0	1	1	0	1
1	1.	0	1	1	1	1	0	0	0	1	1

1.10111100011×2^1

Add exponents to normalization exponent and put in excess 127:

$1 + 4 + 1 + 127 = 133 = 10000101$

Write in single precision:

0	10000101	1011 1100 0110 0000 0000 000

Example:
Perform the following for IEEE-754, single precision

1. Show the representation of $x = 93.3125$

   $x = 93.125_{10} = 1011101.001_2 = 1.011101001 \times 2^6$

0	1 0 0 0 0 0 1 0 1	0 1 1 1 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0

2. calculate $x * y$ for y equal to

0	1 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0

   exponent: 128+133-127=134

   float: shortcut, note that y only has two 1's in the expansion (hidden and near end) and they are farther apart than the length of the significant portion of x. This will cause the x float to be placed starting at these locations. The comma below notes where the last bit of precision lies.

   $$z_{fl} = 1.01110100100000000000101, 1101001$$

   Note that the first bit after the comma is a 1 so the number gets rounded up.

   z is

0	1 0 0 0 0 1 1 0	0 1 1 1 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 1 0

Example:
Convert 3.03125 to IEEE single precision

3	.	03125
1	1 0	0625
0	1 0	125
	0	25
	0	5
	1	0

$3.03125_{10} = 11.00001_2 = 1.100001_2 \times 2^1$

$1 + 127 = 128$

0	1 0 0 0 0 0 0 0	1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Now perform the following on your result and

0	1 0 0 0 0 1 0 0	0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0

1. Addition

   $x = 1.0000000100000001_2 \times 2^5$

   $y = 1.100001_2 \times 2^1 = 0.0001100001_2 \times 2^5$

$$\begin{aligned} x+y &= 1.0000000100000001_2 \times 2^5 + 0.0001100001_2 \times 2^5 \\ &= (1.0000000100000001_2 + 0.0001100001_2) \times 2^5 \\ &= (1.0001100101000001_2) \times 2^5 \end{aligned}$$

0	1 0 0 0 0 1 0 0	0 0 0 1 1 0 0 1 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0

2. Multiplication

   exponent is $132 + 128 - 127 = 133$

   significant is $1.0000000100000001 \times 1.100001 = 1.10000101100000101100001$

0	1 0 0 0 0 1 0 1	1 0 0 0 0 1 0 1 1 0 0 0 0 1 0 1 1 0 0 0 0 1 0

Example:
Perform the following for IEEE-754, single precision

1. Show the representation of $x = 0.8125$

0	0 1 1 1 1 1 1 0	1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

2. calculate (show steps) $x * y$ for x from above and

   y is

1	1 0 0 0 0 0 0 1	1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

   Exponent: $(10000001 + 01111110) - 01111111 = 11111111 - 01111111 = 1000000$

   float= $1.101 * 1.11 = 10.11011 = 1.011011 \times 2^1$, so add 1 to exponent

1	1 0 0 0 0 0 0 1	0 1 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

3. Perform the multiplication above in decimal and verify the answer.

   $.8125 * (-7) = -5.6875 = -101.1011_2$

1.10 Rounding versus Chopping

Rounding is almost always used because of two reasons. To see both, let the interval between two numbers in the representation is 2δ then for rounding $x - fl(x) \in [-\delta, \delta)$, while for chopping it is $x - fl(x) \in [0, 2\delta)$. The first problem is that the error magnitude is up to twice as large for chopping. This is obviously bad, but it is not as bad as the second problem. The second problem is that all the errors of chopping have the same sign, so no error cancellation is possible when calculations are done. To see why this is bad, consider the following.

Example:

Find out the error in calculating $\sum_{i=1}^{n} x_i$ on a computer. First note that what you actually calculate is $\sum_{i=1}^{n} fl(x_i)$. The error (actual minus calculated) is thus $Err = |(\sum_{i=1}^{n} x_i) - (\sum_{i=1}^{n} fl(x_i))|$. Also let $fl(x_i) = x_i + \gamma_i$ for γ_i in the error interval of your method.

$$\begin{aligned} Err &= \left| (\sum_{i=1}^{n} x_i) - (\sum_{i=1}^{n} (x_i + \gamma_i)) \right| \\ &= \left| (\sum_{i=1}^{n} x_i) - (\sum_{i=1}^{n} x_i + \sum_{i=1}^{n} \gamma_i) \right| \\ &= \left| \sum_{i=1}^{n} x_i - \sum_{i=1}^{n} x_i - \sum_{i=1}^{n} \gamma_i \right| \\ &= \left| \sum_{i=1}^{n} \gamma_i \right| \\ &\leq \sum_{i=1}^{n} |\gamma_i| \end{aligned}$$

For chopping the last inequality is actually an equality, i.e. chopping always has the worst case error. For a typical case on rounding the errors are distributed with some positive and some negative, thus cancelation can occur. For large sums (many terms) the law of large numbers and an assumed uniform distribution of γ_i indicates that the error for rounding will go to 0! This is a great result.

1.11 Absolute and Relative Error

The two most basic types of errors are absolute and relative. The absolute error of an estimate $\hat{x}$ of a number x, is the difference between them.

$$AE = \Delta x \tag{1.1}$$
$$= x - \hat{x} \tag{1.2}$$

The relative error of an estimate $\hat{x}$ of a number x, is the difference between them divided by x.

$$RE = \frac{\Delta x}{x} \tag{1.3}$$
$$= \frac{x - \hat{x}}{x} \tag{1.4}$$

Often the the absolute value of either the absolute or relative error is taken, though not in every case, so it is important to know if only the magnitude or magnitude and phase (sign) is required. Unfortunately, there is no standard nomenclature or convention, though more

often than not the magnitude (take the absolute value) is desired. Usually the phase/sign information is only retained in the error if it is plausible to have and will be useful in the algorithm or analysis.

1.12 Propagation of Error

We have seen that representing the real numbers on a computer involves errors. When we use floating point numbers in a calculation rather than the actual numbers the errors can grow. The errors caused by using floating point approximations are called propagated errors. Two ways of bounding propagation errors exist. The forward method involves explicitly calculating the errors and is called interval arithmetic. The backward method involves finding a condition number, which gives a bound on how big the error can grow.

1.13 Interval Arithmetic

Let's consider the error in a computation between the true values (xT, yT) and the approximate values (xA, yA). We only know the approximate values and the error bounds

Note that the error is could be positive or negative so we must consider the positive and negative bounds. First, we will look at the error for addition or subtraction.

Now let's consider multiplication.

It is easy to see that this can quickly become very hard to deal with. Consider for instance multiplying two n-by-n matrices, which would involve n^3 multiplies. Keeping track of all of them would rapidly become impossible. We will consider one final operation, namely division.

Again we can see that things can become very complicated quickly.

1.14 Forward and Backward Error

Two other distinctions in error types that come up in stability of algorithms are forward and backward errors. They are concerned with a piece of data, x, that is acted on by an equation or algorithm, $f(\cdot)$ to produce a result, y.

$$y = f(x) \tag{1.5}$$

This probably seems pretty straight-forward, but what happens when I can't actually calculate $f(\cdot)$? For instance, how can I calculate a transcendental function[4] like $\sin(x)$? The answer is we have to approximate the equation or algorithm with another one that is calculable on a computer. We will examine some ways to do this later, but first we need a way to see how good our approximation is, which is what forward and backward errors gives

[4]Transcendental functions cannot be expressed as a finite sequence of algebraic operations - arithmetic and roots

us. We will denote our approximate algorithm by $\hat{f}(\cdot)$. Now if we give our approximate algorithm the true starting value of x, what it calculates will not be y but $\hat{y}$.

$$\hat{y} = \hat{f}(x) \tag{1.6}$$

The error of the results, given the same starting point, gives us an idea of how good our approximation is. If it is small then we say the algorithm is stable. The difference of the values calculated is the forward error.

$$\begin{aligned} FE_{abs} &= \Delta y \\ &= y - \hat{y} \\ &= f(x) - \hat{(}x) \\ FE_{rel} &= \frac{\Delta y}{y} \\ &= \frac{y - \hat{y}}{y} \\ &= \frac{f(x) - \hat{f}(x)}{f(x)} \end{aligned}$$

In general we will use the relative forward error, though you could use either. Forward error is a useful and intuitive concept, but it is not always easy to quantify for a range of values, since you have to explicitly calculate for each one. This makes it less useful for algorithm analysis. Enter the backward error. Its definition will seem weird and probably even more difficult, but it turns out to be easy in many cases to bound for a wide range of problems. In other words we use it because it makes our lives easy.

The backward error is the smallest value, Δx such that

$$f(\hat{x}) = \hat{f}(x) \tag{1.7}$$

$$\hat{x} = x + \Delta x \tag{1.8}$$

holds. Solving this we find

$$\begin{aligned} f(x + \Delta x) &= \hat{f}(x) \\ x + \Delta x &= f^{-1}\left(\hat{f}(x)\right) \\ \Delta x &= f^{-1}\left(\hat{f}(x)\right) - x. \end{aligned}$$

The absolute backward error is Δx and the relative backward error is $\frac{\Delta x}{x}$.

1.15 Condition Number

To make this a little more understandable, I am going to draw a distinction not used in the literature[5], that of condition number at a point and condition number of an algorithm.

[5]This warning exists only to prevent you from expecting others to know or use this, though it is much clearer in my mind.

The condition number of an algorithm is the worst condition number at a point for all the possible points, thus

$$cond(f) = \sup_x cond(f(x)) \tag{1.9}$$

The condition number at a point is then how much the output changes given a change in the input, thus it is a ratio of the relative forward error over the relative backward error.

$$\begin{aligned} cond(f(x)) &= \left| \frac{\frac{y-\hat{y}}{y}}{\frac{x-\hat{x}}{x}} \right| \\ &= \left| \frac{\frac{f(x)-\hat{f}(x)}{f(x)}}{\frac{x-f^{-1}(\hat{f}(x))}{x}} \right| \end{aligned}$$

Note that by definition of the backward error, we have $f(\hat{x}) = \hat{f}(x)$ and $\Delta x = x - \hat{x}$.

$$cond(f(x)) = \left| \frac{\frac{f(x)-f(\hat{x})}{f(x)}}{\frac{\Delta x}{x}} \right|$$

In this form it looks a lot like a calculus formula, which suggests using the derivative to simplify the equation. This of course requires the derivative to exist. Recall that

$$f'(x) = lim_{\Delta x \to 0} \frac{f(x) - f(x + \Delta x)}{\Delta x}.$$

We don't have a limit, so we can only approximate this

$$f'(x) \approx \frac{f(x) - f(x + \Delta x)}{\Delta x}$$

thus

$$\begin{aligned} cond(f(x)) &= \left| \frac{\frac{f(x)-f(\hat{x})}{f(x)}}{\frac{\Delta x}{x}} \right| \\ &= \left| \frac{f(x) - f(\hat{x})}{\Delta x} \frac{x}{f(x)} \right| \\ &= \left| \frac{f(x) - f(x + \Delta x)}{\Delta x} \frac{x}{f(x)} \right| \\ &\approx \left| f'(x) \frac{x}{f(x)} \right| \end{aligned}$$

The condition number at a point thus gives us how the errors grow at that point, and the condition number of the function gives us the worst case bound on how bad our errors can grow.

Note this is not the only way we could have developed this. We could have required our function to be continuous on $[x,\hat{x}]$ and differentiable on $(x,\hat{x})$, which is looser than the completely differentiable requirement I made above. We can thus use the mean value theorem to see

$$f'(c) = \frac{f(x) - f(\hat{x})}{x - \hat{x}}$$

We now note that since c is between the true, x, and approximate, $\hat{x}$ values, and that the interval is on the order of 10^{-16} for IEEE double-precision arithmetic. We can thus assume c is approximately x.

$$f'(x) \approx \frac{f(x) - f(\hat{x})}{x - \hat{x}}$$

The derivative of $f(x)$ at x, is called the absolute condition number. To get the relative condition number you just have to divide each of the differences by their true value, which works out to multiplying by $\frac{x}{f(x)}$, thus

$$cond(f(x)) \approx \left| f'(x) \frac{x}{f(x)} \right|$$

Same equation either way.

The condition number shows how the error of the approximation will influence the error of the calculation. The condition number is nice in that it cleanly handles the error bounds. It is not as precise as the error in the interval arithmetic, but it is tractable even for large matrix operations, which will involve the norms of the matrices rather than the elements. Quite a savings!

1.15.1 Sums

We have spoken a lot about summation, but we want to look at one final area of sums before we move on. Consider the following summation:

$$100000 + 45 + 45 + 45 + 45 \tag{1.10}$$

In real numbers it doesn't matter if we add the 45's first or the 100000. In floating point numbers it does matter! Floating point numbers are not associative. To see this consider a 4 decimal place accuracy machine that uses rounding, and is nicely implemented. In this case we see that

$$100000 + 45 = 100000$$

so if we add as stated we find the sum is 100000 for the series (rather than 100180). If we add the 45's first we find that

$$45 + 45 + 45 + 45 = 180,$$

then

$$100000 + 180 = 100200.$$

A much better result. These sums occur in a variety of places, from standard series, to evaluating integrals, to inner products of vector, and matrix multiplication. In short you should be aware of the lack of the associative property.

Example

Write C/C++ code to sum the following $\sum_{i=1}^{100} \frac{1}{i^2}$. Make sure you do it in the right order.

```
double sum=0;
int i;

for(i=100;i>=0;i--){
    sum+=1.0/(i*i);}
```

Kahan's Compensated Summation

Listing 1.1: Kahan's Compensated Summation

```
function Sum=compsum(x)
  [row,col]=size(x);
  n=max(row,col);
  Sum=0;
  carry=0;
  for i=1:row
    for j=1:col
      x_corrected = x(i,j)-carry;
      next_sum = Sum + x_corrected;
      carry = (next_sum - Sum) - x_corrected;
      Sum=next_sum;
    end
  end
endfunction
```

Chapter 2

Zero Finding

Almost every interesting problem in mathematics can be reduced to trying to find the zeros of a function. The next several classes will be spent examining how we find zeros. In general, you cannot explicitly solve for the zeros so you need to make iterative procedures to find them. Today we will look at two methods: bisection and Newton's method.

2.1 Bisection

Bisection is a nice method in that it is guaranteed to converge and you can state exactly how many iterations it will take. The idea of the bisection method is to pick an interval where a root is guaranteed to exist. This is done by making the sign of the function different on the two ends of the interval.

For instance consider the function in Figure 2.1. The interval we start with is $[0, 1]$ and it is marked by red lines. We then calculate the midpoint (.5 marked by the light green line) and check which of the new intervals ($[0, .5]$ or $[.5, 1]$) has the zero in it (by checking which interval has a sign change). In this case the next interval is [0,.5]. We then halve the interval again and continue. The next several intervals marked on the plot are $[.25, .5]$, $[.375, .5]$, $[.4375, .5]$, and finally $[.4375, .46875]$.

2.2 Newton's Method

Newton's Method essentially is an algebraic re-writing of the tangent line of a function at a point. Let the function we are trying to find the zero of be denoted by $f(x)$ and the point be x_i. We know that the slope of the tangent line has to be the same as the slope of the graph at the point x_i, thus the tangent line is $y = f'(x_i)x + b$. We can find b by noting the line passes through $(x_i, f(x_i))$, so

$$\begin{aligned} f(x_i) &= f'(x_i)x_i + b \\ b &= f(x_i) - f'(x_i)x_i. \end{aligned}$$

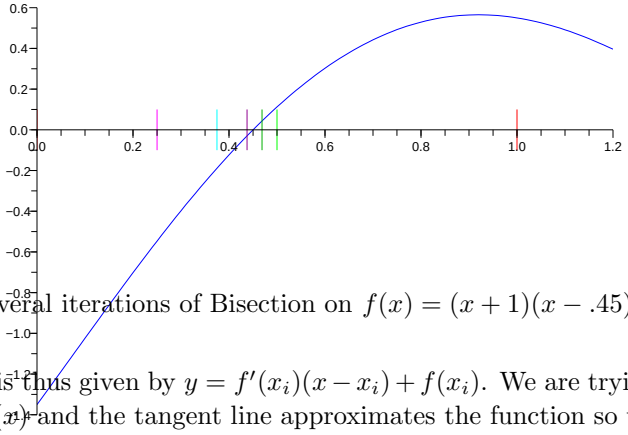


Figure 2.1: Several iterations of Bisection on $f(x) = (x+1)(x-.45)(x-1.5)(x-2)$

The tangent line is thus given by $y = f'(x_i)(x - x_i) + f(x_i)$. We are trying to find the zeros of the function $f(x)$ and the tangent line approximates the function so we want to find the point where the tangent line intercepts the x-axis.

$$\begin{aligned} 0 &= f'(x_i)(x_{i+1} - x_i) + f(x_i) \\ 0 &= x_{i+1} - x_i + \frac{f(x_i)}{f'(x_i)} \\ x_{i+1} &= x_i - \frac{f(x_i)}{f'(x_i)} \end{aligned}$$

Newton's methods is thus the next estimate of the root is $x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$. We can see this graphically in Figure 2.2. Newton's method for $f(x) = x^2$ is thus $x_{i+1} = x_i - \frac{x_i^2}{2x_i}$ or $x_{i+1} = \frac{x_i}{2}$ The first estimate is $x_0 = 2$, the next is $x_1 = 1$, then $x_2 = .5$ and finally $x_3 = .25$. Notice that the limit is zero as desired, and we can get as close as we want by repeating the method until $x_{i+1} - x_i < tol$, where tol is some tolerance we select.

We can then use Taylor's formula to obtain an error bound. Let α be the actual value of the root.

$$\begin{aligned} e_{n+1} &= \alpha - x_{n+1} \\ &= \alpha - \left(x_n - \frac{f(x_n)}{f'(x_n)} \right) \\ &= e_n + \frac{f(\alpha) - e_n f'(\alpha) + \frac{e_n^2}{2} f''(\zeta)}{f'(x_n)} \end{aligned}$$

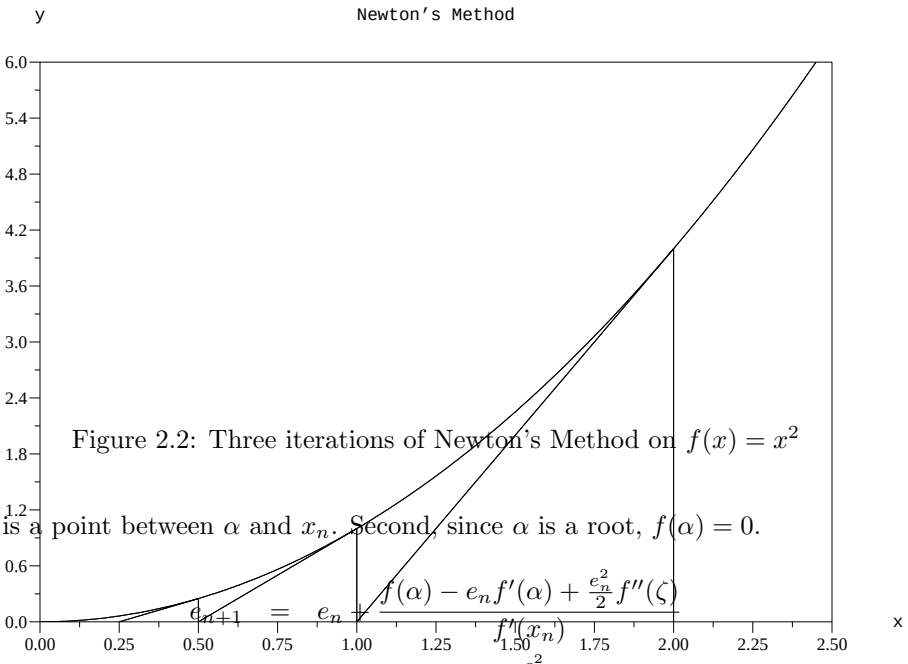


Figure 2.2: Three iterations of Newton's Method on $f(x) = x^2$

First, ζ is a point between α and x_n. Second, since α is a root, $f(\alpha) = 0$.

$$\begin{aligned}
e_{n+1} &= e_n + \frac{f(\alpha) - e_n f'(\alpha) + \frac{e_n^2}{2} f''(\zeta)}{f'(x_n)} \\
&= e_n + \frac{-e_n f'(\alpha) + \frac{e_n^2}{2} f''(\zeta)}{f'(x_n)} \\
&= e_n \left(1 - \frac{f'(\alpha)}{f'(x_n)}\right) + e_n^2 \frac{f''(\zeta)}{2f'(x_n)} \\
&= e_n \frac{f'(x_n) - f'(\alpha)}{f'(x_n)} + e_n^2 \frac{f''(\zeta)}{2f'(x_n)}
\end{aligned}$$

When x_n is close to α then $f'(x_n) - f'(\alpha) \approx 0$, so

$$\begin{aligned}
e_{n+1} &= e_n^2 \frac{f''(\zeta)}{2f'(x_n)} \\
&= e_n^2 C.
\end{aligned}$$

When we are close to the root, the error is dropping off as a square, thus Newton's method has quadratic convergence.

2.3 Secant

Newton's Method requires the knowledge of the first derivative of the function. Often the derivative is very complicated to evaluate and will take a long (relatively anyway) time to do so. In many cases the first derivative may not be available. In some cases it might not even exist at all points in the interval of interest. Even when it is available it could be near zero which would cause numerical problems in evaluating it, even if it is in the region of convergence. For all of these regions a new method was devised, which drew on the material leading up to calculus.

Recall that the tangent line was found as the limit of a series of secant lines. We can say that the derivative can thus be approximated by

$$f(x) \approx \frac{f(x_1) - f(x_2)}{x_1 - x_2}.$$

Thus if we know two points, we can approximate the function by a straight line between them and use the x-intercept as the next point to evaluate. We now need two points instead of one and a derivative. We refer to this as a two-point method because of the need of multiple points. We will need two estimates to begin our evaluation. Given two initial guesses, x_0 and x_1, the slope, m, is given by

$$m = \frac{f(x_1) - f(x_0)}{x_1 - x_0}.$$

Using this we find the next point, x_2 by using the point-slope form of a line

$$\begin{aligned} f(x_2) - f(x_1) &= \frac{f(x_1) - f(x_0)}{x_1 - x_0}(x_2 - x_1) \\ x_2 - x_1 &= \frac{x_1 - x_0}{f(x_1) - f(x_0)}(f(x_2) - f(x_1)) \\ x_2 &= x_1 - f(x_1)\frac{x_1 - x_0}{f(x_1) - f(x_0)}. \end{aligned}$$

We thus have the equation for the next estimate:

$$x_{n+1} = x_n - f(x_n)\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}. \tag{2.1}$$

Note that you can store the previous function evaluation and then you will not need to do two function evaluations per iteration.

Now we want to calculate the error. To do this we will subtract eq 2.1 from $\alpha = \alpha$.

$$\begin{aligned}
e_{n+1} &= \alpha - x_{n+1} \\
&= \alpha - \left(x_n - f(x_n) \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})} \right) \\
&= \alpha - \frac{f(x_n)x_{n-1} - f(x_{n-1})x_n}{f(x_n) - f(x_{n-1})} \\
&= \frac{f(x_n)(\alpha - x_{n-1}) - f(x_{n-1})(\alpha - x_n)}{f(x_n) - f(x_{n-1})} \\
&= \frac{f(x_n)e_{n-1} - f(x_{n-1})e_n}{f(x_n) - f(x_{n-1})} \\
&= e_n e_{n-1} \frac{\frac{f(x_n)}{e_n} - \frac{f(x_{n-1})}{e_{n-1}}}{f(x_n) - f(x_{n-1})} \\
&= e_n e_{n-1} \frac{\frac{f(x_n)}{e_n} - \frac{f(x_{n-1})}{e_{n-1}}}{x_n - x_{n-1}} \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})} \\
&\approx e_n e_{n-1} \frac{\frac{f(x_n)}{e_n} - \frac{f(x_{n-1})}{e_{n-1}}}{x_n - x_{n-1}} \frac{1}{f'(\alpha)}
\end{aligned}$$

We need to evaluate $\frac{f(x_n)}{e_n}$, so we will use Taylor's Theorem for $f(x)$ evaluated at α. We find that

$$\begin{aligned}
\frac{f(x_n)}{e_n} &= \frac{f(\alpha) + (\alpha - x_n)f'(\alpha) + \frac{1}{2}(\alpha - x_n)^2 f''(\alpha) + \mathcal{O}((\alpha - x_n)^3)}{e_n} \\
&= \frac{e_n f'(\alpha) + \frac{1}{2}e_n^2 f''(\alpha) + \mathcal{O}(e_n^3)}{e_n} \\
&= f'(\alpha) + \frac{1}{2}e_n f''(\alpha) + \mathcal{O}(e_n^2)
\end{aligned}$$

Resuming our evaluation of e_{n+1} we find

$$\begin{aligned}
e_{n+1} &\approx e_n e_{n-1} \frac{f'(\alpha) + \frac{1}{2}e_n f''(\alpha) + \mathcal{O}(e_n^2) - f'(\alpha) - \frac{1}{2}e_{n-1}f''(\alpha) + \mathcal{O}(e_{n-1}^2)}{x_n - x_{n-1}} \frac{1}{f'(\alpha)} \\
&= e_n e_{n-1} \frac{\frac{1}{2}e_n f''(\alpha) - \frac{1}{2}e_{n-1}f''(\alpha) + \mathcal{O}(e_{n-1}^2)}{x_n - x_{n-1}} \frac{1}{f'(\alpha)} \\
&= e_n e_{n-1} \frac{\frac{1}{2}(e_n - e_{n-1})f''(\alpha) + \mathcal{O}(e_{n-1}^2)}{x_n - x_{n-1}} \frac{1}{f'(\alpha)} \\
&= e_n e_{n-1} \frac{\frac{1}{2}(x_n - x_{n-1})f''(\alpha) + \mathcal{O}(e_{n-1}^2)}{x_n - x_{n-1}} \frac{1}{f'(\alpha)} \\
&= e_n e_{n-1} (\frac{1}{2}f''(\alpha) + \mathcal{O}(e_{n-1}^2)) \frac{1}{f'(\alpha)} \\
&\approx e_n e_{n-1} \frac{f''(\alpha)}{2f'(\alpha)} \\
&\approx e_n e_{n-1} M.
\end{aligned}$$

This is similar to Newton's method which suggests that

$$e_{n+1} = A e_n^c,$$

which implies

$$\begin{aligned}
e_n &= A e_{n-1}^c \\
A^{-c^{-1}} e_n^{c^{-1}} &= e_{n-1}.
\end{aligned}$$

Substituting and collecting terms we find

$$\begin{aligned}
(A e_n^c) &= (e_n)(A^{-c^{-1}} e_n^{c^{-1}})M \\
A^{1+c^{-1}} M^{-1} &= e_n^{1-c+c^{-1}} \\
B &= e_n^{1-c+c^{-1}}.
\end{aligned}$$

Since the left hand side is a constant the exponent must be zero, because e_n is a variable. This means

$$\begin{aligned}
0 &= 1 - c + c^{-1} \\
&= c^2 - c - 1 \\
c &= \frac{1 \pm \sqrt{1+4}}{2},
\end{aligned}$$

or c must be the golden ratio. This implies that the secant method converges superlinearly.

2.4 Regula Falsi

2.5 Fixed Points

A fixed point is a point in the domain of a function, which maps its domain back into its domain, that satisfies $\alpha = C(\alpha)$. Since α does not change when it is mapped by the function it is fixed, hence the name. We need to look at what the idea that underlies fixed points: contractions. A contraction $y = C(x)$, is a mapping from a closed interval in X into another closed interval in Y with the property that for some $b = C(a)$ (usually a and b are both the origin but it is not required), $\|\cdot\|_x$ a norm on X, and $\|\cdot\|_y$ a norm on Y we have:

$$\|x - a\|_x > \|y - b\|_y = \|C(x) - C(a)\|_y$$

for all $x \in X$ and $y \in Y$. Usually we have X and Y are $\Re$ and $a = b$, which gives us that $|x - a| > |C(x) - a|$. Take the derivative of both sides and we see

$$1 > |C'(x)|.$$

This brings up a key point, we must have that the magnitude of the function's slope is less than 1. If you think about this it makes sense, as for slope magnitudes greater than one there will be growth and we are looking a funcions which shrink things. While this is a simple idea, it has many profound implications. The book proves nicely how the uniqueness of solution, convergence, etc.. One thing that should be highlated has to do with rate of convergence. Given a contraction defined on an interval $[a, b]$ with some point, $\alpha = C(\alpha) \in [a, b]$ called a fixed point, we can define the iteration $x_{n+1} = C(x_n)$. We then have (using the mean value theorem)

$$\begin{aligned} \alpha - x_{n+1} &= C(\alpha) - C(x_n) \\ &= C'(d)(\alpha - x_n) \\ |\alpha - x_{n+1}| &< |\alpha - x_n|. \end{aligned}$$

We have linear convergence from this. Consider the following paradox.

Let a function $g(x)$ be defined by

$$g(x) = x - \frac{f(x)}{f'(x)}$$

and let $f(x)$ have a single root in some interval $[a, b]$. From the book we know this must have a fixed point in the interval and the iteration $x_{n+1} = g(x_n)$ will converge to the fixed point. This method thus has linear convergence from what we have proven above. This iteration is Newton's Method though, so it has Quadratic convergence. What gives? The convergence of a fixed point algorithm is at least linear but it can be better if $C'(\alpha) = 0$. Notice that the derivative of $g(x)$ is given by

$$\begin{aligned} g'(x) &= 1 - \frac{(f'(x))^2 - f(x)f''(x)}{(f'(x))^2} \\ &= \frac{f(x)f''(x)}{(f'(x))^2}. \end{aligned}$$

Note that for $x = \alpha$ we trivially have that $g'(\alpha) = 0$, which satisfies our requirement for faster convergence.

How can I get a function $g(x)$ that satisfies the requirements? Many ways exist but consider the following. For a function $f(x)$ with a zero at $x = \alpha$ in an interval $[a, b]$, that has $\beta = \max_{x\in[a,b]} |f'(x)|$, we define the iteration

$$x_{n+1} = x_n - \gamma sign(f'(x_n))f(x_n)$$

with $0 < \gamma\beta < 2$. We then see that

$$\begin{aligned} g(x) &= x - \gamma sign(f'(x))f(x) \\ g'(x) &= 1 - \gamma sign(f'(x))f'(x) \\ &= 1 - \gamma|f'(x)| \end{aligned}$$

and thus $1 > g'(x) > -1$. Note that if we choose γ such that $0 < \gamma\beta < 1$ then $1 > g'(x) > 0$ and the sequence $\{x_i\}_{i=0}^{\infty}$ converges to α from one side (no alternating). The parameter γ is refered to as the **step size**. As a final note, we can use Aitken's Δ^2 method as outlined in the book to refine the estimate x_n. Replace α with $\hat{x}_n$ and you have a refinement and acceleration method that will work on any linearly convergent algorithm. It can thus be used on general fixed point methods.

Homework 4.3: 6, 13

2.6 Continuation Methods

One of the essential problems in root finding is to find a good place to start. We have spoken about the progressively doubling intervals till we find a sign change. I mentioned this was not the fastest or best, but would work. I wanted to give you what I think is one of the best. It is refered to as a continuation method or sometimes a homotopy.

A homotopy, h, is a continuous connection between two functions, f and g, that maps one space, X, to another, Y:

$$h : [0,1] \times X \to Y$$

such that $h(0,x) = g(x)$ and $h(1,x) = f(x)$. Two simple homotopies we will use are listed below.

1.

$$h(\lambda, x) = \lambda f(x) + (1-\lambda)g(x)$$

2.

$$\begin{aligned} h(\lambda, x) &= \lambda f(x) + (1-\lambda)(f(x) - f(\alpha_0)) \\ &= f(x) - (1-\lambda)f(\alpha_0) \end{aligned}$$

The first one is the most general. Assume we want to find the roots of f, but we know the roots of g. By picking a sequence of λ values from zero to one, we will slowly make the roots move from the known positions of g to the unknown positions of f. We usually try to pick g so it has the same number of roots as the function f.

The second method is a frequently used one if I don't want to find a function g. We are in essence biasing the original fucntion so that at α_0 the homotopy has a root for $\lambda = 0$. This gives a nice starting point. The following theorem tells us when this will work.

Theorem 1 (Ortega and Rheinboldt) *If $f : \mathbb{R}^n \to \mathbb{R}^n$ is continuously differentiable and if $\|[f'(x)]^{-1}\| \leq M$ on $\mathbb{R}^n$, then for any $\alpha_0 \in \mathbb{R}^n$ there is a unique curve $\{\alpha(\lambda) : 0 \leq \lambda \leq 1\}$ in $\mathbb{R}^n$ such that $f(\alpha(\lambda)) - (1-\lambda)f(\alpha_0) = 0$, with $0 \leq \lambda \leq 1$. The function $\lambda \mapsto \alpha(\lambda)$ is a continuously differentiable solution to the initial value problem $\alpha' = -[f'(\alpha)]^{-1}f(\alpha_0)$, where $\alpha(0) = \alpha_0$.*

Essentially this tells us if f is smooth and the first derivative doesn't get too close to zero then you can use this start one of our rootfinding methods, for instance Newton's Method. Often this method is solved by using a numerical integration technique which we will cover in a few weeks. For instance if Euler's method is used then it turns out to generate Newton's Method in λ!

2.7 Multiple Roots

One thing that always caused us problems in all our methods is multiple roots. I will present a simple technique for handling this case. Let our function, f, with root of multiplicity, 2, be given by

$$f(x) = (x-\alpha)^2 f_1(x),$$

where f_1 has no root at α. Take the derivative of $f(x)$ to obtain

$$\begin{aligned} f'(x) &= (x-\alpha)f_1(x) + (x-\alpha)^2 f_1'(x) \\ &= (x-\alpha)(f_1(x) + (x-\alpha)f_1'(x)) \\ &= (x-\alpha)f_2(x), \end{aligned}$$

where $f_2(x)$ has no root at α. We now have a funcition with a single root at the same place that the original function had a double root. This can be done for higher multiplicity roots, and does not require knowing α as we are taking the derivative then finding α using one of our techniques.

2.8 Sensitivity

This is refered to as stability of the roots in the books, but it is more closely related to the sensitivity of a differential equation to perturbations in its coefficients. For instance, consider a famous problem due to Wilkinson.

Problem 1 (Wilkinson) *Find the roots of the polynomial $f(x)$ given by*

$$\begin{aligned} f(x) &= (x-1)(x-2)\cdots(x-20) \\ &= x^{20} - 210x^{19} + \cdots + 20! \end{aligned}$$

The roots are clearly one through twenty. Perturb the coefficient -210 to $-210 - 2^{-23}$. The change is in one coefficient only, and that in the 7^{th} decimal place. The roots are now

1.000000000	6.000006944	$10.095266145 \pm 0.643500904j$
2.000000000	6.999697234	$11.793633881 \pm 1.652329728j$
3.000000000	8.007267603	$13.992358137 \pm 2.518830070j$
4.000000000	8.917250249	$16.730737466 \pm 2.812624894j$
4.999999928	20.846908101	$19.502439400 \pm 1.940330347j$

The problem is not round-off. The roots of high-order coefficients can be extremely sensitive to changes in the coefficients. This is a problem particularly when the coefficients are experimentally determined.

Chapter 3

Interpolation

We will now look at the problem of finding a polynomial to fit a set of points. The points could come from measurements in an experiment, or it could come from a complex function we want to approximate. In either case we will begin by considering the case where we want our polynomial to be exact at these values. An obvious question is why the emphasis on polynomials, when so many other functions exist. Indeed we do see the use of other basis (sin and cos in Fourier for example), but still polynomials hold a special place in many applications. One major reason is the Theorem of Wiestrass from Real Analysis. It basically says that polynomials can approximate any function (assuming you use the entire basis).

3.1 Lagrange Interpolation Basis

Probably the nicest way to visualize the interpolation polynomials is to consider the Lagrange interpolation basis functions. For the set of points, $\{x_0, x_1, \ldots, x_n\}$ define the following polynomial:

$$L_i(x) = \frac{\prod_{j \neq i}(x - x_j)}{\prod_{j \neq i}(x_i - x_j)}.$$

We note in particular that $L_i(x_j) = \delta_{i,j}$, which allows us to get the interpolation polynomial nicely. The interpolating polynomial is then given by

$$P_n(x) = \sum_{i=0}^{n} y_i L_i(x).$$

The importance of the Lagrange basis giving us the Kronecker delta function cannot be over-emphasized, as it is the essential idea in getting the solution.

Often the points are selected to be evenly spaced due to constraints in the basic system. While this is not the best for errors, it is often a physical necessity (for example many data samplers are constrained this way). In this case we can simplify the expression using

$$\mu = \frac{x - x_0}{x_1 - x_0}.$$

This is covered well in the book.

3.2 Newton's Divided Difference

Newton's divided difference is a similar method to Taylor approximation but instead of matching derivatives exactly at a point, it nearly approximates the derivative to exactly match certain points. They are the same if the interpolation points are made to coincide. The result is the same as Lagrange's formula. We will start our derivation of the formula by considering the Taylor approximation around the point x_0.

$$p_k(x) = \sum_{i=0}^{k} \frac{(x-x_0)^i}{i!} f^{(i)}(x_0)$$

The simplest case is when $k = 0$, in which case we have a horizontal line through $f(x_0)$.

$$\begin{aligned} p_0(x) &= \frac{(x-x_0)^0}{0!} f^{(0)}(x_0) \\ &= \frac{1}{1} f(x_0) \\ &= f(x_0) \end{aligned}$$

This is very easy to convert into a one point interpolation formula, as it is already one. I will use capitals for the divided difference formula.

$$P_0(x) = f(x_0)$$

Now let's consider the case when $k = 1$, in which case we have a line tangent to the curve through the point $(x_0, f(x_0))$.

$$\begin{aligned} p_1(x) &= \frac{(x-x_0)^0}{0!} f^{(0)}(x_0) + \frac{(x-x_0)^1}{1!} f^{(1)}(x_0) \\ &= f(x_0) + (x-x_0) f^{(1)}(x_0) \end{aligned}$$

To convert this we have to consider how to discritize the derivative. The first derivative is

$$\begin{aligned} f^{(1)}(x) &= \frac{d}{dx} f(x) \\ &= \lim_{x_1 \to x} \frac{f(x_1) - f(x)}{x_1 - x} \end{aligned}$$

We can approximate this by not allowing x_1 to go to x (i.e. remove the limit), then by noting we are consider the derivative at the point $x = x_0$, we have a neat expression.

$$\begin{aligned} f^{(1)}(x) &\approx F[x_0, x_1] \\ &= \frac{f(x_1) - f(x_0)}{x_1 - x_0} \end{aligned}$$

Putting this back into the expression we have the second divided difference formula

$$\begin{aligned} P_1(x) &= f(x_0) + (x - x_0)F[x_0, x_1] \\ &= f(x_0) + (x - x_0)\frac{f(x_1) - f(x_0)}{x_1 - x_0} \\ &= P_0(x) + (x - x_0)\frac{f(x_1) - f(x_0)}{x_1 - x_0} \end{aligned}$$

Just to show that this is the same as the first Lagrange interpolator

$$\begin{aligned} P_1(x) &= f(x_0) + (x - x_0)\frac{f(x_1) - f(x_0)}{x_1 - x_0} \\ &= f(x_0)\frac{x_1 - x_0}{x_1 - x_0} + (f(x_1) - f(x_0))\frac{x - x_0}{x_1 - x_0} \\ &= f(x_0)\frac{x_1 - x_0}{x_1 - x_0} + f(x_1)\frac{x - x_0}{x_1 - x_0} - f(x_0)\frac{x - x_0}{x_1 - x_0} \\ &= f(x_0)\frac{x_1 - x_0}{x_1 - x_0} - f(x_0)\frac{x - x_0}{x_1 - x_0} + f(x_1)\frac{x - x_0}{x_1 - x_0} \\ &= f(x_0)\frac{x_1 - x}{x_1 - x_0} + f(x_1)\frac{x - x_0}{x_1 - x_0} \end{aligned}$$

The final line is Lagrange's interpolator for two points.

Let's do one more step, then I will show the general solution. The reason for the extra step is it shows the difference between Taylor and Newton.

$$\begin{aligned} p_2(x) &= \frac{(x - x_0)^0}{0!}f^{(0)}(x_0) + \frac{(x - x_0)^1}{1!}f^{(1)}(x_0) + \frac{(x - x_0)^2}{2!}f^{(2)}(x_0) \\ &= f(x_0) + (x - x_0)f^{(1)}(x_0) + (x - x_0)^2\frac{f^{(2)}(x_0)}{2} \end{aligned}$$

Before we do anything else, we have to unwind the $(x - x_0)^2$ term. Why you may ask? Is it not already in a usable form? Well yes and no. As I stated in the beginning, Taylor assumed coinciding points, but Newton assumed different points to be interpolated. Up until now this has not been an issue. This is the first challenge. Newton used basis polynomials of the form $n_k = \prod_{i=0}^{k-1}(x - x_i)$, so we have to use $(x - x_0)(x - x_1)$ for this case instead of $(x - x_0)^2$. Note that in general we will replace $(x - x_0)^k$ by $\prod_{i=0}^{k-1}(x - x_i)$.

Now on to the derivative term. It might seem odd to combine the factorial term with the derivative, but it has practicality and some intuition on its side. Let me first give the formula.

$$\begin{aligned} \frac{f^{(2)}(x_0)}{2} &= F[x_0, x_1, x_2] \\ &= \frac{F[x_1, x_2] - F[x_0, x_1]}{x_2 - x_0} \\ &= \frac{\frac{f(x_2) - f(x_1)}{x_2 - x_1} - \frac{f(x_1) - f(x_0)}{x_1 - x_0}}{x_2 - x_0} \end{aligned}$$

At this point I can explain how the coll If the points are equidistant, then this makes lots of since as the denominator of the final term is then twice the step size. When you would go to the third derivative you would have the difference of two second derivatives that would have the one half scaling built in, divided by a length that was three times the size so this would have to have a factor of three in the denominator. The one third from the length together with the one half from the second derivative gives a factor of one over three factorial. Since the next term up would use the previous term, but have a larger denominator, we have an induction step[1]

Putting this back into the expression we have the second divided difference formula

$$\begin{aligned}
P_2(x) &= f(x_0) + (x-x_0)F[x_0,x_1] + (x-x_0)(x-x_1)F[x_0,x_1,x_2] \\
&= f(x_0) + (x-x_0)\frac{f(x_1)-f(x_0)}{x_1-x_0} + (x-x_0)(x-x_1)\frac{\frac{f(x_2)-f(x_1)}{x_2-x_1} - \frac{f(x_1)-f(x_0)}{x_1-x_0}}{x_2-x_0} \\
&= P_1(x) + (x-x_0)(x-x_1)F[x_0,x_1,x_2] \\
&= P_1(x) + (x-x_0)(x-x_1)\frac{\frac{f(x_2)-f(x_1)}{x_2-x_1} - \frac{f(x_1)-f(x_0)}{x_1-x_0}}{x_2-x_0}
\end{aligned}$$

Just to show that this is the same as the second Lagrange interpolator

$$\begin{aligned}
P_2(x) &= P_1(x) + (x-x_0)(x-x_1)(x-x_0)(x-x_1)\frac{\frac{f(x_2)-f(x_1)}{x_2-x_1} - \frac{f(x_1)-f(x_0)}{x_1-x_0}}{x_2-x_0} \\
&= f(x_0)\frac{x_1-x}{x_1-x_0} + f(x_1)\frac{x-x_0}{x_1-x_0} \\
&\quad +(x-x_0)(x-x_1)\frac{\frac{f(x_2)-f(x_1)}{x_2-x_1} - \frac{f(x_1)-f(x_0)}{x_1-x_0}}{x_2-x_0} \\
&= f(x_0)\frac{x_1-x}{x_1-x_0} + f(x_1)\frac{x-x_0}{x_1-x_0} \\
&\quad +(x-x_0)(x-x_1)\left(\frac{f(x_2)-f(x_1)}{(x_2-x_1)(x_2-x_0)} - \frac{f(x_1)-f(x_0)}{(x_1-x_0)(x_2-x_0)}\right) \\
&= f(x_0)\frac{x_1-x}{x_1-x_0} + f(x_1)\frac{x-x_0}{x_1-x_0} \\
&\quad +(f(x_2)-f(x_1))\frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)} - (f(x_1)-f(x_0))\frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_1-x_0)} \\
&= f(x_0)\frac{x_1-x}{x_1-x_0} + f(x_0)\frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_1-x_0)} \\
&\quad +f(x_1)\frac{x-x_0}{x_1-x_0} - f(x_1)\frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)} - f(x_1)\frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_1-x_0)} \\
&\quad +f(x_2)\frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}
\end{aligned}$$

[1]This is probably not obvious at this point, but I have left it this way to give you a challenge. Sorry but trying is the only way to learn. Try to write the recursion, and drop by or send an email if you don't get it.

$$
\begin{aligned}
P_2(x) &= f(x_0)\left(\frac{(x_0-x_2)(x-x_1)}{(x_0-x_2)(x_0-x_1)}+\frac{(x-x_0)(x-x_1)}{(x_0-x_2)(x_0-x_1)}\right)\\
&\quad +f(x_1)\frac{x-x_0}{x_1-x_0}-f(x_1)\frac{(x-x_0)(x-x_1)}{x_2-x_0}\left(\frac{1}{x_2-x_1}+\frac{1}{x_1-x_0}\right)\\
&\quad +f(x_2)\frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}\\
&= f(x_0)\frac{(x-x_0+x_0-x_2)(x-x_1)}{(x_0-x_2)(x_0-x_1)}\\
&\quad +f(x_1)\frac{x-x_0}{x_1-x_0}-f(x_1)\frac{(x-x_0)(x-x_1)}{x_2-x_0}\left(\frac{x_1-x_0+x_2-x_1}{(x_2-x_1)(x_1-x_0)}\right)\\
&\quad +f(x_2)\frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}\\
&= f(x_0)\frac{(x-x_2)(x-x_1)}{(x_0-x_2)(x_0-x_1)}\\
&\quad +f(x_1)\frac{x-x_0}{x_1-x_0}-f(x_1)\frac{(x-x_0)(x-x_1)}{x_2-x_0}\left(\frac{x_2-x_0}{(x_2-x_1)(x_1-x_0)}\right)\\
&\quad +f(x_2)\frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}\\
&= f(x_0)\frac{(x-x_2)(x-x_1)}{(x_0-x_2)(x_0-x_1)}\\
&\quad +f(x_1)\frac{(x_1-x_2)(x-x_0)}{(x_1-x_2)(x_1-x_0)}+f(x_1)\frac{(x-x_0)(x-x_1)}{(x_1-x_2)(x_1-x_0)}\\
&\quad +f(x_2)\frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}\\
&= f(x_0)\frac{(x-x_2)(x-x_1)}{(x_0-x_2)(x_0-x_1)}\\
&\quad +f(x_1)\frac{(x-x_1+x_1-x_2)(x-x_0)}{(x_1-x_2)(x_1-x_0)}\\
&\quad +f(x_2)\frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}\\
&= f(x_0)\frac{(x-x_2)(x-x_1)}{(x_0-x_2)(x_0-x_1)}+f(x_1)\frac{(x-x_2)(x-x_0)}{(x_1-x_2)(x_1-x_0)}+f(x_2)\frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}
\end{aligned}
$$

The last line is the second Lagrange polynomial. It took a little algebra, but I think it is worthwhile to try a

3.2.1 General Form

Derivatives

$$\begin{aligned} F[x_0, x_1] &= \frac{f(x_1) - f(x_0)}{x_1 - x_0} \\ F[x_0, x_1, \cdots, x_n] &= \frac{F[x_1, \cdots, x_n] - F[x_0, \cdots, x_{n-1}]}{x_n - x_0} \end{aligned}$$

Polynomial Recursion

$$\begin{aligned} P_0(x) &= f(x_0) \\ P_{k+1} &= P_k + (x - x_0)(x - x_1) \cdots (x - x_k) F[x_0, x_1, \cdots, x_{k+1}] \end{aligned}$$

3.2.2 Error

The key area to note from here is that the error is given by either of the following formulas.

$$\begin{aligned} f(x) - P_n(x) &= \prod_{i=0}^{n} (x - x_i) \frac{f^{(n+1)}(c_x)}{(n+1)!} \\ &= \prod_{i=0}^{n} (x - x_i) F[x_0, x_1, \cdots, x_n, x] \end{aligned}$$

The important part of this is to note that these are themselves polynomials of order $n + 1$. Consider the plot of a polynomial with equi-spaced roots. It is trivial to note that the height of the peaks between the roots is bigger towards the outside of the interval.

3.3 Tchebychev Polynomials

Tchebychev (also spelled Chebyshev) polynomials are the basis set of polynomials that reduces the maximum error in interpolation by putting more of the zero crossings to the edges. The Tchebychev polynomials are defined on the interval $[-1, 1]$ by the following recursion

$$T_0 = 1 \tag{3.1}$$

$$T_1 = x \tag{3.2}$$

$$T_n = 2xT_{n-1} - T_{n-2} \tag{3.3}$$

The first several Tchebychev polynomials are

$$\begin{aligned}
T_0(x) &= 1 && (3.4)\\
T_1(x) &= x && (3.5)\\
T_2(x) &= 2x^2 - 1 && (3.6)\\
T_3(x) &= 2x(2x^2 - 1) - x && (3.7)\\
&= 4x^3 - 3x && (3.8)\\
T_4(x) &= 2x(4x^3 - 3x) - (2x^2 - 1) && (3.9)\\
&= 8x^4 - 8x^2 + 1 && (3.10)\\
T_5(x) &= 2x(8x^4 - 8x^2 + 1) - (4x^3 - 3x) && (3.11)\\
&= 16x^5 - 20x^3 + 5x && (3.12)\\
T_6(x) &= 2x(16x^5 - 20x^3 + 5x) - (8x^4 - 8x^2 + 1) && (3.13)\\
&= 32x^6 - 48x^4 + 18x^2 - 1 && (3.14)\\
T_7(x) &= 2x(32x^6 - 48x^4 + 18x^2 - 1) - (16x^5 - 20x^3 + 5x) && (3.15)\\
&= 64x^7 - 112x^5 + 56x^3 - 7x && (3.16)
\end{aligned}$$

The roots of the Tchebychev polynomial of degree $n+1$, $T_{n+1}(x)$, are given by

$$\begin{aligned}
T_{n+1}(x_k) &= 0 \qquad \Longleftrightarrow && (3.17)\\
x_k &= \cos\left(\frac{2n+1-2k}{2n+2}\pi\right) \ \forall k \in [1,\ldots,n] && (3.18)
\end{aligned}$$

3.4 Splines

For splines we want to fit a cubic polynomial for each interval so that the first and second derivatives between two sections match on the boundary. Thus for n points we need $n-1$ spline sections.

$$S(x) = \begin{cases} S_0(x) & x_0 \le x \le x_1 \\ S_1(x) & x_1 \le x \le x_2 \\ \vdots & \vdots \\ S_i(x) & x_i \le x \le x_{i+1} \\ \vdots & \vdots \\ S_{n-2}(x) & x_{n-2} \le x \le x_{n-1} \end{cases} \tag{3.19}$$

with[2]

$$S_i(x) = a_i(x - x_i)^3 + b_i(x - x_i)^2 + c_i(x - x_i) + d_i \tag{3.23}$$

[2]Note that we could have defined this lots of different ways. For instance, one popular way of developing the same set of equations we will find is to start from

$$s(x) = a_1(x_j - x)^3 + a_0(x - x_{j-1})^3 + b_1(x_j - x) + b_0(x - x_{j-1}) \tag{3.20}$$

Each spline section has four requirements

1. It must go through the boundary point on its left,

$$\begin{aligned} y_i &= S_i(x_i) && (3.24)\\ &= a_i(x_i - x_i)^3 + b_i(x_i - x_i)^2 + c_i(x_i - x_i) + d_i && (3.25)\\ y_i &= d_i && (3.26) \end{aligned}$$

2. It must go through the boundary point on its right,

$$\begin{aligned} y_{i+1} &= S_i(x_{i+1}) && (3.27)\\ &= a_i(x_{i+1} - x_i)^3 + b_i(x_{i+1} - x_i)^2 + c_i(x_{i+1} - x_i) + d_i && (3.28)\\ &= a_i(x_{i+1} - x_i)^3 + b_i(x_{i+1} - x_i)^2 + c_i(x_{i+1} - x_i) + y_i && (3.29)\\ y_{i+1} - y_i &= a_i(x_{i+1} - x_i)^3 + b_i(x_{i+1} - x_i)^2 + c_i(x_{i+1} - x_i) && (3.30)\\ \frac{y_{i+1} - y_i}{x_{i+1} - x_i} &= a_i(x_{i+1} - x_i)^2 + b_i(x_{i+1} - x_i) + c_i && (3.31)\\ c_i &= \frac{y_{i+1} - y_i}{x_{i+1} - x_i} - a_i(x_{i+1} - x_i)^2 - b_i(x_{i+1} - x_i) && (3.32) \end{aligned}$$

3. It must have continuous slopes with the splines on either side,

$$\dot{S}_i(x) = 3a_i(x - x_i)^2 + 2b_i(x - x_i) + c_i \quad (3.33)$$

and thus

$$\begin{aligned} \dot{S}_{i-1}(x_i) &= \dot{S}_i(x_i) && (3.34)\\ 3a_{i-1}(x_i - x_{i-1})^2 + 2b_{i-1}(x_i - x_{i-1}) + c_{i-1} &= 3a_i(x_i - x_i)^2 + 2b_i(x_i - x_i) + c_i && (3.35)\\ 3a_{i-1}(x_i - x_{i-1})^2 + 2b_{i-1}(x_i - x_{i-1}) + c_{i-1} &= c_i && (3.36)\\ 3a_{i-1}(x_i - x_{i-1})^2 + 2b_{i-1}(x_i - x_{i-1}) &= c_i - c_{i-1} && (3.37) \end{aligned}$$

4. It must have continuous slopes with the splines on either side,

$$\ddot{S}_i(x) = 6a_i(x - x_i) + 2b_i \quad (3.38)$$

Which would give us (these will make sense in a couple pages)

$$\begin{aligned} a_i &= \frac{M_{j-i}}{6(x_j - x_{j-1})} && (3.21)\\ b_i &= \frac{y_{j-i} - \frac{1}{6}M_{j-i}(x_j - x_{j-1})^2}{(x_j - x_{j-1})} && (3.22) \end{aligned}$$

and thus

$$\begin{aligned}
\ddot{S}_{i-1}(x_i) &= \ddot{S}_i(x_i) && (3.39)\\
6a_{i-1}(x_i - x_{i-1}) + 2b_{i-1} &= 6a_i(x_i - x_i) + 2b_i && (3.40)\\
6a_{i-1}(x_i - x_{i-1}) + 2b_{i-1} &= 2b_i && (3.41)\\
6a_{i-1}(x_i - x_{i-1}) &= 2b_i - 2b_{i-1} && (3.42)\\
a_{i-1} &= \frac{2b_i - 2b_{i-1}}{6(x_i - x_{i-1})} && (3.43)
\end{aligned}$$

We could just stick the four resulting equations into a big matrix and solve, but that is not very efficient as there are $4n-4$ equations to be solved, it would be nice to simplify things. We note that the term $2b_i$ appears a lot, so we make the following definition

$$M_i = 2b_i \tag{3.44}$$

and then solve for each of the four unknowns in terms of the new variable and the x_k and y_k terms. The four equations we have from above with our definition make five key equations for our problem

1. $d_i = y_i$
2. $c_i = \frac{y_{i+1}-y_i}{x_{i+1}-x_i} - a_i(x_{i+1} - x_i)^2 - b_i(x_{i+1} - x_i)$
3. $3a_{i-1}(x_i - x_{i-1})^2 + 2b_{i-1}(x_i - x_{i-1}) = c_i - c_{i-1}$
4. $a_i = \frac{2b_{i+1}-2b_i}{6(x_{i+1}-x_i)}$
5. $b_i = \frac{1}{2}M_i$

The first one is already ok, as is the fifth (the definition). The fourth one is easy to put into the new variables

$$\begin{aligned}
a_i &= \frac{2b_{i+1} - 2b_i}{6(x_{i+1} - x_i)} && (3.45)\\
&= \frac{M_{i+1} - M_i}{6(x_{i+1} - x_i)}. && (3.46)
\end{aligned}$$

We now have ways to calculate a_i, b_i, and d_i in the new variables. We only need an equation for the c_i and another to find the M_i. Conveniently we still have two more equations. The second one in our list gives c_i in terms of a_i and b_i but these are known in terms of M_i so

we can just substitute

$$\begin{aligned} c_i &= \frac{y_{i+1}-y_i}{x_{i+1}-x_i} - a_i(x_{i+1}-x_i)^2 - b_i(x_{i+1}-x_i) && (3.47)\\ &= \frac{y_{i+1}-y_i}{x_{i+1}-x_i} - \left(\frac{M_{i+1}-M_i}{6(x_{i+1}-x_i)}\right)(x_{i+1}-x_i)^2 - \left(\frac{1}{2}M_i\right)(x_{i+1}-x_i) && (3.48)\\ &= \frac{y_{i+1}-y_i}{x_{i+1}-x_i} - \left(\frac{M_{i+1}-M_i}{6} + \frac{1}{2}M_i\right)(x_{i+1}-x_i) && (3.49)\\ &= \frac{y_{i+1}-y_i}{x_{i+1}-x_i} - \left(\frac{1}{6}M_{i+1} + \frac{1}{3}M_i\right)(x_{i+1}-x_i) && (3.50) \end{aligned}$$

Now we only need to find the values of the M_i and we can then find the values of the other variables, thus instead of solving for $4n-4$ unknowns we only need to find $n-1$ unknowns. Since solving for them will take n^3 operations, all that math cut the work down by 4^3 or to a sixty fourth of the brute force idea. The equation for the M_i comes from equation 3 in the list. First let's simplify the left hand side.

$$\begin{aligned} 3a_{i-1}(x_i-x_{i-1})^2 + 2b_{i-1}(x_i-x_{i-1}) &= c_i - c_{i-1} && (3.51)\\ 3\frac{M_i-M_{i-1}}{6(x_i-x_{i-1})}(x_i-x_{i-1})^2 + M_{i-1}(x_i-x_{i-1}) &= c_i - c_{i-1} && (3.52)\\ \frac{M_i-M_{i-1}}{2}(x_i-x_{i-1}) + M_{i-1}(x_i-x_{i-1}) &= c_i - c_{i-1} && (3.53)\\ (M_i+M_{i-1})\frac{x_i-x_{i-1}}{2} &= c_i - c_{i-1} && (3.54) \end{aligned}$$

Now lets simplify the right hand side.

$$\begin{aligned} (M_i+M_{i-1})\frac{x_i-x_{i-1}}{2} &= \left(\frac{y_{i+1}-y_i}{x_{i+1}-x_i} - \left(\frac{1}{6}M_{i+1}+\frac{1}{3}M_i\right)(x_{i+1}-x_i)\right)\\ &\quad - \left(\frac{y_i-y_{i-1}}{x_i-x_{i-1}} - \left(\frac{1}{6}M_i+\frac{1}{3}M_{i-1}\right)(x_i-x_{i-1})\right) && (3.55)\\ &= \left(\frac{y_{i+1}-y_i}{x_{i+1}-x_i} - \frac{y_i-y_{i-1}}{x_i-x_{i-1}}\right)\\ &\quad -\frac{1}{6}M_{i+1}(x_{i+1}-x_i) - \frac{1}{3}M_i(x_{i+1}-x_i)\\ &\quad +\frac{1}{6}M_i(x_i-x_{i-1}) + \frac{1}{3}M_{i-1}(x_i-x_{i-1}) && (3.56) \end{aligned}$$

Now collect terms in M_i onto the left hand side, leaving the terms with no M_i on the right.

$$\begin{aligned}
\left(\frac{x_{i+1}-x_i}{6}\right) M_{i+1} + \\
\left(\frac{x_i - x_{i-1}}{2} - \frac{x_i - x_{i-1}}{6} + \frac{x_{i+1}-x_i}{3}\right) M_i + \\
\left(\frac{x_i - x_{i-1}}{2} - \frac{x_i - x_{i-1}}{3}\right) M_{i-1} &= \frac{y_{i+1}-y_i}{x_{i+1}-x_i} - \frac{y_i - y_{i-1}}{x_i - x_{i-1}} \quad (3.57) \\
\frac{x_{i+1}-x_i}{6} M_{i+1} + \frac{x_{i+1}-x_{i-1}}{3} M_i + \frac{x_i - x_{i-1}}{6} M_{i-1} &= \frac{y_{i+1}-y_i}{x_{i+1}-x_i} - \frac{y_i - y_{i-1}}{x_i - x_{i-1}} \quad (3.58)
\end{aligned}$$

The only thing that we need is to calculate M_i for the natural cubic spline, which is done by requiring $M_1 = M_n = 0$ and solving the following matrix system

$$\begin{aligned}
Ax &= b \\
A &= \begin{bmatrix} \alpha_2 & \beta_2 & 0 & \cdots & 0 \\ \beta_2 & \alpha_3 & \beta_3 & \ddots & \vdots \\ 0 & \beta_3 & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \alpha_{n-2} & \beta_{n-2} \\ 0 & \cdots & 0 & \beta_{n-2} & \alpha_{n-1} \end{bmatrix} \\
x &= \begin{bmatrix} M_2 \\ \vdots \\ M_{n-1} \end{bmatrix} \qquad b = \begin{bmatrix} \gamma_2 - \gamma_1 \\ \vdots \\ \gamma_{n-1} - \gamma_{n-2} \end{bmatrix} \\
\alpha_i &= \frac{x_{i+1} - x_{i-1}}{3} \qquad \beta_i = \frac{x_{i+1} - x_i}{6} \qquad \gamma_i = \frac{y_{i+1} - y_i}{x_{i+1} - x_i}
\end{aligned}$$

When we have found the M_i we can then substitute these values to find the original splines, and thus plot them.

$$\begin{aligned}
S_i(x) &= a_i(x - x_i)^3 + b_i(x - x_i)^2 + c_i(x - x_i) + d_i && (3.59) \\
a_i &= \frac{M_{i+1} - M_i}{6(x_{i+1} - x_i)} && (3.60) \\
b_i &= \frac{1}{2} M_i && (3.61) \\
c_i &= \frac{y_{i+1} - y_i}{x_{i+1} - x_i} - \left(\frac{1}{6} M_{i+1} + \frac{1}{3} M_i\right)(x_{i+1} - x_i) && (3.62) \\
d_i &= y_i && (3.63)
\end{aligned}$$

3.4.1 Not-a-Knot Cubic Spline

We can also find the M_i for the not-a-knot cubic spline, which is often preferred by solving a similar system

$$
\begin{aligned}
Ax &= b \\
A &= \begin{bmatrix}
\psi_1 & \beta_1 & 0 & 0 & \cdots & 0 & 0 \\
\beta_1 & \alpha_2 & \beta_2 & 0 & \cdots & 0 & 0 \\
0 & \beta_2 & \alpha_3 & \beta_3 & \ddots & \vdots & \vdots \\
0 & 0 & \beta_3 & \ddots & \ddots & 0 & 0 \\
\vdots & \vdots & \ddots & \ddots & \alpha_{n-2} & \beta_{n-2} & 0 \\
0 & 0 & \cdots & 0 & \beta_{n-2} & \alpha_{n-1} & \beta_{n-1} \\
0 & 0 & \cdots & 0 & 0 & \beta_{n-1} & \phi_2
\end{bmatrix} \\
x &= \begin{bmatrix} M_1 \\ M_2 \\ \vdots \\ M_{n-1} \\ M_n \end{bmatrix} \qquad b = \begin{bmatrix} \gamma_1 - f'(x_1) \\ \gamma_2 - \gamma_1 \\ \vdots \\ \gamma_{n-1} - \gamma_{n-2} \\ f'(x_n) - \gamma_{n-1} \end{bmatrix} \\
\alpha_i &= \frac{x_{i+1} - x_{i-1}}{3} \qquad \beta_i = \frac{x_{i+1} - x_i}{6} \qquad \gamma_i = \frac{y_{i+1} - y_i}{x_{i+1} - x_i} \\
\psi_1 &= \frac{x_2 - x_1}{3} \qquad \phi_2 = \frac{x_n - x_{n-1}}{3}
\end{aligned}
$$

or (if you don't know the derivative)

$$
\begin{aligned}
Ax &= b \\
A &= \begin{bmatrix}
\psi_1 & \psi_2 & 0 & 0 & \cdots & 0 & 0 \\
\beta_1 & \alpha_2 & \beta_2 & 0 & \cdots & 0 & 0 \\
0 & \beta_2 & \alpha_3 & \beta_3 & \ddots & \vdots & \vdots \\
0 & 0 & \beta_3 & \ddots & \ddots & 0 & 0 \\
\vdots & \vdots & \ddots & \ddots & \alpha_{n-2} & \beta_{n-2} & 0 \\
0 & 0 & \cdots & 0 & \beta_{n-2} & \alpha_{n-1} & \beta_{n-1} \\
0 & 0 & \cdots & 0 & 0 & \phi_2 & \phi_1
\end{bmatrix} \\
x &= \begin{bmatrix} M_1 \\ M_2 \\ \vdots \\ M_{n-1} \\ M_n \end{bmatrix} \qquad b = \begin{bmatrix} \psi_3 \\ \gamma_2 - \gamma_1 \\ \vdots \\ \gamma_{n-1} - \gamma_{n-2} \\ \phi_3 \end{bmatrix} \\
\alpha_i &= \frac{x_{i+1} - x_{i-1}}{3} \qquad \beta_i = \frac{x_{i+1} - x_i}{6} \qquad \gamma_i = \frac{y_{i+1} - y_i}{x_{i+1} - x_i} \\
\xi_1 &= x_2 - x_1 \qquad \xi_2 = x_2 - z_1 \qquad \xi_3 = z_1 - x_1 \\
\psi_1 &= \frac{\xi_2^3 - \xi_1^2 \xi_2}{6\xi_1} \qquad \psi_2 = \frac{\xi_3^3 - \xi_1^2 \xi_3}{6\xi_1} \qquad \psi_3 = f(z_1) - \frac{\xi_2 y_1 + \xi_3 y_2}{\xi_1} \\
\xi_4 &= x_n - x_{n-1} \qquad \xi_5 = x_n - z_2 \qquad \xi_6 = z_2 - x_{n-1} \\
\phi_1 &= \frac{\xi_5^3 - \xi_4^2 \xi_5}{6\xi_4} \qquad \phi_2 = \frac{\xi_6^3 - \xi_4^2 \xi_6}{6\xi_4} \qquad \phi_3 = f(z_2) - \frac{\xi_5 y_{n-1} + \xi_6 y_n}{\xi_4}
\end{aligned}
$$

Note, you can easily enter the matrix A into Matlab by using the command diag. For instance, if you put the entries of A that are on the main diagonal into the vector $A1$, the first sub-diagonal into $A2$, and the first super-diagonal into $A3$, then in Matlab you enter, *A=diag(A1)+diag(A2,-1)+diag(A3,1);*.

Homework

section 5.3: 7 section 5.4: 3, 5

Chapter 4

Approximation

Up till now we have dealt with interpolation, where we want to exactly match a set of points. In reality, we are often more concerned with having a good overall approximation rather than an exact matching at a few points. There are a lot of ways to approximate a function. In general there are two main areas discrete and continuous. We will cover the discrete case. The continuous method involves some functional analysis and we do not have the time to cover it well. If you are interested it can provide a fun project, and I have some good resources you can use.

4.1 Least Squares Approximation

We proceed with the discrete case. The discrete case involves measuring the function to be approximated at a series of points, and then finding the best coefficients in some sense for some functions of interest.

Some sense? What do I mean by that? Well, put simply, there are a variety of different methods of measuring how good an approximation is. The standard method is the one we will concentrate on, and it is called least squares. As with many things in Math, least squares owes its basis to Gauss, who used it to calculate orbits of objects in the solar system [28]. Numerous other works have covered solution methods [6, 44] and solved cases such as special structure [14, 16], sparse matrices [17, 29], and numerical issues [7, 13, 57]. Least squares assumes that the matrix, A is known exactly ($E_A = 0$), and thus all errors occur in the observations, b, only. It is sometimes also referred to as Errors-in-Observations, due to the assumption of the errors occurring only in b. The basic idea is to reduce the sum of the squares of the distances from the measurements to the function to be fitted at each of the x values. The problem can be stated as

$$\min_x \|Ax - b\|^2. \tag{4.1}$$

The last point is very important because it is the basis of much of the problems in least squares. In essence the answer you get is dependent on your choice of independent variables.

The key idea to get is that there are reasons to look beyond least squares.

Consider the problem of calibrating a gas thermometer. Gas thermometers are based on Charles' law, which states that the volume of a fixed mass of gas at a fixed pressure is proportional to its temperature. A simple gas thermometer can be made by trapping some gas with a mercury plug in a capillary tube that is open on only one end [48]. The volume is thus proportional to the height of the plug. The equation of the thermometer is thus $hc_1 = T$, where h is the height of the plug, c_1 is the constant we want to know, and T is the absolute temperature. We place the gas thermometer in a stirred liquid bath with a known thermometer. We heat the bath and take height and temperature measurements at various times. The LS solution gives us that $\hat{c}_1 = h^\dagger T$, but we can see that this minimizes the error in the measured temperature, T, from the predicted temperature, $hh^\dagger T$. By the same token we could use the relation $h = c_2 T$, with $c_2 = \frac{1}{c_1}$. The LS solution, $\hat{c}_2 = T^\dagger h$, thus minimizes the error between the measured height, h, and the predicted height $TT^\dagger h$. A problem arises in the LS method in that generally $\hat{c}_1 \neq \frac{1}{\hat{c}_2}$. This can be seen easily in Figure 4.1. The slope of the line designated temperature errors, is $\hat{c}_1$, while the slope of the line designated height errors is $\frac{1}{\hat{c}_2}$. The line designated theoretical is the "true" system from which the estimates were generated. It is easy to see that the slopes are not the same, and thus $\hat{c}_1 \neq \frac{1}{\hat{c}_2}$. The LS solution does not even perfectly handle the case where the system matrix is "known", which gives us cause to be concerned as to how it will perform when there are perturbations to the system matrix.

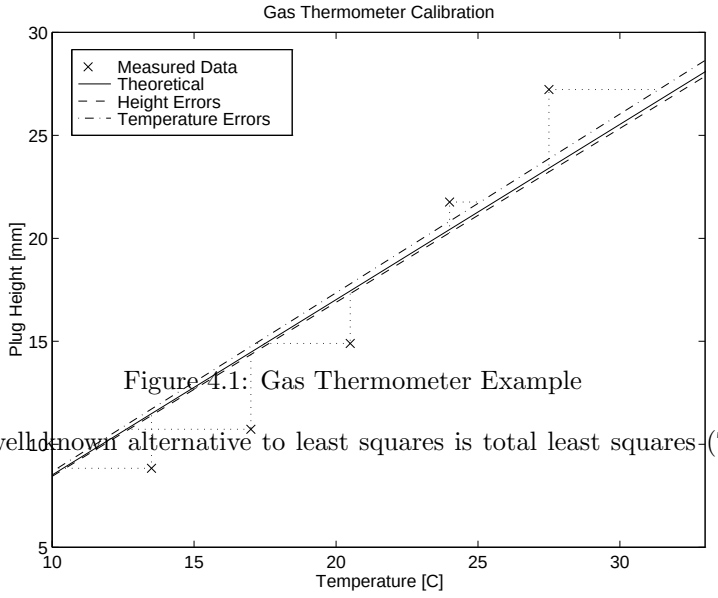


Figure 4.1: Gas Thermometer Example

The most well-known alternative to least squares is total least squares (TLS). In TLS

we look at the perpendicular distance to the function. This handles many of the problems of least squares but is more sensitive to errors, as it is "optimistic" in how it looks at the problem. A huge body of literature is dedicated to this problem, and this is the central area of my dissertation (available at www.r2labs.org). While some of these other methods are very interesting, we will stick to least squares for the moment, but we will remember that problems can occur and so if we have problems we know there are things we can do.

Getting back to business we have a set of m points (x_i, y_i) and a group of n functions $\phi_i(x)$ that we want to use to approximate the points with. We thus have m equations to find n coefficients.

$$\begin{aligned} y_1 &- \sum_{i=1}^{n} a_i \phi_i(x_1) \\ y_2 &- \sum_{i=1}^{n} a_i \phi_i(x_2) \\ &\vdots \\ y_m &- \sum_{i=1}^{n} a_i \phi_i(x_m) \end{aligned}$$

We can rewrite these into a matrix formulation, as

$$Y - \Phi A$$

where

$$\begin{aligned} Y &= \begin{bmatrix} y_1 & y_2 & \cdots & y_m \end{bmatrix}^T \\ \Phi &= \begin{bmatrix} \phi_1(x_1) & \phi_2(x_1) & \cdots & \phi_n(x_1) \\ \phi_1(x_2) & \phi_2(x_2) & \cdots & \phi_n(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(x_m) & \phi_2(x_m) & \cdots & \phi_n(x_m) \end{bmatrix} \\ A &= \begin{bmatrix} a_1 & a_2 & \cdots & a_m \end{bmatrix}^T. \end{aligned}$$

At this point we want to minimize the square error which is what the 2-norm does, so we have $\min_A \|Y - \Phi A\|_2^2$. The norm we are minimizing is called the cost function. The solution is given by $A = \Phi^\dagger Y$, where $\Phi^\dagger$ is called the pseudo-inverse of Φ. Prove it? Sure! To avoid getting into some deeper areas of linear algebra we will assume that Φ has linearly independent columns. This is not restrictive, as we usually have a lot of measurements and only a few functions we want to fit to them ($m >> n$).

We recall from calculus that the minimum occurs when the gradient (derivative) is zero.

We thus take the gradient of the cost with respect to A and set it equal to zero to obtain

$$\begin{aligned} 0 &= \nabla_A \|Y - \Phi A\|_2^2 \\ &= \nabla_A (Y - \Phi A)^T (Y - \Phi A) \\ &= -\Phi^T (Y - \Phi A) \\ &= \Phi^T \Phi A - \Phi^T Y \\ \Phi^T Y &= \Phi^T \Phi A \end{aligned}$$

The last line is what is referred to as the normal equation(s). Note that some pluralize it to reflect that the single matrix equation reflects n scalar equations. I don't care, use what you like. We note that if Φ has linearly independent columns, then $(\Phi^T \Phi)^{-1}$ exists.

$$\begin{aligned} \Phi^T \Phi A &= \Phi^T Y \\ A &= (\Phi^T \Phi)^{-1} \Phi^T Y \\ A &= \Phi^\dagger Y \end{aligned}$$

You might wonder how the last step works. Some might just call it a definition but in reality it is because $(\Phi^T \Phi)^{-1} \Phi^T$ satisfies the four conditions of a pseudo inverse (called the Penrose conditions).

1. $\Phi \Phi^\dagger \Phi = \Phi$
2. $\Phi^\dagger \Phi \Phi^\dagger = \Phi^\dagger$
3. $\Phi \Phi^\dagger = (\Phi \Phi^\dagger)^T$
4. $\Phi^\dagger \Phi = (\Phi^\dagger \Phi)^T$

The properties are simple and easy to check, and yes, you have to check all four. Many times a candidate matrix fails only one of them. The first two properties tell us that it correctly maps the range spaces from the fundamental theorem of linear algebra, and the second two tell us the composite maps are symmetric. The pseudo-inverse always exists and is unique. Additionally, when the true inverse exists, it is the pseudo-inverse. These are just a few of the many reasons to love the pseudo-inverse...

The result is established. The nice thing about how we have handled things here is we have not specified what the functions are (they have to be linearly independent but that is no problem) or how many of them we want to fit. You can now fit any combination of functions you like.

The solution is the same for a deterministic assumption, as for an assumption that additive zero mean gaussian noise is present in the measurements, b. The solution can be thought of as the projection of b into the range of A, as seen in Figure 4.2. The cost criterion is appealing to physical intuition, since it requires the solution to account for all of the measurements the system could have produced. The solution only requires the basic data (system matrix and measurements), and the complexity of the solution is the standard of comparison. It is easy see why the least squares criterion is popular, but it is not without

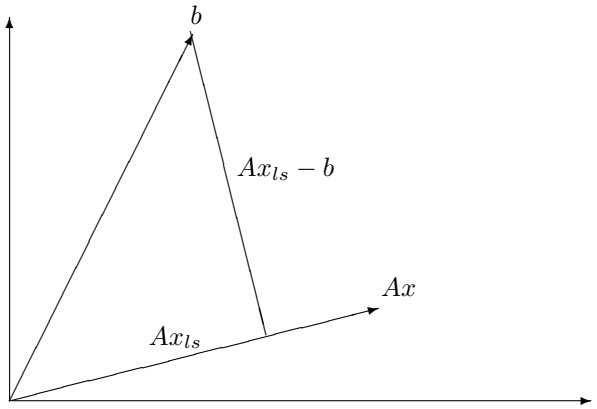


Figure 4.2: Geometric Interpretation of Least Squares Solution

its problems. The choice of independent variables, which will be shown below, and scaling problems, which are outlined in [49], are few of the well known problems with least squares.

As an example let's look at linear least squares for the points (0,1), (1,2), and (2,3). We need to find the coefficients m, and b for the line. We construct our matrices

$$\begin{aligned} Y &= \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}^T \\ \Phi &= \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 2 \end{bmatrix}^T \\ A &= \begin{bmatrix} b & m \end{bmatrix}^T. \end{aligned}$$

As a second example, consider fitting e^{ax} to (0,1), (1,.5), and (2,.25). To separate the coefficient, a, from the variable, x we take the natural log of $y_i = e^{ax_i}$ to obtain $\ln(y_i) = ax$. We can proceed as before now.

As a third example we will consider the second problem where we have noise (random errors) in the measurements. These three examples are coded into Matlab by

Listing 4.1: Matlab code for examples

```
Y=[1;2;3];
Ye=log([1;.5;.25]);
Yee=log([1;.5;.25]+.3*rand(3,1));
X=[0;1;2];
One=ones(3,1);
Phi=[One,X];
```

```
A=Phi\Y
norm(Y-Phi*A)

Ae=X\Ye
Aee=X\Yee

Xf=0:.05:2;

Yfe=exp(Ae.*Xf);
Yfee=exp(Aee.*Xf);

p1=[-.1,2.1];
p2=[-.1,3.1];
q=[0,0];

subplot(3,1,1)
plot(X,Y,'w*',X,Phi*A,'w-',p1,q,'w-',q,p2,'w-')
axis([p1,p2])
subplot(3,1,2)
plot(X,exp(Ye),'w*',Xf,Yfe,'w-',p1,q,'w-',q,p2,'w-')
axis([p1,p2])
subplot(3,1,3)
plot(X,exp(Yee),'w*',Xf,Yfee,'w-',p1,q,'w-',q,p2,'w-')
axis([p1,p2])
```

and we get the output below and in Fig 4.3.

```
A =
    1.0000
    1.0000
ans =
    0
Ae =
   -0.6931
Aee =
   -0.4650
```

Homework 8.6: 1,3

4.2 Total Least Squares

$$\min_x \|\bar{A}x - \bar{b}\|$$
$$s.t.$$

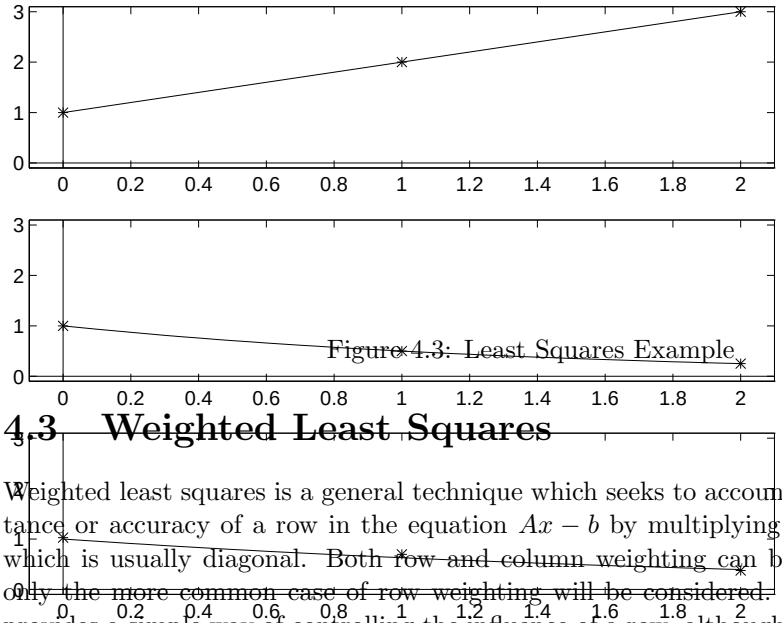


Figure 4.3: Least Squares Example

4.3 Weighted Least Squares

Weighted least squares is a general technique which seeks to account for the relative importance or accuracy of a row in the equation $Ax - b$ by multiplying by a weighting matrix, which is usually diagonal. Both row and column weighting can be done [44, 62], though only the more common case of row weighting will be considered. Weighted least squares provides a simple way of controlling the influence of a row, although it unfortunately suffers from ill conditioning. Some major uses of weighted least squares are iterative improvement of a least squares solution [2, 3, 5, 8, 36], electrical networks, and finite elements [64]. The weighted least squares problem can be stated as

$$\min_x \left\| W^{\frac{-1}{2}} (Ax - b) \right\|^2,$$

with solution

$$\begin{aligned} A^T W^{-1} A x &= A^T W^{-1} b \\ x &= \left(A^T W^{-1} A \right)^{-1} A^T W^{-1} b. \end{aligned}$$

The main source of ill conditioning is the matrix W, which is by its very nature designed to have a large spread in its Eigenvalues. Vavasis [66] claims that this ill condition causes usual techniques for least squares type problems to yield highly inaccurate answers. Given

the ill conditioning, it is reasonable to ask if x ever can become infinite due to W. Both Stewart [63] and Todd [65] were able to establish independently that for all positive-definite real diagonal matrices, W, the following supremums are finite:

1. $\sup\left\{\|(A^TW^{-1}A)^{-1}A^TW^{-1}\|\right\}$
2. $\sup\left\{\|A(A^TW^{-1}A)^{-1}A^TW^{-1}\|\right\}$

Since the supremums are finite, x must be finite. Another question to answer is if a stable method for solution exists. No method has been shown to be stable using the usual backward error analysis technique, but [41] shows one exists if stability is defined as

$$\|x_{True} - x_{Est}\| \leq \epsilon \cdot f(A) \cdot \|b\|,$$

where ϵ is machine precision and $f(A)$ is some function of A that does not depend on W. It is important to note again that a special definition of stability is needed for this problem, due to the conditioning problem in W. The basic solution of [41] can then be expressed as

1	QR factor (with pivoting) $A^TW^{\frac{-1}{2}}$:
	$A^TW^{\frac{-1}{2}} = Q_1R_1P$
2	Reduced QR factor (without pivoting) R_1^T:
	$R_1^T = Q_{2,1}R_{2,1}$
3	Solve by back substitution for z:
	$R_{2,1}z = Q_{2,1}^TPW^{\frac{-1}{2}}b$
4	Multiply:
	$x = Q_1z$

The first QR factorization is to provide stabilization. The second QR factorization is to solve the least squares problem. The net effect is to factor $A^TW^{\frac{-1}{2}}$ into $Q_1R_{2,1}^TQ_{2,1}^TP$, which is essentially a complete orthogonal decomposition [35].

Alternately, the solution can be stated in terms of the equilibrium system

$$\begin{bmatrix} W & A \\ A^T & 0 \end{bmatrix} \begin{bmatrix} y \\ x \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix},$$

and then found by any method desired, such as the QR factorization. While this has not been shown to be stable, the alternate formulation shows that the weighted least squares problem is essentially a special case of the generalized least squares problem [35, 54, 55, 56, 43, 1].

Finally, the weights can be difficult to select if they do not arise naturally. For instance, in simple DC electrical networks the weights can be considered as resistances, and the A matrix defines the adjacencies, b gives the voltage sources, and x is the desired node voltages. Such cases naturally give rise to the weights. What happens when this does not happen? The engineer is left trying to apply heuristics to select a weighting matrix. Given the drawbacks, weighted least squares will not be considered further.

4.4 Constrained Least Squares

Often constraints naturally arise in problems. A fitting function could have prescribed values, a physical system could have limits on its operation, or a solution in a particular

set could be desired. Probably the most basic constrained problem is the least squares Quadratic Inequality problem described in [35]. The problem can be stated as

$$\min_x \|Ax - b\| \qquad \text{subject to} \qquad \|Bx - d\| \leq \alpha.$$

Often $d = 0$ and B is nonsingular, though this is not required. The problem can be solved by the method of Lagrange multipliers, which has the nice bonus of having connections to Tikhonov's method, which is dealt with in Section 4.6. The problem becomes

$$\min_{x,\lambda} \|Ax - b\|^2 + \lambda(\|Bx - d\|^2 - \alpha^2).$$

Taking derivative and setting equal to zero the solution is

$$x = (A^T A + \lambda B^T B)^{-1}(A^T b + \lambda B^T d) \tag{4.2}$$

$$g(\lambda) = \|B(A^T A + \lambda B^T B)^{-1}(A^T b + \lambda B^T d) - d\|^2 - \alpha^2 = 0. \tag{4.3}$$

Equation 4.2 gives the one parameter family of solutions for the problem. When the value of the Lagrange multiplier, λ, is known the unique solution is specified. Equation 4.3 is called the secular equation in [35] and this designation will be used throughout my writings. The purpose of Equation 4.3 is to find the value of the Lagrange multiplier, λ. The multiplier is found by any root finding method desired, though typically Newton's method or bisection is used. The general procedure of finding a solution to a secular equation by root finding is used in many methods.

A particular case of the least squares quadratic inequality, minimization over a sphere, is of particular importance and has been studied extensively, see [4, 19, 20, 21, 22, 25, 27, 35, 53, 61, 62]. The minimization of a least squares problem over a sphere has strong connections to robustness, and is strongly connected to the Ridge Regression and Cross-validation problems, which will be discussed in Section 4.5. The basic problem is

$$\min_{\|x\| \leq \alpha} \|Ax - b\|.$$

Following the procedure outlined above, the solution is found to be

$$x = \begin{cases} A^\dagger b & \text{if} \quad \|A^\dagger b\| \leq \alpha \\ (A^T A + \lambda I)^{-1} A^T b & \text{else} \end{cases}.$$

This special case covers the solution being confined to a particular set, and thus forces the solution to stay bounded. Since the solution is always bounded to a reasonable size it prevents one problem associated with lack of robustness, namely solutions being unstable and growing without bound. A major problem with this is how to know a priori the size of the true $\|x\|$. An error on the guess of the size of x can cause a reduction in the signal strength (as $\|x\|$ is forced to be smaller than the guess). Another problem is that λ can in general be quite large, but experience shows that a small value of λ is more desirable as large values tend to remove fine details (usually carried in the singular vectors associated with smaller singular values) first. The solution obtained from large values of λ tend to bear little resemblance to the true solution in all but the major details. This is a key area of my research, how to get a good value for the regression parameter, λ so it neither becomes unstable nor loses data.

4.5 Ridge Regression

The Ridge Regression problem is an important special case of constrained least squares. Ridge Regression can also be considered a special case of Tikhonov regularization, which is covered in Section 4.6. Golub and Van Loan [35] describe the RR problem as

$$\min_x \|Ax - b\|^2 + \lambda \|x\|^2 \tag{4.4}$$

with the criterion for picking $\lambda > 0$ such as $\|x(\lambda)\| \leq \alpha$, i.e. minimization over a sphere as discussed in Section 4.4.

Other techniques for selecting the ridge parameter exist, such as the generalized cross-validation function [34, 23]. The cross-validation function seeks to reduce the dependence of the solution on any one experiment, and thus increases the robustness of the problem, as seen in [35]. The cost function for the cross-validation problem is given by

$$C(\lambda) = \frac{1}{m} \sum_{k=1}^{m} w_k \left[\frac{\bar{b}_k - \sum_{j=1}^{r} u_{kj} \bar{b}_j \left(\frac{\sigma_j^2}{\sigma_j^2 + \lambda} \right)}{1 - \sum_{j=1}^{r} u_{kj}^2 \left(\frac{\sigma_j^2}{\sigma_j^2 + \lambda} \right)} \right]^2$$

with

- w_k a weight on the importance of the k^{th} row (or experiment),
- the SVD of A given by $U \Sigma V^T$,
- u_{jk} is the j, k^{th} element of U,
- σ_j is the j^{th} diagonal element of the diagonal matrix Σ,
- and $\bar{b} = U^T b$.

Details on the minimization of this cost function are discussed in [34]. The case of the Ridge Regression problem with the cross-validation function used to select λ is often called the cross-validation problem. No matter how the value of λ is selected, the expression for $x(\lambda)$ is given by $x(\lambda) = (A^T A + \lambda I)^{-1} A^T b$. It can be easily seen that each component of the RR solution is smaller than the corresponding component of the LS problem, and thus the robustness is gained at the cost of signal strength (or information content).

4.6 Tikhonov

The Tikhonov problem can be expressed as

$$\min_x \|Ax - b\|^2 + \lambda \|Lx\|^2. \tag{4.5}$$

Note L can be indefinite. Two parameters can be chosen by the designer to select the desired solution. The first parameter is L, which is used to specify conditions on x. For

instance, a solution with a small norm could be desired, which would correspond to picking L to be the identity matrix. Alternately, a solution with a small derivative could be desired, which corresponds to picking L to be the discrete approximation of the derivative operator. Similar to weighted least squares, weights could be placed on particular portions of x to limit their sizes.

The second parameter is λ. Rather than chose λ directly, as is done for L, a requirement for λ in terms of the rest of the problem is usually chosen. For instance, the problem of minimizing a solution on a sphere used $\|x(\lambda)\| \leq \alpha$. What requirement should be used becomes the central discussion of Tikhonov regularization.

- In [40], it was shown that a non-zero λ produces smaller error on average.
- The discrepancy principle [50] assumes the true system has been corrupted by noise and uses the standard deviation of the noise, to find λ.
- The L-curve [38] assumes the system is corrupted by noise but does not require as much information on the noise properties as the discrepancy principle.
- Bounded variations for piecewise continuous functions with at most countably many discontinuities, are handled in [51].
- Generalized cross-validation [34], mentioned earlier tries to minimize the dependence on any one trial.
- Residual and singular value plots have also been suggested [60] to pick λ.
- Minimizing the lengths of confidence intervals [58] has been done.
- Even parameter choices for iterative solution methods exist [42].
- Most interesting though are the methods that attempt to minimize the distance to the true solution, such as [24, 30, 37, 59, 52].

In particular, consider the most recent method as covered in [52]. Let the SVD of A be $U\Sigma V^T$ and define $\beta = U^T b$. The Tikhonov solution to $Ax \approx b$, with $L = I$ is

$$\begin{aligned} x_{tik} &= V(\Sigma^T\Sigma + \lambda I)^{-1}\Sigma^T\beta \\ &= \sum_{i=1}^{n} \frac{\sigma_i\beta_i}{\sigma_i^2 + \lambda} v_i. \end{aligned}$$

The true system with noise ϵ can be expressed as

$$\begin{aligned} x_{true} &= V\Sigma^\dagger(\beta - \epsilon) \\ &= \sum_{i=1}^{n} \frac{\beta_i - \epsilon_i}{\sigma_i} v_i. \end{aligned}$$

Minimizing the distance between these two values gives the condition for λ. To compute the function exactly requires the knowledge of ϵ, which is not known. An approximation

can be made if the system satisfies the discrete Picard condition (the data values $\beta_i - \epsilon_i$ goes to zero faster than the singular values) and β_i is a true value plus noise. With these assumptions and the standard deviation, s, of the noise, the root of the function

$$\sum_{i=1}^{n} \frac{\beta_i^2 \lambda}{(\sigma_i^2 + \lambda)^3} - \sum_{i=1}^{k-1} \frac{s^2}{(\sigma_i^2 + \lambda)^2} - \sum_{i=k}^{n} \frac{\beta_i^2}{(\sigma_i^2 + \lambda)^2}$$

gives the value of λ. While the value obtained is an approximation, O'Leary [52] shows that the resulting x value, $x_{tik}(\lambda)$ is close to the true (using the not approximated value of λ_{true}) value $x_{tik}(\lambda_{true})$, and that in particular

$$\frac{\|x_{tik}(\lambda_{true}) - x_{tik}(\lambda)\|}{\|x_{tik}(\lambda_{true})\|} \leq \frac{|\lambda_{true} - \lambda|}{\sigma_n^2 + \lambda}.$$

Additionally, as the standard deviation of the noise, s, goes to zero, $x_{tik}(\lambda)$ goes to x_{true}. A nice result. An alternate method of choosing λ is presented at the same time using

$$x_{alt} \quad = \quad \sum_{i=1}^{n} \frac{\beta_i}{\sigma_i + \lambda} v_i.$$

The choice was suggested for Hermitian positive definite matrices [26], convolution problems with reordering [18], and some additional cases [39, 15]. The alternate method proceeds similarly with a small alteration in the function, whose root must be found. The fact that two methods are suggested, indicates that no one best method exists. Both perform well however, and demonstrate robustness, which was a major goal.

Tikhonov regularization, works by damping out the terms that correspond to the smaller singular values, see for example [33]. This can be thought of geometrically as finding a worse model within some bounded region from the original model and solving the LS problem on this new model. Tikhonov regularization generates robust solutions, but the wealth of techniques to select λ shows that there is no obvious best technique. A drawback to Tikhonov regularization is thus also one of its strengths, namely the wide variety of techniques to pick λ. The criterion for picking λ is really dependent on the solution desired, for instance, the choice of the solution lying in a ball is usually done to fulfill a heuristic requirement for boundedness. In the end this method usually ends up being more based on the skill and experience of the engineer who sets up the problem. This is exactly how the problem is treated in [33, 52]. The damping of terms corresponding to smaller singular values, essentially means that data and thus accuracy will be lost. Most of the accuracy loss is due to the "waterbed effect", in that accuracy and robustness are competing goals, so advances in one area causes losses in another. Such competing goals thus are not so much a problem but rather a design decision based on the problem requirements. The assumptions are another matter. By adding an implicit heuristic element, the problem de facto includes the "gut feel" of the designer. While this may seem appealing, it is not rigorous, and does not allow for confidence in the final result. A good guess will give a good result, a bad one a bad result, but there is no way of assessing the guesses. The desire for a more philosophically pleasing and mathematically rigorous method for posing robust problems led to the development of the min max problem.

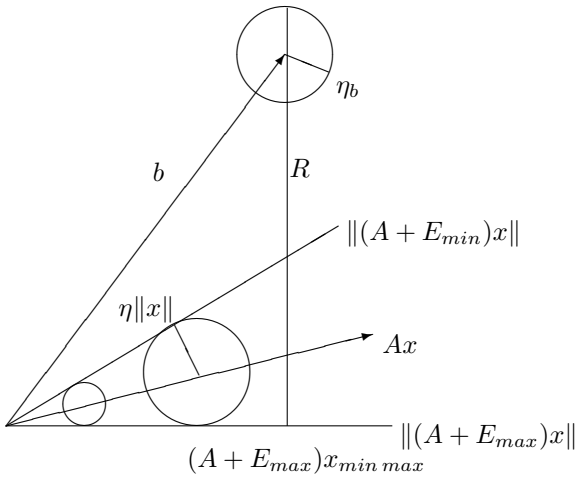


Figure 4.4: Geometric Interpretation of Min Max Solution

4.7 Min Max

The min max problem was proposed and solved separately in [11] by secular equation techniques and in [32] by Linear Matrix Inequality techniques. This section will concentrate on the secular equation formulation. The LMI techniques are discussed in Section 4.10

Simply stated the min max problem seeks to find the worst model in a bounded region, and then solve the problem based on this worst case scenario. Mathematically it is written as

$$\min_x \max_{\begin{bmatrix}\|E_A\| \le \eta \\ \|E_b\| \le \eta_b\end{bmatrix}} \|(A+E_A)x - (b+E_b)\|. \tag{4.6}$$

This problem can be shown to be equivalent to solving a problem with similar form to the Tikhonov problem, see [11]. Equation 4.6 can be interpreted geometrically by Figure 4.4. The maximization forms the hyperspheres around A and b. The cone around A is formed by varying the size of x. The solution, x, and the residual, R, are found by connecting the furthest points on the hyperspheres. The maximization restricts the problem to the lower line of the cone. The minimization selects the point on the lower cone such that the line segment from the furthest point on the hypersphere around b to the lower cone is perpendicular to the lower cone. The norm used in [11] is the 2-norm, though [67] extends it to other norms. The min max problem becomes

$$\min_x(\|Ax-b\| + \eta\|x\| + \eta_b), \tag{4.7}$$

which differs from the typical Tikhonov problem in that the norms are not squared. As

opposed to the Tikhonov problem, the term η now has a physical intuition also, that being the amount of uncertainty in the matrix. Computing the min max solution takes longer than computing the solution to a Tikhonov problem if a simple choice of regression parameter is chosen for the Tikhonov problem, so it is logical to ask why one would want to spend the extra operations to do so. The simple answer is that the two problems can give arbitrary differences, which we will examine in Section 4.8.

In the form of Equation 4.7 it is easy to see that the problem is continuous but non-smooth, since it is non-differentiable whenever $x = 0$ or when $Ax = b$. The solution to Equation 4.7 and thus Equation 4.6, is summarized in Table 4.1. For Table 4.1, let the SVD of A be given by

$$A = \begin{bmatrix} U_1 U_2 \end{bmatrix} \begin{bmatrix} \Sigma \\ 0 \end{bmatrix} V^T.$$

Partition the vector $U^T b$ into

$$\begin{bmatrix} U_1 U_2 \end{bmatrix}^T b = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix},$$

and introduce the secular equation

$$g(\psi) = b_1^T (\Sigma^2 - \eta^2 I)(\Sigma^2 + \psi I)^{-2} b_1 - \frac{\eta^2}{\psi^2} \|b_2\|^2,$$

which has a unique positive root, denoted $\bar{\psi}$ under the conditions noted in Table 4.1. Finally define

$$\tau_1 = \frac{\|\Sigma^{-1} b_1\|}{\|\Sigma^{-2} b_1\|} \qquad \text{and} \qquad \tau_2 = \frac{\|A^T b\|}{\|b\|}.$$

The solution is thus given below. Notice that the least squares solution, $A^\dagger b$, is the min max solution under special conditions. In one case a scaled family of the least squares solution solves the problem. In general though the solution is given by finding the unique root of the secular equation, $g(\psi)$ in the positive quadrant. When η is large the solution is zero.

4.8 Comparison of Min Max and Tikhonov

At this point, it is reasonable to ask if there is a similar, but simpler way to solve the problem, which exhibits the desired behavior of min max that can be solved instead of the min max methodology of [11, 67, 32]. One candidate solution that has been suggested is Tikhonov regulation. It has a large body of literature, such as [33, 52], and a closed form solution. Start by noting that a reasonable choice for the parameter λ in the Tikhonov problem is to chose it to be equal to the square of the uncertainty, since all the other terms are squared and this will account for the size of the uncertainty. In this case the model has a closed form solution which is given by

$$\hat{x} = \left(A^T A + \eta^2 I \right)^{-1} A^T b. \tag{4.8}$$

	$b \in \mathcal{R}(A)$	$b \notin \mathcal{R}(A)$
$\eta \geq \tau_2$	0	0
$\tau_1 < \eta < \tau_2$	$x = (A^T A + \bar{\psi} I)^{-1} A^T b$	$x = (A^T A + \bar{\psi} I)^{-1} A^T b$
$\eta \leq \tau_1$	$x = A^{\dagger} b$	$x = (A^T A + \bar{\psi} I)^{-1} A^T b$
$\eta = \tau_1 = \tau_2$	$x = \beta A^{\dagger} b$ with $0 \leq \beta \leq 1$	$x = (A^T A + \bar{\psi} I)^{-1} A^T b$

Table 4.1: Min Max Solution

Note that this is clearly a regularized estimator, with the regularization parameter given by the bound in the error. Note also that for the min max problem that if $Ax \neq b$ and $x \neq 0$ then the min max problem also has a solution with a similar form given by

$$\hat{x} = \left(A^T A + \alpha I\right)^{-1} A^T b \tag{4.9}$$

$$\alpha = \eta \frac{\|Ax - b\|}{\|x\|}. \tag{4.10}$$

The min max problem is also a regularized solution, with the regularization parameter given by α. Since α is dependent on unknown values it must be calculated, which is usually done by a secular equation. The logical question is, "Why not use the Tikhonov cost function, which has the closed form solution?" To answer this it must be seen if the Tikhonov problem's regularization parameter can be arbitrarily larger or smaller. If the Tikhonov problem's parameter can be arbitrarily larger, then the solution can be over regularized and thus valuable information can be lost. If the Tikhonov parameter can be arbitrarily smaller, then the solution can be under regularized and thus the solution might not be robust. Thus to compare the two, examine the ratio of the min max problem's regularization parameter, α, to the Tikhonov problem's regularization parameter, η^2. Doing so, obtain

$$\frac{\alpha}{\eta^2} = \frac{\|Ax_{mm} - b\|}{\eta \|x_{mm}\|}. \tag{4.11}$$

4.8.1 Over-Regularization

First, see if the Tikhonov problem can be over regularized, which is the more dangerous problem. This corresponds to the ratio being arbitrarily small. Note that $\|Ax_{mm} - b\| \leq \|b\|$ at the solution, by noting the cost at the solution must be less than the cost at the point

$x = 0$. Thus,

$$\frac{\alpha}{\eta^2} \leq \frac{\|b\|}{\eta \|x_{mm}\|}. \tag{4.12}$$

It is clearly possible to pick A and b such that $\eta\|x_{mm}\| \gg \|b\|$. For example consider the following simple system,

$$A = \begin{bmatrix} 0.2 \\ 0 \end{bmatrix} \qquad b = \begin{bmatrix} 5 \\ 1 \end{bmatrix} \qquad \eta = 0.1. \tag{4.13}$$

For this simple system the min max problem has a solution of $x_{mm} = 22.11$ while the modified problem has a solution of $x_T = 20$. We note that for this problem the Tikhonov regularization parameter is twice as large as the min max problem. Clearly the over-regularization has also yielded a loss of information that is not warranted by the problem. We note that while this simple example does not show an arbitrarily large ratio difference, since it is used only as a numerical motivation. To see the arbitrary difference consider the following for $\delta \ll 1$,

$$A = \begin{bmatrix} \delta \\ 0 \end{bmatrix} \qquad b = \begin{bmatrix} \frac{1}{\delta} \\ \delta \end{bmatrix} \qquad \eta = \frac{\delta}{2}. \tag{4.14}$$

For this system note that the least squares (LS) solution is given by $x_{LS} = \frac{1}{\delta^2}$, and the min max system is $x_{mm} = \frac{1}{\delta^2} - \frac{1}{\delta\sqrt{3}}$. Note that since $\delta \ll 1$, the min max estimate is extremely close to the LS solution. The Tikhonov problem solution is given by $x_T = \frac{4}{5\delta^2}$, which is easily seen to be arbitrarily far from the desired solution, since for $\delta \ll 1$ the two candidate solutions differ by almost 20% of an arbitrarily large number. Moreover, the ratio of regularization parameters is approximately given by the arbitrarily small number,

$$\frac{\alpha}{\eta^2} \approx \frac{4}{\sqrt{3}}\delta^2. \tag{4.15}$$

4.8.2 Under-Regularization

The second area to be considered is if the Tikhonov problem can be under-regularized. This corresponds to the ratio of α over η^2 being arbitrarily large. Note that $\|Ax_{mm} - b\| \geq \|P_{A^\perp} b\|$, thus

$$\frac{\alpha}{\eta^2} \geq \frac{\|P_{A^\perp} b\|}{\eta \|x_{mm}\|}. \tag{4.16}$$

It is clearly possible to pick A and b such that $\|x_{mm}\| \ll \|P_{A^\perp} b\|$. For example consider the following simple system,

$$A = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \qquad b = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \qquad \eta = 1. \tag{4.17}$$

Note that since the perturbation is as large as the norm of the A matrix, $x_{mm} = 0$, which corresponds to $\alpha \to \infty$. This is intuitively pleasing, as it confirms the belief that no valid information exists for a system with uncertainty as large as the system. Note also that $x_{LS} = 1$. Now the Tikhonov problem has the solution $x_T = \frac{1}{2}$. Not only is this clearly too optimistic an answer, it is also unrealistic. The ratio is infinite and thus arbitrarily large, as was desired to be shown. Thus while the Tikhonov problem has nice properties for calculation, its estimator can be arbitrarily different than the min max problem. Additionally, the Tikhonov problem does not correspond to physical intuition as can be seen in the last example above. The min max problem can thus not be altered to an apparently similar problem and solved for that system.

4.9 Non-Degenerate Min Min

The non-degenerate min min problem was presented in [12] for the case of the 2-norm and extended to other norms in [67]. The essential idea is to assume, similar to total least squares, that the actual system $A + E_A$ and $b + E_b$ is such that $b + E_b$ is as close to being in the subspace defined by $A + E_A$ as possible. The residual is then minimized over all choices x. The problem is thus expressed as

$$\min_x \min_{\begin{bmatrix} \|E\| \leq \eta \\ \|E_b\| \leq \eta_b \end{bmatrix}} \|(A + E_A)x - (b + E_b)\|. \tag{4.18}$$

The geometric view is very similar to the min max problem and is provided in Figure 4.5. The cone around A and the ball around b are the same as before (i.e.: all possible values for the problem). The min min problem is thus to find the smallest distance from the ball to the cone, which is shown in the figure. Note that it is possible for the ball and cone to have points in common, this is the degenerate case, and is covered in Chapter **??**. This section will cover the non-degenerate condition. Two main issues are:

1. find a computable condition for checking degeneracy,
2. find a secular equation and region to find the solution.

First, find a computable condition for degeneracy. For the problem to be non-degenerate the residual must be greater than the possible perturbation. In equation form this is

$$\eta\|x\| < \|Ax - b\|. \tag{4.19}$$

This equation depends on the solution, x, so an equation with only A, b, and η is desired. By squaring the degeneracy condition, Equation 4.19, the condition becomes

$$x^T(A^TA - \eta^2 I)x - 2x^TA^Tb + b^Tb > 0. \tag{4.20}$$

For this to hold for all x, the minimum value of the function must be greater than zero. The function must have a finite minimum, and that minimum must be positive. To have a

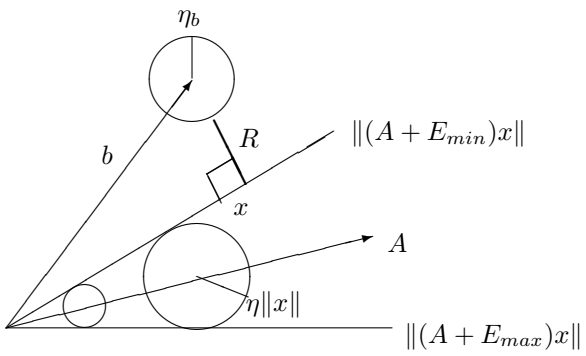


Figure 4.5: Geometric Interpretation of Min Min Solution

finite minimum, the following must hold

$$A^T A - \eta^2 I > 0,$$

or defining the minimum singular value of A to be σ_{min},

$$\sigma_{min} > \eta. \tag{4.21}$$

Noting that in practice $\eta > 0$ ($\eta = 0$ is least squares), this requires that A is full rank, which is assumed from now on. Provided Equation 4.21 holds, the minimum value of Equation 4.20 is

$$b^T \left[I - A(A^T A - \eta^2 I)^{-1} A^T \right] b > 0. \tag{4.22}$$

The problem is non-degenerate if A is full rank, and both Equation 4.21 and Equation 4.22 hold.

The computable condition is needed to see if the non-degenerate case applies (if it doesn't the non-degenerate case of Chapter **??** holds) and is useful in the proof. The proof of the solution is too lengthy to present here, readers are referred to [12] for a full treatment. Assume the problem is non-degenerate and let the SVD of A be

$$A = \begin{bmatrix} U_1 & U_2 \end{bmatrix} \begin{bmatrix} \Sigma \\ 0 \end{bmatrix} V^T,$$

with smallest singular value σ_n and corresponding left singular vector u_n. Define

$$\begin{bmatrix} b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} U_1 & U_2 \end{bmatrix}^T b.$$

Then for the secular equation,

$$g(\psi) = b_1^T (\Sigma^2 - \eta^2 I)(\Sigma^2 - \psi I)^{-2} b_1 - \frac{\eta^2}{\psi^2} \|b_2\|^2,$$

find the unique root of $g(\psi)$ in the interval (η^2, σ_n^2) and if it exists call it $\hat{\psi}$, otherwise let $\hat{\psi} = \sigma_n^2$. If $\hat{\psi} < \sigma_n^2$ then the solution is

$$x = (A^T A - \hat{\psi} I)^{-1} A^T b,$$

else there are two solutions

$$x = V \begin{bmatrix} (\bar{\Sigma}^2 - \sigma_n^2 I)^{-1} \bar{\Sigma} \bar{b}_1 \\ \pm \frac{\sigma_n}{\sqrt{\sigma_n^2 - \eta^2}} \sqrt{-\bar{g}(\sigma_n^2)} \end{bmatrix}$$

with

$$\begin{aligned} \bar{g}(\psi) &= g(\psi) - (u_n^T b)^2 \frac{\sigma_n^2 - \eta^2}{(\sigma_n^2 - \psi)^2} \\ \Sigma &= \begin{bmatrix} \bar{\Sigma} & 0 \\ 0 & \sigma_n \end{bmatrix} \\ b_1 &= \begin{bmatrix} \bar{b}_1 \\ b_{1,n} \end{bmatrix} = \begin{bmatrix} \bar{b}_1 \\ 0 \end{bmatrix}. \end{aligned}$$

My dissertation completed the analysis of this problem by solving the degenerate case, which turns out to be the more general situation.

4.10 LMI Techniques

Of all the techniques presented the Linear Matrix Inequality (LMI) techniques are the most flexible. Most of the techniques both currently used can be solved using LMI techniques. The Backward Error method is an example of a problem that does not fit into the LMI framework, due to its rational cost function. The principle concern of this section is to consider the LMI techniques that are similar to what is covered in my dissertation. In [9, 10, 31, 32, 45, 46, 47], the LMI methodology for solving the min max problem with and without structure were covered. This section will cover two principle areas of [32], that being the structured and unstructured case.

The unstructured perturbations are identical to the min max case. The problem is defined as

$$\min_x \max_{\|E E_b\|_F \leq 1} \|(A + E_A)x - (b + E_b)\|.$$

Note that the bound is 1 since the problem can always be normalized to this by dividing A and b by any other bound thus yielding a problem of the form above. The problem can be reformulated as a Second-Order Cone Programming (SOCP) problem of the form

$$\begin{aligned} &\min \lambda \\ &s.t. \quad \begin{bmatrix} \|Ax - b\| \leq \lambda - \tau \\ \left\| \begin{bmatrix} x \\ 1 \end{bmatrix} \right\| \leq \tau \end{bmatrix}. \end{aligned}$$

Define ψ to be $\frac{(\lambda-\tau)}{\tau}$. The solution can then be shown to be

$$x = \left\{ \begin{bmatrix} (A^T A + \psi I)^{-1} A^T b & \text{if } \psi > 0 \\ A^\dagger b & \text{else} \end{bmatrix} \right.$$

with λ and τ are the unique optimal points for the system. The parameter ψ is the same as was found in the min max problem by secular equation techniques.

At this point it is reasonable to ask why further work should be done. The basic reason is speed. Each iteration of a SOCP is basically $O((m+n)n^2)$ and [32] asserts that the number of iterations is almost constant and independent of the problem size, resulting in a reasonably sized constant multiplying the n^3. In contrast, solving a secular equation can be done in iterations that are n^2 and then the overall solution takes n^3 but has a smaller constant since it mostly comes from the calculation of the SVD (the "light" version of the SVD can be used further saving time). In [32], it is noted that both have the same order of complexity, which is true, but order is not the only determiner, the constant that is ignored when reporting order can greatly influence practical speed. The speed advantage of secular techniques is noted in [32], thus secular equation techniques have a slight advantage over SOCP. In [46] it is asserted that the secular equation technique is simpler and that LMI techniques only have advantage over secular techniques on robust regression problems when additional constraints need to be applied. Noting that SOCP problems can be solved faster than SDP (semi-definite programming) problems, the use of secular equation techniques becomes more apparent. On a different line, developing algorithms to solve problems in different ways is a valuable goal in and of itself, that yields benefits in many areas from theoretical understanding to programming implementations. The nice features of the LMI techniques thus do not rule out the other solution techniques, rather they complement each other and serve to give a more complete understanding.

The second area to cover is the structured perturbations problem. The structured perturbations can approximate the multi-column, multi-row, and general min max problems developed in my dissertation and some cases not expressible in one of the proposed techniques. Note that the structured case can approximate but not directly solve the cases covered in my dissertation. The structured perturbations are of the form

$$\begin{aligned} E(\delta_1, \ldots, \delta_p) &= \sum_{k=1}^{p} \delta_k E_k \\ E_b(\delta_1, \ldots, \delta_p) &= \sum_{k=1}^{p} \delta_k E_{b,k} \end{aligned}$$

where p is the number of basic perturbations, and E_k is a fixed basic perturbation on A and $E_{b,k}$ is a fixed basic perturbation on b. The problem with trying to directly solve the

multi-column case (for instance) is that the basic perturbations would need to be,

$$\begin{aligned} E_1 &= \frac{Ax-b}{\|Ax-b\|}\frac{\begin{bmatrix} x_1^T & 0 & \dots & 0 \end{bmatrix}}{\|x_1\|} \\ &\vdots \\ E_p &= \frac{Ax-b}{\|Ax-b\|}\frac{\begin{bmatrix} 0 & \dots & 0 & x_p^T \end{bmatrix}}{\|x_p\|}. \end{aligned}$$

This requires E_k to be dependent on x and thus the LMI to solve the problem that is presented, is no longer linear. The multiple column case is used in [32] as a motivation for the linear-fractional case, which is shown to be NP-hard in the general case and for which an upper bound for the worst case residual was obtained. As special cases, the multiple column, multiple row, and general block perturbation cases are not NP-hard, and allow for the results in my dissertation. The three cases could be approximated with the structured problem (as opposed to the linear-fractional) by examining a series of problems with the values of E_k based off the previous problem's solution, starting with say $x = A^{\dagger}b$, though it is not obvious if this would converge. Optionally the problem could be approximated by some other column (row, or general also) structure, though it will not necessarily generate the same one found in my dissertation. Undoubtedly an LMI formulation can be found to solve the same problem, the key point is that the structured formulation as outlined in [32] will not. The formulation in [32] does permit important structures, such as Toeplitz, that cannot be handled in the formulations of my dissertation. Each formulation is useful and has a place, the needs of each individual problem under consideration determine which to use. Returning to the structured problem, first define the following:

$$\begin{aligned} M &= \begin{bmatrix} E_1x - E_{b,1} & \dots & E_px - E_{b,p} \end{bmatrix} \\ F &= M^TM \\ g &= M^T(Ax-b) \\ h &= \|Ax-b\|. \end{aligned}$$

The problem can be written as

$$\min_x \max_{\|\delta\|\le 1} \begin{bmatrix} 1 \\ \delta \end{bmatrix}^T \begin{bmatrix} h & g^T \\ g & F \end{bmatrix} \begin{bmatrix} 1 \\ \delta \end{bmatrix}.$$

The problem can then be solved by the SDP,

$$\begin{aligned} &\min \lambda \\ &s.t. \quad \begin{bmatrix} \lambda-\tau & 0 & (Ax-b)^T \\ 0 & \tau I & M^T \\ Ax-b & M & I \end{bmatrix} \ge 0. \end{aligned}$$

The SDP formulation, while not as efficient as a SOCP formulation, is still polynomial time, and can be solved by interior point algorithms. The variety of structures that can be handled

or approximated by the structured technique in [32] is tremendous and covers most of the cases of interest. Problems still exist which cannot be directly handled, or which could be solved more efficiently by specialized solvers.

4.11 Simple Comparative Example

The previous discussion includes three general groups of problem formulations that can be used in estimation. The following provides a feel for how these problems operate on a simple example. Consider a simple one dimensional "skyline" image that has been blurred. A "skyline" image is a one dimensional image that looks like a city skyline when graphed, and thus is the most basic image processing example. "Skyline" images involve sharp corners, and it is of key importance to accurately locate these corner transitions. Blurring occurs often in images, for example atmospheric conditions, dust or imperfections in the optics can cause a blurred image. Blurring is usually modelled as a gaussian blur, which is a smoothing filter. The gaussian blur causes greater distortion on the corners, which is exactly where we do not want it. The component of a gaussian blur with standard deviation, σ, in position, (i,j), is given by

$$G_{i,j} = e^{-\left(\frac{i-j}{\sigma}\right)^2}.$$

Going on the presumption that the exact blur that was applied is not known (σ unknown) the exact system cannot be recovered. While the original system cannot be perfectly extracted, some improvement on the blurred image is desirable. The blur is "known" to be small compared to the information so some improvement should be possible. The least squares solution fails completely, yielding a result that is about three orders of magnitude off in the scale and is oscillating badly, see Figure 4.6. Notice that the total least squares solution is better than the least squares solution (only off by an order of magnitude and the oscillations are slower), but still not acceptable. The Tikhonov solution works well due to its increased robustness. All of the methods of my dissertation can be seen to yield very good solutions to the problem.

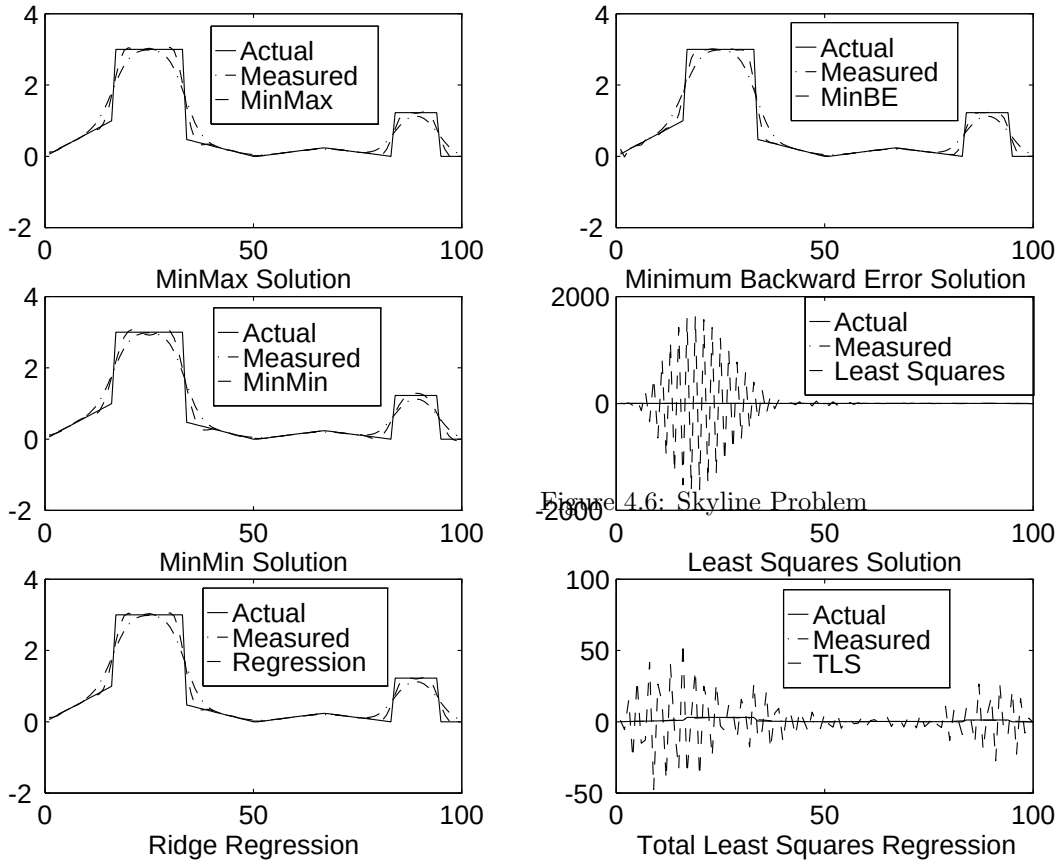


Figure 4.6: Skyline Problem

Chapter 5

Solving Systems of Linear Equations

5.1 Solving Triangular Systems

We will begin our solution of general linear equations by considering how to solve triangular systems. This simpler form of system is particularly easy to solve, and it is easy to make other systems become triangular. In later sections reducing a matrix to triangular form becomes a standard tool.

As a side note, we really don't like reducing to a triangular matrix. It is better to reduce to upper or lower Hessenberg form[1]. Similarly, real numerical systems often avoid diagonalizing and reduce to tridiagonal matrix[2]. The reason is it is often difficult to zero the elements right next to the main diagonal, so we often get most of the way and then finish the job with an iterative method. Starting with an iterative method would be slow. This way we get the best of both systems.

5.1.1 Forward Substitution

Consider a system $Lx = b$, where L is a square lower triangular matrix.

$$\begin{bmatrix} l_{1,1} & 0 & \cdots & 0 \\ l_{2,1} & l_{2,2} & \ddots & \vdots \\ \vdots & \vdots & \ddots & 0 \\ l_{n,1} & l_{n,2} & \cdots & l_{n,n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \tag{5.1}$$

[1] triangular plus one non-zero diagonal immediately past the main diagonal

[2] The only non-zero elements are on the main diagonal and the diagonals immediately next to it

The first row tells us

$$\begin{aligned} l_{1,1}x_1 &= b_1 \\ x_1 &= \frac{b_1}{l_{1,1}} \end{aligned} \tag{5.2}$$

Using the solution of the first line and the equation of the second row we find

$$\begin{aligned} l_{2,1}x_1 + l_{2,2}x_2 &= b_2 \\ x_2 &= \frac{b_2 - l_{2,1}x_1}{l_{2,2}} \\ x_2 &= \frac{b_2 - l_{2,1}\frac{b_1}{l_{1,1}}}{l_{2,2}} \end{aligned} \tag{5.3}$$

In essence we are removing the effect of x_1 on b and then solving just as we did in Eq 5.2. With the knowledge of x_2 we will need to remove its effect on b for subsequent calculations. We can generalize this into a simple procedure. The one thing we have to make sure is that the elements on the main diagonal never become zero[3], as we need to divide by them.

Listing 5.1: SciLab code for forward substitution

```
function x=forward_substitution(l,b)
n=size(l,2); // n is number of columns in l
x=zeros(n,1);

for col = 1:n
  row = col; // consider main diagonal
  if l(row,col)==0 then // lower triangular matrix is
      break;            //singular if zero on diagonal
  end
  x(col)=b(row)/l(row,col); // calculate next component of x
  for row=col+1:n  // remove effect from subsequent rows
    b(row) = b(row) - l(row,col)*x(col);
  end
end
endfunction
```

Listing 5.1 shows SciLab code for a forward substitution function. Note that it has complexity $\frac{n^2}{2}$, which is much better than the n^3 for matrix inversion. In Listing 5.2, is SciLab code that uses the forward substitution function. The exec command causes SciLab to run the contents of the file containing forward_substitution, thus defining it for use on the next line.

[3]There are a lot of ways of explaining why this is from the matrix itself, here are a two. For lower (or upper) triangular systems the determinant is the product of the diagonal elements, thus if one is zero then the determinant is zero and no inverse exists. A zero on the diagonal of a triangular (upper or lower) matrix means there is a zero eigenvalue, and hence the matrix is not invertible.

Listing 5.2: SciLab test code for forward substitution

```
n=5;
l=zeros(n,n);
for row=1:n
  for col = 1:row
    l(row,col)=rand(1);
  end
end
xtrue=rand(n,1)
b=l*xtrue;

exec forward_substitution.sci;
x=forward_substitution(l,b)
```

5.1.2 Backward Substitution

Now consider a system $Ux = b$, where U is a square upper triangular matrix.

$$\begin{bmatrix} u_{1,1} & u_{1,2} & \cdots & u_{1,n} \\ 0 & u_{2,2} & \cdots & u_{2,n} \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & u_{n,n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \tag{5.4}$$

The last row tells us

$$\begin{aligned} u_{n,n}x_n &= b_n \\ x_n &= \frac{b_n}{u_{n,n}} \end{aligned} \tag{5.5}$$

Using the solution of the last line and the equation of the second to last row we find

$$\begin{aligned} u_{n-1,n}x_n + u_{n-1,n-1}x_{n-1} &= b_{n-1} \\ x_{n-1} &= \frac{b_{n-1} - u_{n-1,n}x_n}{u_{n-1,n-1}} \\ x_{n-1} &= \frac{b_{n-1} - u_{n-1,n}\frac{b_n}{u_{n,n}}}{u_{n-1,n-1}} \end{aligned} \tag{5.6}$$

In essence we are removing the effect of x_n on b and then solving just as we did in Eq 5.5. With the knowledge of x_{n-1} we will need to remove its effect on b for subsequent calculations. We can generalize this into a simple procedure. As in the lower triangular matrix of the forward substitution case, the one thing we have to make sure is that the elements on the main diagonal never become zero[4], as we need to divide by them.

[4]There are a lot of ways of explaining why this is from the matrix itself, here are a two. For lower (or upper) triangular systems the determinant is the product of the diagonal elements, thus if one is zero then the determinant is zero and no inverse exists. A zero on the diagonal of a triangular (upper or lower) matrix means there is a zero eigenvalue, and hence the matrix is not invertible.

Listing 5.3: SciLab code for backward substitution

```
function x=backward_substitution(u,b)
n=size(u,2); // n is number of columns
x=zeros(n,1); // predefine size makes the code faster,
              // because no memory allocation calls
for col = n:-1:1  // bottom up for backward
  row = col; // consider main diagonal
  if u(row,col)==0 then // lower triangular matrix is
      break;            //singular if zero on diagonal
  end
  x(col)=b(row)/u(row,col); // calculate next component of x
  for row=1:col-1  // remove effect from subsequent rows
    b(row) = b(row) - u(row,col)*x(col);
  end
end
endfunction
```

Listing 5.3 shows SciLab code for a backward substitution function. Note that it has complexity $\frac{n^2}{2}$, which is much better than the n^3 for matrix inversion. In Listing 5.4, is SciLab code that uses the backward substitution function. The exec command causes SciLab to run the contents of the file containing backward_substitution, thus defining it for use on the next line.

Listing 5.4: SciLab test code for backward substitution

```
n=5;
u=zeros(n,n);
for row=1:n
  for col = row:n
      u(row,col)=rand(1);
  end
end
xtrue=rand(n,1)
b=u*xtrue;

exec backward_substitution.sci;
x=backward_substitution(u,b)
```

5.2 LU Factorization

Since we have ways to solve a system of matrices, $Ax = b$, where A is either upper or lower triangular, it makes sense to ask can we transform any matrix into this form and then solve it. It turns out we can, and it is fairly fast, but not numerically stable[5]. First I provide code

[5]Basically this means it can give very wrong answers under certain circumstances. Don't do this with badly conditioned matrices unless you really know what you are doing.

in Listing 5.5 to find both L (a lower triangular matrix) and U (an upper triangular matrix), such that $LU = A$. Theoretically you could use this and your forward and backward solver to get the solution. In practice we don't do this for two reasons. First, it cannot handle a zero value on the main diagonal (row = col), which will be handled by pivoting. Second, though is that we don't need to know L, we just need U and $L^{-1}b$. This can be seen by the following

$$Ax = b \tag{5.7}$$
$$LUx = b \tag{5.8}$$

Note that this step is seen simply by noting $A = LU$ by construction of L and U.

$$LUx = b \tag{5.9}$$
$$L^{-1}LUx = L^{-1}b \tag{5.10}$$
$$Ux = L^{-1}b \tag{5.11}$$

The next step is the basic algebraic rule of multiplying both sides by the same thing does not change them. The last step combines the definition of an inverse, namely $L^{-1}L = I$, and the property of I that $IU = U$. Thus to find x we need U and $L^{-1}b$. The code for this is in Listing 5.6. Note you could also just write a function that returns U given A, and then pass $\begin{bmatrix} A & b \end{bmatrix}$ instead of just A, and your function will return $\begin{bmatrix} U & L^{-1}b \end{bmatrix}$.

Listing 5.5: Gaussian Elimination (LU factorization)

```
function [l,u]=lu_no_pivot(A)
[m,n]=size(A);
l=eye(m,m);
u=A;
for colPivot=1:n
  // for legibility
  // pivot is on main diagonal (row=col)
  rowPivot=colPivot;
  // make sure the pivot isn't zero, as
  // we must divide by it
  if u(rowPivot,colPivot)==0 then
    break;
  end
  // calculate the next col of l
  for row=rowPivot+1:n
    l(row,colPivot)=u(row,colPivot)/u(rowPivot,colPivot);
  end
  // update u
  for row=rowPivot+1:n
    u(row,colPivot)=0;
    for col=colPivot+1:n
      update=l(row,colPivot)*u(rowPivot,col)
```

```
      u(row,col)=u(row,col)-update;
    end
  end
end
endfunction
```

Listing 5.6: Solving $Ax = b$ using LU factorization

```
function [u,linvb]=lusoler_no_pivot(A,b)
[m,n]=size(A);
l=eye(m,m);
u=A;
linvb=b
for colPivot=1:n
  // pivot is on main diagonal (row=col)
  rowPivot=colPivot;
  // make sure the pivot isn't zero, as
  // we must divide by it
  if u(rowPivot,colPivot)==0 then
    break;
  end
  // calculate the next col of l
  for row=rowPivot+1:n
    l(row,colPivot)=u(row,colPivot)/u(rowPivot,colPivot);
  end
  // update u
  for row=rowPivot+1:n
    u(row,colPivot)=0;
    for col=colPivot+1:n
      update=l(row,colPivot)*u(rowPivot,col)
      u(row,col)=u(row,col)-update;
    end
    update=l(row,colPivot)*linvb(rowPivot);
    linvb(row)=linvb(row)-update;
  end
end
endfunction
```

5.2.1 Partial Pivoting

Listing 5.7: Gaussian Elimination with row pivoting (LU factorization with partial pivoting)

```
function [p,l,u]=lu_partial_pivot(A)
[m,n]=size(A);
l=eye(m,m);
```

```
p=l;
u=A;
for colPivot=1:n
  // find the largest element in column
  // start with main diagonal
  rowPivot=colPivot;
  colmax=abs(u(rowPivot,colPivot));
  for row=colPivot+1:n
    if abs(u(row,colPivot))>colmax then
      rowPivot=row;
      colmax=abs(u(rowPivot,colPivot));
    end
  end
  if rowPivot>colPivot then
    for col=1:n
      temp=u(colPivot,col);
      u(colPivot,col)=u(rowPivot,col);
      u(rowPivot,col)=temp;
      temp=p(colPivot,col);
      p(colPivot,col)=p(rowPivot,col);
      p(rowPivot,col)=temp;
    end
    for col=1
    rowPivot=colPivot;
  end
  // make sure the pivot isn't zero, as
  // we must divide by it
  if u(rowPivot,colPivot)==0 then
    break;
  end
  // calculate the next col of l
  for row=rowPivot+1:n
    l(row,colPivot)=u(row,colPivot)/u(rowPivot,colPivot);
  end
  // update u
  for row=rowPivot+1:n
    u(row,colPivot)=0;
    for col=colPivot+1:n
      update=l(row,colPivot)*u(rowPivot,col)
      u(row,col)=u(row,col)-update;
    end
  end
end
endfunction
```

5.2.2 Cholesky Factorization

For symmetric ($A = A^T$), positive definite ($x^T Ax > 0 \; \forall x \neq 0$) we can perform a special form of the LU factorization called the Cholesky factorization ($A = LL^T$). To see how this works consider the following

$$L = \begin{bmatrix} l_{1,1} & 0 & \cdots & 0 \\ l_{2,1} & l_{2,2} & \ddots & \vdots \\ \vdots & \vdots & \ddots & 0 \\ l_{n,1} & l_{n,2} & \cdots & l_{n,n} \end{bmatrix}$$

$$LL^T = \begin{bmatrix} l_{1,1} & 0 & \cdots & 0 \\ l_{2,1} & l_{2,2} & \ddots & \vdots \\ \vdots & \vdots & \ddots & 0 \\ l_{n,1} & l_{n,2} & \cdots & l_{n,n} \end{bmatrix} \begin{bmatrix} l_{1,1} & l_{2,1} & \cdots & l_{n,1} \\ 0 & l_{2,2} & \cdots & l_{n,2} \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & l_{n,n} \end{bmatrix}$$

$$= \begin{bmatrix} l_{1,1}^2 & l_{1,1}l_{2,1} & \cdots & l_{1,1}l_{n,1} \\ l_{2,1}l_{1,1} & l_{2,2}^2 + l_{2,1}^2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & l_{n,n-1}l_{n-1,n-1} + \sum_{i=1}^{n-2} l_{n,i}l_{i,n-1} \\ l_{n,1}l_{1,1} & \cdots & l_{n,n-1}l_{n-1,n-1} + \sum_{i=1}^{n-2} l_{n,i}l_{i,n-1} & l_{n,n}^2 + \sum_{i=1}^{n-1} l_{n,i}^2 \end{bmatrix}$$

Listing 5.8: Cholesky Factorization

```
function L=Cholesky(A)
n=size(A,2);
for col=1:n
  // work from diagonal out
  row=col;
  // square root of diagonal
  A(row,col)=sqrt(A(row,col));
  // scale rest of the row by the new diagonal
  for sub_row = row+1:n
    A(sub_row,col)=A(sub_row,col)/A(row,col);
  end
  // cleanup
  for sub_col=col+1:n
    // remove current column from remaining cols
    for sub_row=sub_col:n
      temp=A(sub_row,col)*A(sub_col,row);
      A(sub_row,sub_col)=A(sub_row,sub_col)-temp;
    end
    // zero the row above the diagonal
    A(row,sub_col)=0;
  end
```

```
end
L=A;
endfunction
```

5.3 QR Factorization

While in most practical systems LU or a specialized version like Cholesky will work nicely, there are times when numerical conditioning makes these methods dangerous. This is when we can use a more numerically savvy routine, QR factorization. The idea of QR factorization is to obtain an upper triangular matrix, R, similar to LU, but to do it with orthogonal transformations. Orthogonal transforms use orthogonal matrices, which have a variety of nice properties

1. The inverse is the transpose ($Q^{-1} = Q^T$)
2. The norm is 1 ($\|Q\| = 1$)
3. The norm of the inverse is 1 ($\|Q^{-1}\| = \|Q^T\| = \|Q\| = 1$)
4. The condition number is 1 ($cond(Q) = \|Q\|\|Q^{-1}\| = 1$)

The first and last property are particulary nice for us. Since the condition number is 1, we should not cause any errors to grow. Since the transpose is the inverse it is easy and relatively quick to calculate. There are several ways to do QR, which revolve around how to form Q or R. To understand these methods, you first need to understand basis, dimensionality of vector spaces, and orthogonality, but those in turn requires an understanding of independence. If you are not familiar with any of these concepts please read Section 11.1.

5.3.1 Gram-Schmidt Orthogonalization

This is the easiest to understand, but the worst to do.

Listing 5.9: Classic Gram-Schmidt Orthogonalization

```
function [Q,R]=gram_schmidt_classic(A)
  [rows,cols]=size(A);
  Q=A;
  R=zeros(cols,cols);
  for col=1:cols
    for orthogonal_dir = 1:col-1
      R(orthogonal_dir,col)=Q(:,orthogonal_dir)'*A(:,col);
      Q(:,col)=Q(:,col)-R(orthogonal_dir,col)*Q(:,orthogonal_dir);
    end
    R(col,col)=norm(Q(:,col));
    Q(:,col)=Q(:,col)/R(col,col);
  end
```

```
endfunction
```

5.3.2 Modified Gram-Schmidt Orthogonalization

The fix gains some precision but does not fix the root problem.

Listing 5.10: Modified Gram-Schmidt Orthogonalization

```
function [Q,R]=gram_schmidt_modified(A)
  [rows,cols]=size(A);
  Q=A;
  R=zeros(cols,cols);
  for col=1:cols
    R(col,col)=norm(Q(:,col));
    Q(:,col)=Q(:,col)/R(col,col);
    for remaining_col = col+1:cols
      R(col,remaining_col)=Q(:,col)'*Q(:,remaining_col);
      Q(:,remaining_col)=Q(:,remaining_col)-R(col,remaining_col)*Q(:,col);
    end
  end
endfunction
```

5.3.3 Givens Rotations

This works well, and is best suited to sparse matrix solvers because it can be used to target individual, non-zero elements without effecting other terms. Each Givens rotation only effects only two rows or columns in the target matrix (depends on if you do left or right multiplication). A rotation matrix is given by

$$R_\theta = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}. \tag{5.12}$$

Givens rotations are the 4 elements of a rotation matrix in a square pattern centered on the diagonal. Being rotations they are very nice to calculate, easy to express, but have a higher overhead to do them for every non-zero term. When there are only a few non-zero terms (i.e. sparse matrices) this is not bad, and they end up being much more efficient than other methods that will calculate how to zero a zero element.

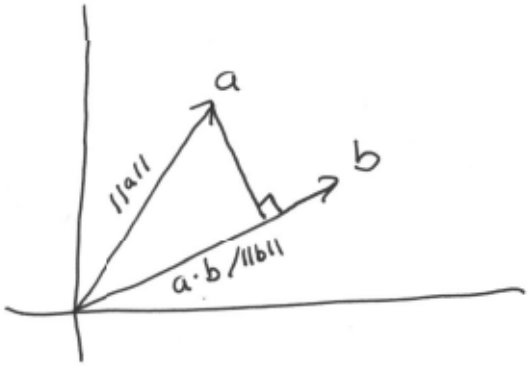
5.3.4 Householder Reflections

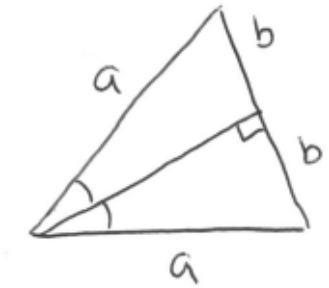
This works well, and is best suited to dense matrix solvers. When matrices have mostly non-zero elements (i.e. "dense matrices") the low overhead of zeroing an entire column with a Householder reflector is very efficient. These are hyperbolic, since they are reflectors, so

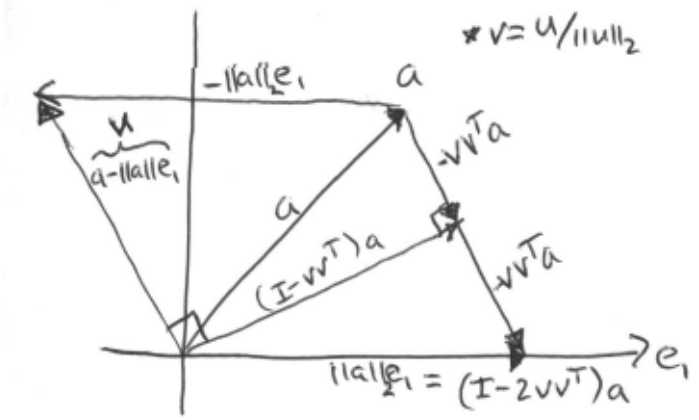
they can have some calculation problems, though this is not common. First I will state the result then I will prove it.

$$
\begin{aligned}
H_i &= I - 2vv^T && (5.13)\\
u &= a_i(i:m) - e_1\|a_i(i:m)\| && (5.14)\\
v &= \frac{u}{\|u\|} && (5.15)
\end{aligned}
$$

Note that by our construction that $v^T v = 1$. Additionally, observe that the size of the matrix diminishes by one row and column each iteration, this is because once a column has been reduced, one more row does not need to be effected. Due to this we only need to work on $A(i:m, i:n)$.







5.4 Singular Value Decomposition

The SVD is extremely stable and is in some sense the ultimate safe thing to do, if it is done right anyway. We do not often do it as it is expensive (read slow) to calculate. The danger in QR is in error buildup in solving, R, by back substitution. It would be much nicer to have a diagonal matrix so errors couldn't buildup. The SVD always exists and is a non-unique (i.e. there are many different SVD's for the same matrix) decomposition that is given by

$$A = U\Sigma V^T \tag{5.16}$$

where U and V are unitary (generalization of orthogonal to complex numbers) matrices, and Σ is a diagonal matrix with the main diagonal ordered so $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$. Thus Σ is not only diagonal, it is positive semidefinite (there could be zeros on the diagonal) and ordered. We can even calculate individual singular values (the σ_i). Some popular uses are

1. $\|A\|_2 = \sigma_1$
2. $\|A^{-1}\|_2 = \sigma_n^{-1}$
3. $cond(A) = \frac{\sigma_1}{\sigma_n}$
4. $\|A\|^2_{Frobenius} = \sum_{i=1}^{n} \sigma_i^2$
5. $A^{-1} = V\Sigma^{-1}U^T$, note Σ^{-1} is easy to calculate because it is diagonal.

Chapter 6

Integration

The fundamental theorem of Calculus tells us that an integral of a function can be expressed in terms of the anti-derivative of the function. Unfortunately, not all functions have anti-derivatives that are expressible in known functions. One of the most famous is the Gaussian probability distribution, which is given by

$$e^{-\left(\frac{x-\mu}{\sigma}\right)^2}.$$

The anti-derivative of this important and frequently occurring function is unknown. How do we handle it? That is the subject of this chapter.

6.1 Riemann

We recall from Calculus that the integral is defined as

$$\int_a^b f(x)dx = \lim_{n\to\infty} \sum_{j=1}^{n} f(p_j)(x_j - x_{j-1}).$$

Now assume that all n of the x_j are evenly spaced on $[a, b]$. We can then write

$$\begin{aligned} h &= \frac{b-a}{n} \\ &= x_j - x_{j-1}. \end{aligned}$$

We can use this to get an expression for the Riemann Sum

$$\begin{aligned}\int_a^b f(x)dx &= \lim_{n\to\infty}\sum_{j=1}^{n} f(p_j)(x_j - x_{j-1}) \\ &= \lim_{n\to\infty}\sum_{j=1}^{n} f(p_j)h \\ &= \lim_{n\to\infty} h\sum_{j=1}^{n} f(p_j).\end{aligned}$$

To evaluate the integral numerically we are not able to take the limit, so we get

$$\int_a^b f(x)dx \approx h\sum_{j=1}^{n} f(p_j).$$

The exact size of n for the approximation to be good is a key aspect of numerical integration. Note also that I have not specified what p_j is, as this form allows you to do a left, right, mid-point, maximum, or minimum. The basic idea here is that we are approximating the function by a constant on the interval.

$$\int_a^b f(x)dx \approx h\sum_{j=1}^{n} f(p_j).$$

Now we want to think about the error. First we want to get an expression for the integral in terms of an exact sum. To do this we use the fundamental theorem of calculus, add nothing, rearrange, and use the mean value theorem.

$$\begin{aligned}\int_a^b f(x)dx &= F(b) - F(a) \\ &= F(x_n) - F(x_0) \\ &= F(x_n) + \sum_{i=1}^{n-1}(F(x_i) - F(x_i)) - F(x_0) \\ &= \sum_{i=1}^{n}(F(x_i) - F(x_{i-1})) \\ &= \sum_{i=1}^{n} f(c_i)(x_i - x_{i-1}) \\ &= \sum_{i=1}^{n} f(c_i)h \\ &= h\sum_{i=1}^{n} f(c_i)\end{aligned}$$

This expression is exact so we can take it and subtract the expression for the midpoint method. We will assume the function has a derivative and we will note that since we have picked the midpoint that any other point in the interval is within half the width of the interval of the midpoint.

$$\begin{aligned} E &= h\sum_{i=1}^{n} f(c_i) - h\sum_{i=1}^{n} f(p_i) \\ &= h\sum_{i=1}^{n} (f(c_i) - f(p_i)) \\ &= h\sum_{i=1}^{n} f'(d_i)(c_i - p_i) \\ &\leq h\sum_{i=1}^{n} f'(d_i)\frac{h}{2} \\ &= \frac{h^2}{2}\sum_{i=1}^{n} f'(d_i) \end{aligned}$$

Recall that h is inversely proportional to n, so we have that the Error is inversely proportional to the square of n.

6.2 Trapezoid

$$\int_a^b f(x)dx \approx h\left(\frac{f(x_0) + f(x_n)}{2} + \sum_{j=1}^{n} f(x_j)\right)$$

6.3 Simpson

$$\int_a^b f(x)dx \approx \frac{h}{3}\left(f(x_0) + f(x_n) + 2\sum_{j=1}^{\frac{n}{2}-1} f(x_{2j}) + 4\sum_{j=1}^{\frac{n}{2}} f(x_{2j-1})\right)$$

6.4 Richardson

Define the value of the integral to be I and the numeric approximation by some method at n node points to be I_n. We note that the error is of the form

$$\begin{aligned} E &= I - I_n \\ &= \frac{c}{n^p}, \end{aligned}$$

with c a constant dependent on the function and p a power dependent on the method. For midpoint and trapezoidal methods $p = 2$, while for Simpson's method $p = 4$. If we double the number of points then the error would be

$$\begin{aligned} E &= I - I_{2n} \\ &= \frac{c}{2^p n^p} \\ &= \frac{1}{2^p}(I - I_n). \end{aligned}$$

Solving for I we find

$$\begin{aligned} I - I_{2n} &= \frac{1}{2^p}(I - I_n) \\ I - \frac{1}{2^p} I &= I_{2n} \frac{1}{2^p} I_n \\ \frac{2^p - 1}{2^p} I &= I_{2n} - \frac{1}{2^p} I_n \\ I &= \frac{2^p}{2^p - 1} I_{2n} - \frac{1}{2^p - 1} I_n \\ I &= \frac{1}{2^p - 1} \left(2^p I_{2n} - I_n\right). \end{aligned}$$

This is Richardson's Extrapolation formula, and it will typically give a big improvement to any of the methods. We can use this estimate of the error to calculate the error

$$\begin{aligned} E &= I - I_{2n} \\ &= \frac{2^p}{2^p - 1} I_{2n} - \frac{1}{2^p - 1} I_n - I_n \\ &= \frac{2^p - (2^p - 1)}{2^p - 1} I_{2n} - \frac{1}{2^p - 1} I_n \\ &= \frac{1}{2^p - 1} I_{2n} - \frac{1}{2^p - 1} I_n \\ &= \frac{1}{2^p - 1}(I_{2n} - I_n). \end{aligned}$$

This is Richardson's error formula. We note that we can get a convergence rate by doing a little algebra on what we have already found.

$$\begin{aligned} I - I_{2n} &= \frac{1}{2^p}(I - I_n) \\ \frac{I - I_n}{I - I_{2n}} &= 2^p \end{aligned}$$

This is a nice equation but in general it is not calculable, as we don't know I and might not know p. We can handle this by considering the quantity

$$\begin{aligned}
\frac{I_{2n} - I_n}{I_{4n} - I_{2n}} &= \frac{I_{2n} - I_n + I - I}{I_{4n} - I_{2n} + I - I} \\
&= \frac{(I - I_n) - (I - I_{2n})}{(I - I_{2n}) - (I - I_{4n})} \\
&= \frac{(I - I_n) - \frac{1}{2^p}(I - I_n)}{\frac{1}{2^p}(I - I_n) - \frac{1}{4^p}(I - I_n)} \\
&= \frac{1 - \frac{1}{2^p}}{\frac{1}{2^p}(1 - \frac{1}{2^p})} \\
&= \frac{1}{\frac{1}{2^p}} \\
&= 2^p.
\end{aligned}$$

We thus have a simple way of calculating the convergence rate, and thus a way to find p.

Homework: 7.1) 2(a), (b), (f) for midpoint, trapezoidal, and simpson. Compare with the error for trapezoidal (7.32) and Richardson's extrapolation.

7.2) 8

6.5 Gaussian Quadrature

And now for something completely different...

Last time we considered the standard way of thinking about integration, namely summing up a bunch of small areas. The technique we will discuss today was introduced in 1814 by Gauss, hence the name. We will now consider the integral

$$I(f) = \int_{-1}^{1} f(t)dt.$$

Note that this is a perfectly general statement, as all we need to do to convert this to an integral of the form $\int_a^b f(x)dx$ is to find a linear mapping between the two intervals of integration:

$$t = [-1, 1] \tag{6.1}$$
$$x = [a, b] \tag{6.2}$$

We thus want to find a function $x = g(t)$ such that $a = g(-1)$, $b = g(1)$, and $g(\cdot)$ is linear.

We have two points which determine a line, so we can use the two-point formula for a line:

$$\begin{aligned} m_g &= \frac{x_1 - x_0}{t_1 - t_0} && (6.3)\\ &= \frac{b-a}{1--1} && (6.4)\\ &= \frac{b-a}{2} && (6.5)\\ x &= x_1 + m_g(t - t_1) && (6.6)\\ &= b + \frac{b-a}{2}(t-1) && (6.7)\\ &= \frac{1}{2}(2b + t(b-a) - (b-a)) && (6.8)\\ &= \frac{1}{2}(t(b-a) + b + a) && (6.9)\\ &= \frac{t(b-a) + b + a}{2} && (6.10) \end{aligned}$$

So, if we define $x = 0.5(t(b-a) + b + a)$ and thus $dx = 0.5(b-a)dx$ then we have[1]

$$\begin{aligned} \int_a^b f(x)dx &= \int_{g(-1)}^{g(1)} f(x)g'(t)dt \\ &= \int_{-1}^{1} f\left(\frac{(b-a)t + b + a}{2}\right)\left(\frac{b-a}{2}\right)dt \end{aligned}$$

We have that the integral is general, if not all that obvious as to why we chose this in the first place (it will become more apparent in a few moments). We still need to show how to estimate the integral. The basic idea is to approximate the integral by weighted evaluations of the function at a series of node points. To get a good estimate we will require that our estimate at n node points will be good for every polynomial up to order $2n-1$. How? Pick w_i and x_i such that

$$I(f) = \sum_{i=1}^{n} w_i f(x_i)$$

holds for all $f \in \{1, x, \ldots, x^{2n-1}\}$. Let's do some examples.

Example:

[1] You will see some books define $t = (2x - a - b)/(b-a)$, and $dt = 2dx/(b-a)$, which is the same formula but not as convenient for what we want.

Consider $n = 1$.

$$\begin{aligned} \int_{-1}^{1} dx &= w_1 f(x_1) \\ 2 &= w_1 \\ \int_{-1}^{1} x dx &= w_1 x_1 \\ 0 &= 2x_1 \\ 0 &= x_1 \end{aligned}$$

Also, consider $n = 2$.

$$\begin{aligned} \int_{-1}^{1} dx &= w_1 f(x_1) + w_2 f(x_2) \\ 2 &= w_1 + w_2 \\ \int_{-1}^{1} x dx &= w_1 f(x_1) + w_2 f(x_2) \\ 0 &= w_1 x_1 + w_2 x_2 \\ \int_{-1}^{1} x^2 dx &= w_1 f(x_1) + w_2 f(x_2) \\ \frac{2}{3} &= w_1 x_1^2 + w_2 x_2^2 \\ \int_{-1}^{1} x^3 dx &= w_1 f(x_1) + w_2 f(x_2) \\ 0 &= w_1 x_1^3 + w_2 x_2^3 \end{aligned}$$

Solve these four equations for four unknowns and we find

$$\begin{aligned} w_1 &= 1 \\ w_2 &= 1 \\ x_1 &= -\frac{1}{\sqrt{3}} \\ x_1 &= \frac{1}{\sqrt{3}} \end{aligned}$$

The node points turn out to be the roots of the Legendre polynomials. The Legendre polynomials are defined on $[-1, 1]$ hence our choice for the limits of integration. You can find the Legendre polynomials by the following properties.

- P_n is a polynomial of order n.
- They are orthogonal

$$\int_{-1}^{1} P_i(x) P_j(x) dx = 0$$

Table 6.1: Gauss-Legendre Abscissae and Weights to 8 Decimal Places

n	Evaluation Points, x_i	Weights, w_i	Degree (2n-1)
1	0.0	2.0	1
2	$\pm\frac{1}{\sqrt{3}}$	1.0	3
3	0.0	0.88888889	5
	±0.77459667	0.55555555	
4	±0.33998104	0.65214515	7
	±0.86113631	0.34785485	
5	0.0	0.56888889	9
	±0.53846931	0.47862867	
	±0.90617985	0.23692689	
6	±0.23861918	0.46791393	11
	±0.66120939	0.36076157	
	±0.93246951	0.17132449	
7	0.0	0.41795918	13
	±0.40584515	0.38183005	
	±0.74153119	0.27970539	
	±0.94910791	0.12948497	
8	±0.18343464	0.36268378	15
	±0.52553241	0.31370665	
	±0.79666648	0.22238103	
	±0.96028986	0.10122854	
10	±0.14887434	0.29552422	19
	±0.43339539	0.26926672	
	±0.67940957	0.21908636	
	±0.86506337	0.14945135	
	±0.97390653	0.06667134	

when $i \neq j$.

- The normalization is

$$\int_{-1}^{1} P_n(x)^2 dx = \frac{2}{2n+1}$$

The first several Legendre polynomials are $\{1, x, x^2 - \frac{1}{3}, x^3 - \frac{3}{5}x, x^4 - \frac{6}{7}x^2 + \frac{3}{35}\}$.

The constants can be found by

$$w_i = \int_{-1}^{1} \prod_{j=1, j\neq i}^{n} \frac{x - x_j}{x_i - x_j} dx$$

In general we do not need to use this as the values are well tabulated, see for instance Table 6.1.

Homework Redo 7.1) 2(a), (b), (f) for gaussian quadrature up to $n = 8$. How does the convergence compare?

Chapter 7

Differentiation

Numerical differentiation seems obvious. In most introductions to calculus, the derivative is introduced as the limit of a series of secant lines. Probably the most basic method of numerically taking the derivative is based off this formula. It turns out that while it seems obvious, numerical derivatives are more difficult as we will see.

7.1 Derivative Approximation

Let's start at the basics, the two point method of obtaining the derivative is based off taking the derivative of the two point interpolation polynomial.

$$\begin{aligned} f(x) &\approx \frac{x_1 - x}{h} f(x_0) + \frac{x - x_0}{h} f(x_1) \\ f'(x) &\approx \frac{-1}{h} f(x_0) + \frac{1}{h} f(x_1) \\ &= \frac{f(x+h) - f(x)}{h}. \end{aligned}$$

This is only exact when we take the limit as h approaches zero. We can see how the basic errors are dealt with by expanding the Taylor sequence.

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2} f''(c)$$

Using the Taylor series in the formula we obtain that the error in the formula is

$$\begin{aligned} E &= f'(x) - \frac{f(x+h) - f(x)}{h} \\ &= f'(x) - \frac{f(x) + hf'(x) + \frac{h^2}{2}f''(c) - f(x)}{h} \\ &= f'(x) - \frac{hf'(x) + \frac{h^2}{2}f''(c)}{h} \\ &= f'(x) - f'(x) + \frac{h}{2}f''(c) \\ &= \frac{h}{2}f''(c). \end{aligned}$$

Two other types of error occur. First, there is subtractive difference errors. We note that since $f(x) \approx f(x+h)$, when we subtract them we will lose precision. Second, there is error propagation. Consider the fact that nothing we do in the real world is ever exact, so we actually measure and calculate a nearby solution:

$$\begin{aligned} f(x) &= \tilde{f}(x) + \epsilon_1 \\ f(x+h) &= \tilde{f}(x+h) + \epsilon_2. \end{aligned}$$

Substituting this into our formula we find

$$\begin{aligned} f'(x) &\approx \frac{f(x+h) - f(x)}{h} \\ &= \frac{\tilde{f}(x+h) + \epsilon_2 - \tilde{f}(x) + \epsilon_1}{h} \\ &= \frac{\tilde{f}(x+h) - \tilde{f}(x)}{h} + \frac{\epsilon_2 - \epsilon_1}{h}. \end{aligned}$$

The resulting error is

$$\begin{aligned} E &= f'(x) - \frac{f(x+h) - \epsilon_2 - f(x) + \epsilon_1}{h} \\ &= f'(x) - \frac{f(x) + hf'(x) + \frac{h^2}{2}f''(c) - \epsilon_2 - f(x) + \epsilon_1}{h} \\ &= f'(x) - \frac{hf'(x) + \frac{h^2}{2}f''(c) - \epsilon_2 + \epsilon_1}{h} \\ &= f'(x) - f'(x) + \frac{h}{2}f''(c) + \frac{\epsilon_1 - \epsilon_2}{h} \\ &= \frac{h}{2}f''(c) + \frac{\epsilon}{h}. \end{aligned}$$

As long as the last term is non-zero, which it will be in general, if h gets to small them the propagation errors dominate, see Figure 7.1. This is the real problem. We did not have this

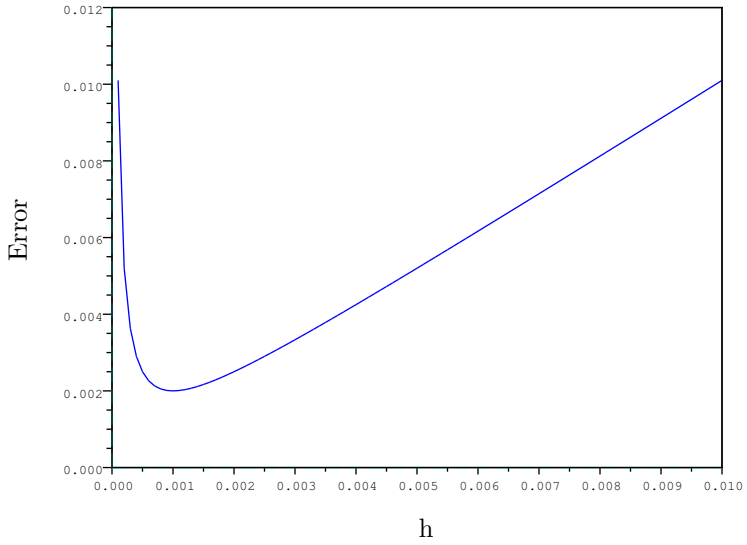


Figure 7.1: Error in calculating the step size for different step sizes, h, for $f''(c) = 2$ and $\epsilon = 10^{-6}$.

problem with numerical integration because we were multiplying by h rather than dividing by it. The problem is more pronounced with higher order derivatives where you will be dividing by powers of h. This problem motivates the use of integral equations rather than differential ones. Unfortunately many integral equations are very ill-conditioned. Anyway, the key idea is that there is a competition between several error types.

7.2 Tangent Line

As of yet we have considered calculating the derivative from above. Let us consider the derivative from below, and we will solve for it using the equation of a line and noting that the slope of the tangent line in the limit as $h = 0$ is the derivative[1].

$$y = mx + b$$

thus

$$\begin{aligned} f(x) &= mx + b \\ f(x-h) &= m(x-h) + b. \end{aligned}$$

[1] We could have just used the same method we used in the last section, but what fun is that?

Take the difference

$$\begin{aligned} f(x) - f(x-h) &= mx - m(x-h) \\ &= mh \\ m_{left} &= \frac{f(x) - f(x-h)}{h} \end{aligned}$$

This is the left approximation of the derivative and

$$m_{right} = \frac{f(x+h) - f(x)}{h}$$

is the right approximation. We get a better estimate by averaging them.

$$\begin{aligned} m_{avg} &= \frac{m_{left} + m_{right}}{2} \\ &= \frac{\frac{f(x)-f(x-h)}{h} + \frac{f(x+h)-f(x)}{h}}{2} \\ &= \frac{f(x) - f(x-h) + f(x+h) - f(x)}{2h} \\ &= \frac{f(x+h) - f(x-h)}{2h} \end{aligned}$$

Interestingly the best thing we can do when we have three points is not to use the one we are trying to estimate the derivative of. The reason is, when we use the left approximation of the derivative, we are really estimating the derivative at $x - \frac{h}{2}$, and similarly for the right approximation. By averaging we get an estimate of the derivative at x.

7.3 Richardson Extrapolation

7.4 Higher Order Derivatives

The second derivative is just the derivative of the first derivative. Since the left and right approximations really give us the approximate derivative at $x \pm \frac{h}{2}$, we can use them, with their separation distance of h, to estimate the second derivative at $f(x)$.

$$\begin{aligned} f''(x) &\approx \frac{f'\left(x + \frac{h}{2}\right) - f'\left(x - \frac{h}{2}\right)}{h} \\ &\approx \frac{\frac{f(x+h)-f(x)}{h} - \frac{f(x)-f(x-h)}{h}}{h} \\ &\approx \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} \end{aligned}$$

To solve for higher derivatives we will need more points: $f(x+2h)$, $f(x-2h)$, etc. The further we go away the worse the approximation will be though, again showing that higher order derivatives become more inaccurate.

7.4.1 Method of Undetermined Coefficients

In addition to the point the derivative is to be evaluated at, we need one additional point per order of the derivative to be taken. That is a first derivative takes 2 points, a second derivative takes 3 points, a third derivative takes 4, and so on. Rather than memorize a bunch of formulas for higher powers, or even iterating the formula as above, we can use the method of undetermined coefficients, which can also be used for integration, interpolation, differential equations, etc.

Say we wanted to find the formula for the second derivative, which we know will require 3 points.

$$f''(x) = Af(x+h) + Bf(x) + Cf(x-h)$$

We will use the Taylor Series to expand $f(x+h)$ and $f(x-h)$ around x.

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2}f''(x) + hot$$
$$f(x-h) = f(x) - hf'(x) + \frac{h^2}{2}f''(x) + hot$$

We now substitute this into the original expression.

$$\begin{aligned} f''(x) &= A\left(f(x) + hf'(x) + \frac{h^2}{2}f''(x) + hot\right) + Bf(x) + \\ &\qquad C\left(f(x) - hf'(x) + \frac{h^2}{2}f''(x) + hot\right) \\ &= (A+B+C)f(x) + (A-C)hf'(x) + (A+C)\frac{h^2}{2}f''(x) + hot \end{aligned}$$

Equating like derivatives on the left and right we have

Derivative	Equation	Implication
$f(x)$	$A+B+C=0$	$B=-(A+C)$
$f'(x)$	$(A-C)h=0$	$A=C$
$f''(x)$	$(A+C)\frac{h^2}{2}=1$	$A=\frac{1}{h^2}$

thus

$$\begin{aligned} f''(x) &= \frac{1}{h^2}f(x+h) + \frac{-2}{h^2}f(x) + \frac{1}{h^2}f(x-h) \\ &= \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}. \end{aligned}$$

Note that we found the same formula a different way.

Chapter 8

Differential Equations

Most practical problems will be described by a differential equation. We will not in general know the form of the solution, but we usually can find how they change with respect to each other. From this basis we would like to be able to find the actual solution.

8.1 General Introduction

Consider the general differential equation

$$\dot{y}(x) = f(x, y(x)). \tag{8.1}$$

Using basic calculus we see that the solution is given by

$$y(x) = \int f(x, y(x))dx + c.$$

Often this is not very useful in solving equations however. The way to get practical solutions for this problem is covered in differential equations, so I will just mention a few.

8.1.1 Existence

The problem 8.1 has a solution on an interval $x_0 \leq x \leq \min(b_x, c)$ and $y_0 \leq y \leq b_y$ if the function f is continuous on the interval for $c = \frac{\|y-y_0\|}{\max_{x,y}\left(\frac{\partial f(x,y)}{\partial y}\right)}$.

8.2 Euler's Method

Euler was without a doubt one of the greatest mathematical minds to have ever lived. He did work in almost every area you can imagine including numerical methods. His method is simple and easy for people to use. Unfortunately, as the old numerics folk theorem goes, "that which is good for a person is bad for a computer and vise versa" Euler's method is

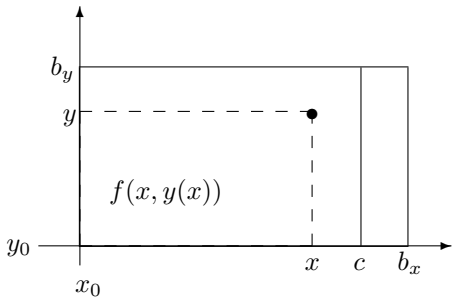


Figure 8.1: Existence requirements

unstable numerically[1]. To introduce the method, say we knew a point, say $(x_0, y(x_0))$, we could denote our first point by the y-coordinate.

$$y_0 = y(x_0)$$

Now say we wanted to find the value of the function a short distance away, say at $x_0 + h$. It would be reasonable to do a first order taylor approximation. Noting that $\dot{y} = f(x, y)$ allows us to write this in simple terms.

$$\begin{aligned} y_1 &= y(x_0 + h) \\ &= y_0 + hf(x_0, y_0) \end{aligned}$$

We could extend this process to find

$$\begin{aligned} y_n &= y(x_0 + nh) \\ &= y_{n-1} + hf(x_{n-1}, y_{n-1}) \\ &= y_{n-2} + hf(x_{n-2}, y_{n-2}) + hf(x_{n-2} + h, y_{n-2} + hf(x_{n-2}, y_{n-2})) \end{aligned}$$

Note that the third (last) term involves estimating based on an estimate. It is here that the problem comes because the errors can build. It is easier to show the problem by taking the Z-Transform of the second line

$$\begin{aligned} y_n &= y_{n-1} + hf(x_{n-1}, y_{n-1}) \\ Y &= z^{-1}Y + hF(z^{-1}X, z^{-1}Y) \end{aligned}$$

[1]The forward method actually, the backward method of Euler is stable though still far from a good method.

Assuming we are dealing with linear functions (a common assumption) we can pull the z^{-1} out. Error is given by

$$\begin{aligned} Y(x) - y_h(x) &= hD(x) + \mathcal{O}(h^2) \\ D'(x) &= g(x)D(x) + \frac{1}{2}Y''(x), \qquad D(x_o) = 0 \\ g(x) &= \left.\frac{\partial f(x,z)}{\partial z}\right|_{z=Y(x)} \end{aligned}$$

8.3 Runge-Kutta

While Euler's method is nice and simple, it is far from the best. Higher order Taylor methods can be derived but these require evaluating multiple derivatives. Even Richardson's extrapolation has limits on its abilities.

Consider the Taylor series of y.

$$\begin{aligned} y(x+h) &= y(x) + hy'(x) + \frac{h^2}{2}y''(x) + \dots \\ &= y(x) + hf(x,y) + \frac{h^2}{2}f'(x,y) + \dots \\ &= y(x) + hf(x,y) + \frac{h^2}{2}\left(f_x(x,y) + f_y(x,y)f(x,y)\right) + \dots \\ &= y(x) + hf(x,y) + \frac{h^2}{2}\left(f_x(x,y) + f_y(x,y)f(x,y)\right) + \dots \\ &= y(x) + hf(x,y) + ah\left(\frac{h}{2a}f_x(x,y) + \frac{h}{2a}f(x,y)f_y(x,y)\right) + \dots \end{aligned}$$

Now we consider the Taylor series in two variables of $f(x,y)$.

$$\begin{aligned} f(x+h,y+k) &= \sum_{n=0}^{\infty}\frac{1}{n!}\left[h\frac{\partial}{\partial x} + k\frac{\partial}{\partial y}\right]^n f(x,y) \\ &= f(x,y) + \left[h\frac{\partial}{\partial x} + k\frac{\partial}{\partial y}\right]f(x,y) + \dots \\ &= f(x,y) + hf_x(x,y) + kf_y(x,y) + \dots \end{aligned}$$

Rearranging we find

$$f(x+h,y+k) - f(x,y) = hf_x(x,y) + kf_y(x,y) + \dots.$$

We use this in our Taylor series of y.

$$\begin{aligned} y(x+h) &= y(x) + hf(x,y) + ah\left(\frac{h}{2a}f_x(x,y) + \frac{h}{2a}f(x,y)f_y(x,y)\right) + \dots \\ &= y(x) + hf(x,y) + h\left(af(x+\frac{h}{2a}, y+\frac{h}{2a}f(x,y)) - af(x,y)\right) + \dots \\ &= y(x) + h(1-a)f(x,y) + ahf(x+\frac{h}{2a}, y+\frac{h}{2a}f(x,y)) + \dots \\ &= y(x) + h(1-a)f(x,y) + ahf(x+b, y+bf(x,y)) + \dots \\ & \qquad b = \frac{h}{2a} \qquad 0 \le a \le 1 \end{aligned}$$

We have defined this in order to take advantage of the many varieties of second order R-K. The most common are: Midpoint (a=1), Modified Euler (a=1/2), and Heun's method (a=3/4).

You can also do R-K for higher orders. The most common is fourth order. The algebra is tedious so I will just present the result.

$$y(x+h) = y(x) + \frac{1}{6}\left(F_1 + 2F_2 + 2F_3 + F_4\right)$$

with

$$\begin{aligned} F_1 &= hf(x,y) \\ F_2 &= hf(x+\frac{h}{2}, y+\frac{F_1}{2}) \\ F_3 &= hf(x+\frac{h}{2}, y+\frac{F_2}{2}) \\ F_4 &= hf(x+h, y+F_3) \end{aligned}$$

8.4 Fehlberg's Method

We have seen that the local error (error in one step due mostly to truncation) is one order of magnitude better than the global error, in general. Often we use Richardson's Error formula to find an estimate of the local error (T_n) so we can adjust the step size to keep things nice. For instance Richardson's Error for Euler's method gives us

$$\begin{aligned} Y(x) - y_h(x) &\approx hD(x) \\ Y(x) - y_{2h}(x) &\approx 2hD(x) \\ Y(x) - y_{2h}(x) &\approx 2(Y(x) - y_h(x)) \\ Y(x) &\approx 2y_h(x) - y_{2h}(x) \\ Y(x) - y_h(x) &\approx (2y_h(x) - y_{2h}(x)) - y_h(x) \\ &\approx y_h(x) - y_{2h}(x). \end{aligned}$$

This gives us a reasonable extrapolation formula, and estimate of the error. Another way to estimate the error would be to look at two estimates from different order methods. For instance you could do a fourth and a fifth order R-K estimate at each step and use the difference to bound the error. If the error at any step became to large then you would decrease the step size and try again. This idea is Fehlberg's method, and it is the basis of most modern ode solvers. This is why Matlab has ode23 and ode45.

8.5 Adams-Bashforth

Up till now we have been looking at solving the differential equation directly. We could however just integrate both sides.

$$\begin{aligned} y' &= f(x,y) \\ \int_{x_n}^{x_{n+1}} y'dx &= \int_{x_n}^{x_{n+1}} f(x,y)dx \\ y(x_{n+1}) - y(x_n) &= \int_{x_n}^{x_{n+1}} f(x,y)dx \\ y(x_{n+1}) &= y(x_n) + \int_{x_n}^{x_{n+1}} f(x,y)dx \end{aligned}$$

Our task is now reduced to trying to find the remaining integral, which we can use the ideas we had from last chapter. Adams-Bashforth of order m uses a polynomial approximation to $f(x,y)$ at the points x_n, x_{n-1}, ..., x_{n-m+1}. Consider the second order Adams-Bashforth

method.

$$\begin{aligned}
f(x,y(x)) &\approx \frac{x_n - x}{h} f(x_{n-1}) + \frac{x - x_{n-1}}{h} f(x_n) \\
\int_{x_n}^{x_{n+1}} f(x,y)dx &\approx \int_{x_n}^{x_{n+1}} \left(\frac{x_n - x}{h} f(x_{n-1}) + \frac{x - x_{n-1}}{h} f(x_n) \right) dx \\
&\approx \frac{x_n x - \frac{1}{2}x^2}{h} f(x_{n-1}) + \frac{\frac{1}{2}x^2 - x_{n-1}x}{h} f(x_n) \Big|_{x_n}^{x_{n+1}} \\
&\approx \frac{x_n h - \frac{x_{n+1}^2 - x_n^2}{2}}{h} f(x_{n-1}) + \frac{\frac{x_{n+1}^2 - x_n^2}{2} - x_{n-1}h}{h} f(x_n) \\
&\approx \frac{x_n h - \frac{x_{n+1}^2 - x_{n+1}x_n + x_{n+1}x_n - x_n^2}{2}}{h} f(x_{n-1}) \\
&\quad + \frac{\frac{x_{n+1}^2 - x_{n+1}x_n + x_{n+1}x_n - x_n^2}{2} - x_{n-1}h}{h} f(x_n) \\
&\approx \frac{x_n h - \frac{x_{n+1}h + x_n h}{2}}{h} f(x_{n-1}) + \frac{\frac{x_{n+1}h + x_n h}{2} - x_{n-1}h}{h} f(x_n) \\
&\approx \frac{2x_n - (x_{n+1} + x_n)}{2} f(x_{n-1}) + \frac{x_{n+1} + x_n - 2x_{n-1}}{2} f(x_n) \\
&\approx \frac{x_n - x_{n+1}}{2} f(x_{n-1}) + \frac{x_{n+1} - x_{n-1} + x_n - x_{n-1}}{2} f(x_n) \\
&\approx \frac{-h}{2} f(x_{n-1}) + \frac{2h + h}{2} f(x_n) \\
&\approx \frac{h}{2}(3f(x_n) - f(x_{n-1}))
\end{aligned}$$

This is kind of ugly, so it would be nice to have a faster way of handling things, especially as the dimensions increase. Luckily there is just such a technique. The method of undetermined coefficients. We start by assuming the general form we want, in this case,

$$\int_{x_n}^{x_{n+1}} f(x,y)dx \approx af(x_n) + bf(x_{n-1}$$

We would like the approximation to work perfectly for constant and linear terms so:

Constant term, $f(x,y) = 1$

$$\begin{aligned}
\int_{x_n}^{x_{n+1}} 1dx &= a \cdot 1 + b \cdot 1 \\
h &= a + b.
\end{aligned}$$

Linear term, $f(x,y) = x$

$$\begin{aligned}
\int_{x_n}^{x_{n+1}} x dx &= a \cdot x_n + b \cdot x_{n-1} \\
\frac{x_{n+1}^2 - x_n^2}{2} &= a \cdot x_n - a \cdot x_{n-1} + a \cdot x_{n-1} + b \cdot x_{n-1} \\
\frac{x_{n+1}^2 - x_n x_{n+1} + x_n x_{n+1} - x_n^2}{2} &= a \cdot (x_n - x_{n-1}) + (a+b) \cdot x_{n-1}.
\end{aligned}$$

Now noting that $a + b = h$

$$\begin{aligned}
\frac{x_{n+1}(x_{n+1} - x_n) + x_n(x_{n+1} - x_n)}{2} &= a \cdot (h) + (h) \cdot x_{n-1} \\
\frac{x_{n+1}h + x_n h}{2} &= a \cdot h + h \cdot x_{n-1} \\
\frac{x_{n+1} + x_n - 2x_{n-1}}{2} &= a \\
\frac{x_{n+1} - x_{n-1} + x_n - x_{n-1}}{2} &= a \\
\frac{2h + h}{2} &= a \\
\frac{3h}{2} &= a.
\end{aligned}$$

Thus since $a + b = h$,

$$b = \frac{-h}{2}$$

which is what we found before.

8.6 Adams-Moulton

Adams-Bashforth considered that we knew only up to $f(x,y)$ only at points up to x_n. What if we assume we can use x_n or a near approximation? Let us again consider just the simple case of linear approximations to function, though we could use any order of polynomial we liked (if we want to do the work).

$$f(x, y(x)) \approx \frac{x - x_n}{h} f(x_{n+1}) + \frac{x_{n+1} - x}{h} f(x_n)$$

Using this we can solve the integral.

$$\begin{aligned}
\int_{x_n}^{x_{n+1}} f(x,y)dx &\approx \int_{x_n}^{x_{n+1}} \left(\frac{x-x_n}{h}f(x_{n+1}) + \frac{x_{n+1}-x}{h}f(x_n)\right)dx \\
&\approx \frac{\frac{1}{2}x^2}{h}f(x_{n+1}-x_n x) + \frac{x_{n+1}x - \frac{1}{2}x^2}{h}f(x_n)\Big|_{x_n}^{x_{n+1}} \\
&\approx \frac{\frac{x_{n+1}^2-x_n^2}{2} - x_n x_{n+1} + x_n^2}{h}f(x_{n+1}) + \frac{x_{n+1}^2 - x_{n+1}x_n - \frac{x_{n+1}^2-x_n^2}{2}}{h}f(x_n) \\
&\approx \frac{x_{n+1}^2 - x_n^2 - 2x_n x_{n+1} + 2x_n^2}{2h}f(x_{n+1}) + \frac{2x_{n+1}^2 - 2x_{n+1}x_n - x_{n+1}^2 + x_n^2}{2h}f(x_n) \\
&\approx \frac{x_{n+1}^2 - 2x_n x_{n+1} + x_n^2}{2h}f(x_{n+1}) + \frac{x_{n+1}^2 - 2x_{n+1}x_n - +x_n^2}{2h}f(x_n) \\
&\approx \frac{(x_{n+1}-x_n)^2}{2h}f(x_{n+1}) + \frac{(x_{n+1}-x_n)^2}{2h}f(x_n) \\
&\approx \frac{h^2}{2h}f(x_{n+1}) + \frac{h^2}{2h}f(x_n) \\
&\approx \frac{h}{2}(f(x_{n+1}) + f(x_n))
\end{aligned}$$

8.7 Stability & Stiff Equations

A good start for looking at stiff equations is to examine the stability of our methods. Consider Forward Euler for

$$y' = -200y, \qquad y(1) = e^{-200}.$$

The solution can easily be seen to be e^{-200x}. Forward Euler gives us

$$\begin{aligned}
y_{i+1} &= y_i - 200 * h * y_i \\
&= y_i(1 - 200 * h).
\end{aligned}$$

We note that this is stable if $\|1 - 200 * h\| < 1$. Since $h > 0$ we must have $h < 0.01$. This is a smooth, monotonically decreasing function that is less than 2^{-87} and greater than zero. Despite the smoothness and flatness, we have to take very small steps. To see this look at Figure 8.2. Note that Backward Euler does not have the same problem. It is defined by

$$\begin{aligned}
y_{i+1} &= y_i - 200 * h * y_{i+1} \\
&= y_i \frac{1}{1 + 200 * h},
\end{aligned}$$

which is stable for all $h > 0$. Stability is not the same thing as stiffness, but they are related. Stiffness is due to multiple scales of the terms, for instance e^{-x}, e^{-200x}. Consider the following equation

$$0 = \ddot{y} + 1001\dot{y} + 1000y, \qquad y(0) = 1, \qquad \dot{y}(0) = -1.$$

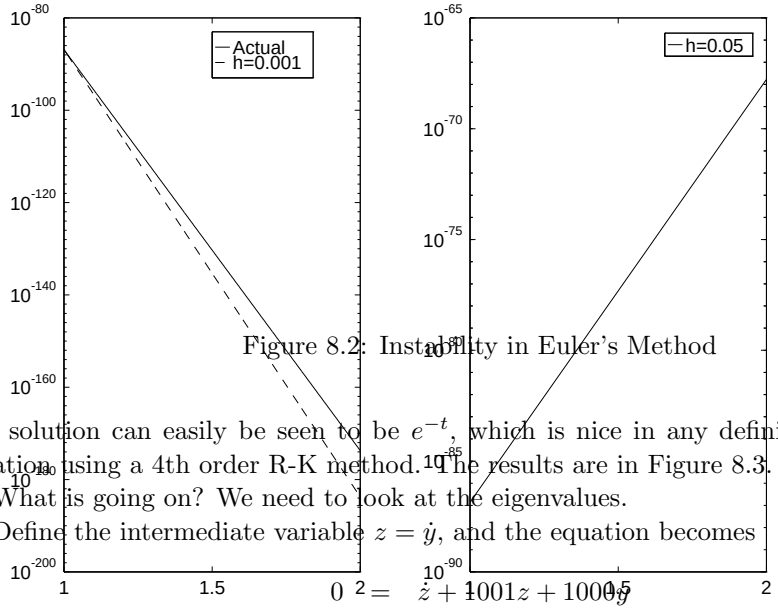


Figure 8.2: Instability in Euler's Method

The solution can easily be seen to be e^{-t}, which is nice in any definition. We solve the equation using a 4th order R-K method. The results are in Figure 8.3.

What is going on? We need to look at the eigenvalues.

Define the intermediate variable $z = \dot{y}$, and the equation becomes

$$\begin{aligned} 0 = \ddot{z} + 1001z + 1000y \\ \dot{z} = -1001z - 1000y. \end{aligned}$$

Putting this in matrix form

$$\begin{bmatrix} \dot{y} \\ z \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -1000 & -1001 \end{bmatrix} \begin{bmatrix} y \\ z \end{bmatrix}$$

The eigenvalues are (-1,-1000). Since the eigenvalues differ by three orders of magnitude we can expect stiffness, which is what Figure 8.3 shows.

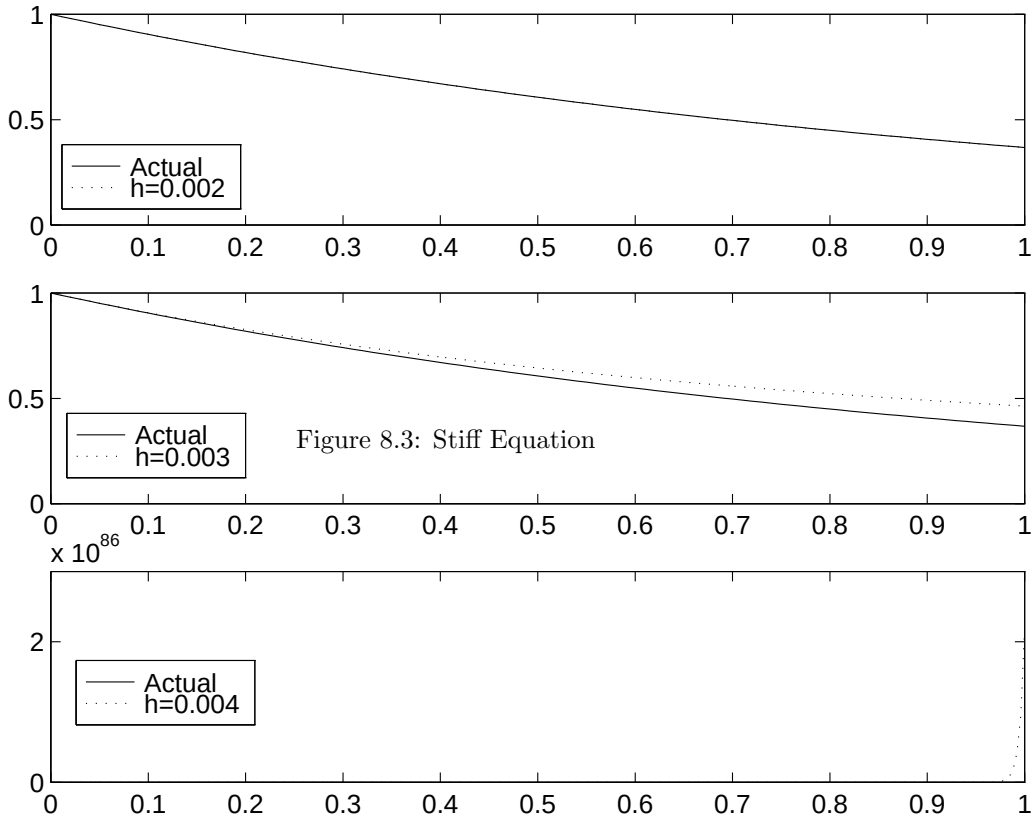


Figure 8.3: Stiff Equation

Chapter 9

Modeling and Simulation

Some systems cannot be practically tested. Consider the spread of a highly infectious disease, or the destruction due to a nuclear war. We can look at smaller examples of these, but the full blown case would be bad to run. What we would like to do is to take our mathematical knowledge and create a system of equations which will have the same behavior. We can then test the real system by seeing how the modeled system behaves. This action of making a a mathematical model be similar to the real system is simulation (same root as similar). Often these systems are expressed as either differential or difference equations. They can thus be solved by our differential equation solvers directly, or transformed (Laplace, Fourier, Z, etc.) so they can be solved by zero-finding.

Lets look at how we can set some of these up, and then give some examples.

9.0.1 Electric Circuit

Kirkoff's law tells us that the sum of the voltage drops around a loop must be zero. We can use this to figure out how to set up some models. To do so we need to note how each element operates. On the attached sheets I have included some copies of physical elements and how they relate. If we consider our basic variable as charge then current is just the time derivative of charge and we can see the elements are just:

$$\begin{aligned} V_{inductor} &= L\ddot{q} \\ V_{resistor} &= R\dot{q} \\ V_{capacitor} &= \frac{1}{C}q \end{aligned}$$

We can then write an equation for each loop and plot the first derivative of each of the loop charges (thus loop current).

Low Pass Filter

The sums of the voltage drops around the loop is

$$V_{in} - R\dot{q} - \frac{1}{C}q = 0 \tag{9.1}$$

The output voltage is the voltage across the capacitor, so

$$V_{out} = \frac{1}{C}q \tag{9.2}$$

We thus need to find q, and we know (rearranging eq 9.1)

$$\dot{q} = \frac{V_{in}}{R} - \frac{1}{RC}q \tag{9.3}$$

This equation becomes code 9.1.

Listing 9.1: Code for the differential equation describing a low pass filter.

```
function qdot=lowpass(q,t)
  R=100;
  C=.01;
  V=sin(100*t)+sin(t);
  qdot=V/R-q/(R*C);
endfunction
```

When called by

Listing 9.2: Code to simulate a low pass filter using RK 2nd order.

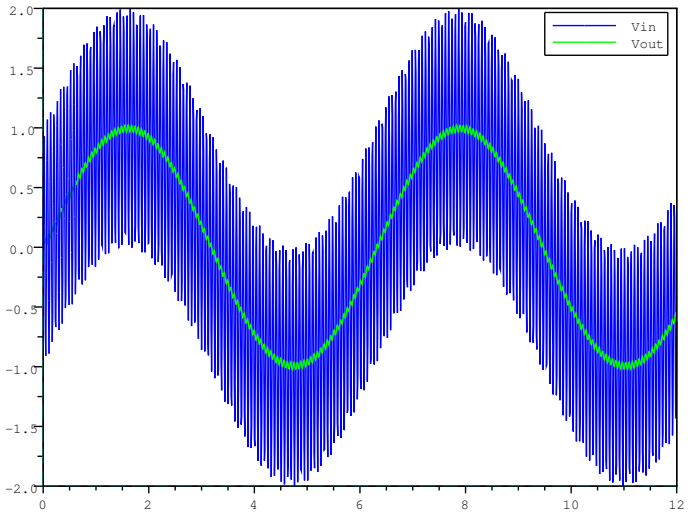
```
t=0:.01:12;
vin=sin(100*t)+sin(t);
q0=0;
q=zeros(t);
q(1)=q0;
h=.1;
a=1;
b=1/(2*a);

for i=2:max(size(t))
  qdot_pt1=lowpass(q(i-1),t(i));
  qdot_pt2=lowpass(q(i-1)+a*h*lowpass(q(i-1),t(i)),t(i)+a*h);
  q(i)=q(i-1)+h*(1-b)*qdot_pt1+b*h*qdot_pt2;
end

C=.01;
Vout=q/C;

plot(t,[vin],'-b',t,[Vout],'-g')
legend('Vin','Vout')
```

Figure 9.1: Input and output voltages of the low pass filter.



9.0.2 System of Masses

Consider a system of masses connected by springs and dampers. We can note that the if we choose our variable to be position then we have:

$$\begin{aligned} F_{spring} &= -kx \\ F_{damper} &= -c\dot{x} \\ F_{mass} &= m\ddot{x} \end{aligned}$$

9.0.3 Linearized Pendulum

$$\begin{bmatrix} \dot{x_1} \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\sqrt{\frac{g}{l}} & -c \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

9.0.4 Fox and Rabbits

Consider the problem where there is a predator and its prey in the area. If there are too many predators they might eat off all the prey and starve themselves. If there are too much of the prey the predators will increase in number because of the plentiful food. How do we model this?

We want several things in our model. First, predators should naturally starve, only the presence of prey offsets this. Prey should naturally breed, and predators offset this. The offset to both should be a scaled proportion of the contacts. Contact should be rare when either population is low and more frequent with a reasonable number of both. These considerations lead to the model that contacts occur at a rate proportional to the product of predators and prey. These rules are combined in the model below.

$$\begin{aligned}\begin{bmatrix}\dot{x_1}\\ x_2\end{bmatrix} &= \begin{bmatrix}-m_1 & 0\\ 0 & m_2\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix} + \begin{bmatrix}b_1x_1x_2\\ -b_2x_1x_2\end{bmatrix}\\ &= \begin{bmatrix}b_1x_2 - m_1 & 0\\ 0 & m_2 - b_2x_1\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix}\end{aligned}$$

To keep the population constant in each we need $b_1x_2 - m_1 = 0$ and $m_2 - b_2x_1 = 0$, thus

$$\begin{aligned}x(1) &= \frac{m_2}{b_2}\\ x(2) &= \frac{m_1}{b_1}\end{aligned}$$

We can implement the basic predator prey rate calculations by the Scilab function

```
function [xdot]=predator_prey_rate(starve,birth,eat,eaten,predator,prey)
    xdot=zeros(2,1);
    xdot(1)=(eat*prey-starve)*predator;
    xdot(2)=(birth-eaten*predator)*prey;
endfunction
```

To specialize this to our foxes and rabbits we could write the Scilab function

```
function [xdot]=fox_rabbit(t,x)
    starve=5;
    birth=10;
    eat=.01;
    eaten=.5;
    predator=x(1);
    prey=x(2);
    getf('predator_prey_rate.sce');
    xdot=predator_prey_rate(starve,birth,eat,eaten,predator,prey);
endfunction
```

Now we need to solve it, so we will use the function ode. The following code will set everything up for us and call ode.

```
getf('fox_rabbit.sce');
x=ode([22;500],0,[0:.01:10],fox_rabbit);
```

```
//set("figure_style","new") //create a figure
subplot(211)
   plot2d([0:.01:10],x')
   xtitle('Fox and Rabbit History','Years','Population');
subplot(212)
   plot2d(x(1,:),x(2,:))
   xtitle('Fox and Rabbit Interaction','Foxes','Rabbits');
```

When we run it we get the following graphic.

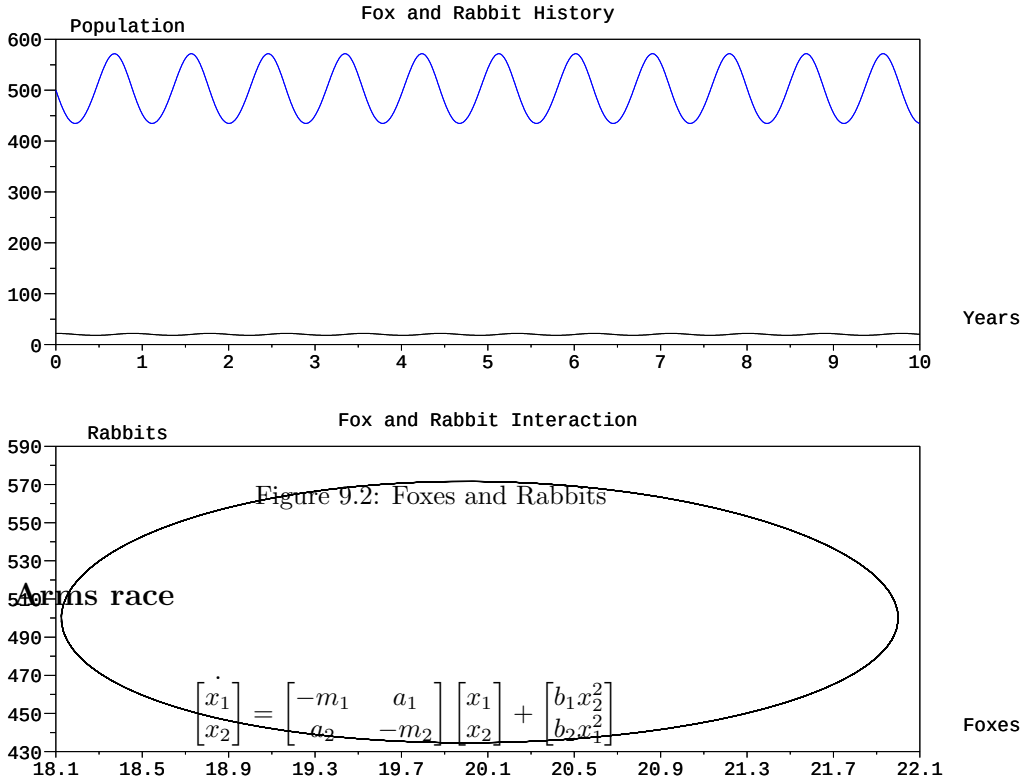


Figure 9.2: Foxes and Rabbits

9.0.5 Arms race

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \end{bmatrix} = \begin{bmatrix} -m_1 & a_1 \\ a_2 & -m_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} b_1 x_2^2 \\ b_2 x_1^2 \end{bmatrix}$$

Chapter 10

Partial Differential Equations

10.1 Basics

Consider the equation

$$f(x,t) = e^{kt}\sin(k\pi x). \tag{10.1}$$

Assume for the moment that x is constant and take the derivative with respect to t. We will use the notation of ∂ to denote a derivative with respect to only one variable, also known as a partial derivative.

$$\frac{\partial}{\partial t}f(x,t) = ke^{kt}\sin(k\pi x) \tag{10.2}$$

$$= f_t(x,t) \tag{10.3}$$

10.2 Wave in a String

Assume we have stretched a string to length L and fixed it at both ends. We want to find the function, $u(x,t)$, which describes the displacement of the string at any point. Now we will consider the basic one-dimensional wave equation,

$$\frac{1}{c^2}u_{tt} = u_{xx}. \tag{10.4}$$

We will try one of the basic techniques, called separation of variables. Basically we posit a solution of the form

$$u(x,t) = f(x)g(t). \tag{10.5}$$

Substitute this back into Eq. 10.4, we obtain

$$\frac{1}{c^2}f(x)g''(t) = f''(x)g(t) \tag{10.6}$$

$$\frac{-k^2}{c^2}f(x)g''(t) = -k^2 f''(x)g(t), \tag{10.7}$$

with k a constant, which does not effect the equality but does provide a useful term when we solve later. To maintain the independence between the x and t terms called for in the separation of variables technique it must be that the x terms on the left and right must be equal (similar for the t terms). Thus we have that

$$\begin{aligned} -k^2 f(x) &= f''(x) && (10.8)\\ \frac{1}{c^2} g''(t) &= -k^2 g(t), && (10.9) \end{aligned}$$

and thus

$$\begin{aligned} f''(x) + k^2 f(x) &= 0 && (10.10)\\ g''(t) + c^2 k^2 g(t) &= 0, && (10.11) \end{aligned}$$

where we have kept the constant k with the terms that have no derivative[1]. These are now ODE's, and we can now use the boundary conditions to solve them. We will use the fact that differential equations can be expanded in terms of exponentials, since the derivative of an exponential is an exponential. Second order differentials without a first order term are trigonometric functions[2].

First note that the boundary condition on the ends is that they are fixed. This means that

$$\begin{aligned} f(0) &= 0 && (10.12)\\ f(L) &= 0. && (10.13) \end{aligned}$$

This implies that

$$f(x) = \sum_{m=1}^{\infty} a_m \sin\left(\frac{2m\pi x}{L}\right). \quad (10.14)$$

We can now substitute Eq 10.14 into Eq 10.10.

$$\begin{aligned} -\sum_{m=1}^{\infty} a_m \left(\frac{2m\pi}{L}\right)^2 \sin\left(\frac{2m\pi x}{L}\right) + k^2 \sum_{m=1}^{\infty} a_m \sin\left(\frac{2m\pi x}{L}\right) &= 0 && (10.15)\\ \sum_{m=1}^{\infty} a_m \left(k^2 - \left(\frac{2m\pi}{L}\right)^2\right) \sin\left(\frac{2m\pi x}{L}\right) &= 0 && (10.16)\\ k &= \frac{2m\pi}{L} && (10.17) \end{aligned}$$

[1] This is a result of experience. Essentially doing this provides a constant to be solved for in each, and also couples the equations.

[2] This can be understood from the fact that the second derivative of sine or cosine is the negative of itself. It can also be seen from the fact that $e^{-j\phi} = \cos\phi + j\sin\phi$, where $j = \sqrt{-1}$.

10.2.1 Partial Discretization

Another way to solve this is to approximate the spatial derivative (discretizing in space), while leaving the time derivative alone. The result is an ODE that approximates the PDE. Say we use the basic higher order derivative formula we developed in the chapter on derivatives, with a step size of Δ, then

$$\begin{aligned}
\frac{1}{c^2}u_{tt} &= u_{xx} \\
\frac{1}{c^2}u_{tt}(t,x_i) &= \frac{u(t,x_{i+1}) - 2u(t,x_i) + u(t,x_{i-1})}{\Delta^2} \\
u''(t,x_i) &= \frac{-2c^2}{\Delta^2}u(t,x_i) + \frac{c^2}{\Delta^2}\left(u(t,x_{i+1}) + u(t,x_{i-1})\right).
\end{aligned}$$

The last line just underscores that u is now only a continuous function of t described by a simple ODE. In essence we have made a whole bunch of interconnected wave functions, one at each x_i. This can be solved by any of the methods in the chapter on differential equations.

Part II

Numerical Linear Algebra

Chapter 11

Vector Spaces

In this chapter I will give the background material needed to understand numerical linear algebra.

11.1 Basis

To understand what a basis is, you first need to understand orthogonality and independence. I will give you both some formal definitions, but more importantly I will try to make them understandable. Let's start with independence.

11.1.1 Independence

Say we have a group of vectors $v_1, \ldots, v_n$. We might want to know if there are any of them that are unnecessary, that is could we make it from a combination of the others. If no vectors are unnecessary, we say the vectors are independent. Formally, we write $v_1, \ldots, v_n$ are independent iff,

$$\sum_{i=1}^{n} \alpha_i v_i = 0 \implies \alpha_i = 0 \forall i \in \{1, \ldots, n\} \tag{11.1}$$

In English, this reads the vectors $v_1, \ldots, v_n$ are independent if and only if the sum of each vectors v_i times a scalar α_i for $i = 1$ to n is zero implies that all the α_i are zero. In simple terms there is no non-zero weighted combinations of the vectors that sums to zero, as this would mean we could write one vector in terms of the others by some algebra. The concept is not that tough but it can get a little messy when calculating, particularly when you are first introduced to the idea.

Let's do an example,

Example 1 *Are the following two dimensional vectors independent?*

$$v_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, v_2 = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \tag{11.2}$$

Solution: *Let's write out the linear combination used in the definition.*

$$\sum_{i=1}^{n} \alpha_i v_i = 0 \tag{11.3}$$

$$\alpha_1 v_1 + \alpha_2 v_2 = 0 \tag{11.4}$$

Now this implies two equations (one for each row of the vectors).

$$\alpha_1 + \alpha_2 = 0 \tag{11.5}$$

$$\alpha_2 = 0 \tag{11.6}$$

Equation 11.6 tells us that the second scalar weight, α_2 must be zero, which combined with Equation 11.5 requires the first scalar weight, α_1, to be zero. Thus by the definition, we have that the vectors are independent.

Now let's consider how many linearly independent vectors there could be, for a vector space of size n, where the size of the space is the number of elements in the vector. It turns out that we can have n of them. The easiest way to see this is to consider the vectors e_i, which are vectors with all zero entries except one, which is a 1 in the ith position. Trivially these vectors are independent (the summation in the definition results in n equations of $\alpha_i = 0$), so there are at least n independent vectors that span the space. Could we add one more and still have linear independence? Well consider the following.

1. The new vector cannot be all zeros or the α associated with it, say α_{n+1}, could be any number to satisfy the sum since the vector would be zero and $\alpha_{n+1} \cdot 0 = 0$.

2. The new vector, say d, must have more than one non-zero term, since if it only had one it would trivially be one of the e_i times a scalar equal to it's ith component (d_i) and thus as a duplicate, you would just have to pick $\alpha_i = -\alpha_{n+1} d_i$ to satisfy the summation without have all α_i zero.

3. An induction argument can be easily constructed now, by noting that adding one more non-zero component to d, say in position j, just requires setting $\alpha_j = -\alpha_{n+1} d_j$, to satisfy the sum without having all α_i zero.

Thus we can see that we cannot add another vector, and thus the maximum number of independent vectors is n. This is not yet a formal proof, many books include one, my purpose is to have you understand the what and the why parts. You are encouraged to study the formal proofs on your own.

One other term often used is a basis. A basis is a group of independent vectors that span a space (span means to cover all of it, and thus there will be a vector for each dimension in the space).

11.1.2 Orthogonality

To discuss orthogonality, I need to have an inner product. Mathematicians use the notation $< a, b >$ to denote the inner product of a and b. Physicists use $a \cdot b$ for the same thing, unless they are doing quantum, in which case they use $< a|b >$. In linear algebra you will often see this as $a^T b$ if the vectors are real, or $a^H b$ if they are complex[1]. I mention all these different ways of writing it as you might run into them in different contexts and you should realize they are the same. For linear algebra we just mean to multiply the corresponding elements of the matrix and sum them up.

$$a^T b = \sum_{i=1}^{n} a_i b_i \tag{11.7}$$

Just for your information this can be extended to infinite dimensional spaces like the space of continuous functions on the interval from 0 to 1 by converting the sum into an integral, thus yielding

$$< a(x), b(x) > = \int_0^1 a(x)b(x)dx \tag{11.8}$$

Getting back to our main point, which is given an inner product, we can find the angle between the vectors. How you ask? Well, it turns out that

$$a^T b = \|a\|_2 \|b\|_2 \cos\theta \tag{11.9}$$

Where $\|\cdot\|_2$ is the 2 norm (described in Section 11.2) and θ is the angle between the vectors a and b. The net result is that we can measure the angle between to vectors using the inner product. If the inner product is zero and the vectors aren't zero, that means that the cosine must be. For the cosine to be zero, the angle must be $\frac{\pi}{2}$ radians (or 90° if you prefer). Things that are $\frac{\pi}{2}$ radians apart are called orthogonal, so if the inner product of two vectors is zero, then the are orthogonal. A lot of our algorithms force use the inner product to measure the angle between two vectors, and then using this measure construct an vector from one of them, that is orthogonal to the other. If you understand this is what they are doing it makes the code much easier to read.

A neat side result is that if a group of vectors are mutually orthogonal, then they are also independent.

[1]Some old texts use $a^* b$ for complex vectors, but it means the same thing as $a^H b$. In any case it denotes transposing and taking the complex conjugate (this is the star notation), which is called the hermitian transpose (this is the H notation).

11.2 Vector Norms

11.2.1 Schatten P-Norms

The basic definition for some whole number, p, and vector, $x \in \mathbb{R}^n$, is

$$\|x\|_p = \sqrt[p]{\sum_{i=1}^{n} |x_i|^p}, \tag{11.10}$$

Three special cases are most important, $p = 1$, $p = 2$, and $p = \lim_{k\to\infty} k$.

Manhattan Norm

Also called the one norm and the taxicab norm, this norm is the case when $p = 1$.

$$\|x\|_1 = \sqrt[1]{\sum_{i=1}^{n} |x_i|^1} \tag{11.11}$$

$$= \sum_{i=1}^{n} |x_i| \tag{11.12}$$

It is readily observed that this is simply the sum of the absolute values of the elements of the vector.

Euclidian Norm

Also known as the two norm, this norm is what most people think of as distance.

$$\|x\|_2 = \sqrt[2]{\sum_{i=1}^{n} |x_i|^2} \tag{11.13}$$

$$= \sqrt{x^T x} \tag{11.14}$$

Infinity Norm

This norm has a simple form that is not obvious at first

$$\|x\|_\infty = \lim_{p\to\infty} \sqrt[p]{\sum_{i=1}^{n} |x_i|^p} \tag{11.15}$$

11.2.2 Equivalence

All the norms on a vector space are equivalent, by which we mean they for any two norms $\|\cdot\|_a$ and $\|\cdot\|_b$, and two scalars α and β, which depend only on the types of norms and

dimension of the space, we have $\alpha\|x\|_a \leq \|x\|_b \leq \beta\|x\|_a \forall x$ in the vector space. For instance, we have for the three most common Schatten P-Norms

	$\|x\|_1$	$\|x\|_2$	$\|x\|_\infty$
$\|x\|_1$	$\|x\|_1 \leq \|x\|_1 \leq \|x\|_1$	$\|x\|_2 \leq \|x\|_1 \leq \sqrt{n}\|x\|_2$	$\|x\|_\infty \leq \|x\|_1 \leq n\|x\|_\infty$
$\|x\|_2$	$\frac{1}{\sqrt{n}}\|x\|_1 \leq \|x\|_2 \leq \|x\|_1$	$\|x\|_2 \leq \|x\|_2 \leq \|x\|_2$	$\|x\|_\infty \leq \|x\|_2 \leq \sqrt{n}\|x\|_\infty$
$\|x\|_\infty$	$\frac{1}{n}\|x\|_1 \leq \|x\|_\infty \leq \|x\|_1$	$\frac{1}{\sqrt{n}}\|x\|_2 \leq \|x\|_\infty \leq \|x\|_2$	$\|x\|_\infty \leq \|x\|_\infty \leq \|x\|_\infty$

Note that the bounds are tight, which means that there are vectors x for which the equalities hold, though obviously not at the same time. For instance if x was an n-vector of ones, then $\|x\|_1 = n\|x\|_\infty$. Another example would be if $x = e_i$ were e_i is a vector of zeros, with a single 1 in the ith position, in which case all the p-norms yield the same value (1). One final note, the diagonal entries just say that the i norm is equal to itself, while remaining the structure of the rest of the entries.

Chapter 12

Matrix Preliminaries

Matrices are an essential tool in control systems design. First lets discuss some basic terms.

$\mathbb{R}$ The real numbers.

$\mathbb{R}^n$ The vectors composed of n-tuples of real numbers.

$\mathbb{R}^{m\times n}$ The matrices composed of m rows and n columns of real numbers.

12.1 Addition and Subtraction

In order to add or subtract matrices, the dimensions must be the same. Essentially we can add two $\mathbb{R}^{m\times n}$ matrices but not a $\mathbb{R}^{m\times n}$ and a $\mathbb{R}^{p\times r}$ matrix or even a $\mathbb{R}^{m\times n}$ and a $\mathbb{R}^{m\times m}$ matrix. Addition (or subtraction) is done by adding (or subtracting) the corresponding elements in the two matrices. For two $\mathbb{R}^{3\times 2}$ matrices addition is given by

$$\begin{bmatrix}1 & 2\\ 3 & 4\\ 5 & 6\end{bmatrix}+\begin{bmatrix}7 & 8\\ 9 & 10\\ 11 & 12\end{bmatrix}=\begin{bmatrix}1+7 & 2+8\\ 3+9 & 4+10\\ 5+11 & 6+12\end{bmatrix}=\begin{bmatrix}8 & 10\\ 12 & 14\\ 16 & 18\end{bmatrix} \tag{12.1}$$

For two $\mathbb{R}^{3\times 2}$ matrices subtraction is given by

$$\begin{bmatrix}1 & 2\\ 3 & 4\\ 5 & 6\end{bmatrix}-\begin{bmatrix}6 & 5\\ 4 & 3\\ 2 & 1\end{bmatrix}=\begin{bmatrix}1-6 & 2-5\\ 3-4 & 4-3\\ 5-2 & 6-1\end{bmatrix}=\begin{bmatrix}-5 & -3\\ -1 & 1\\ 3 & 5\end{bmatrix} \tag{12.2}$$

12.2 Multiplication

In order to do multiplication, we need to have matrices that are compatible to multiply. Recall that in order to add two matrices they had to be the same size. Unfortunately this is not the condition for multiplication. To be able to multiply a matrix A on the right by a

matrix B[1], we must have that the inner matrix dimensions are equal. That is, we require that $A \in \mathbb{R}^{m\times p}$ and $B \in \mathbb{R}^{p\times n}$, Where m, n, and p do not have to be the same, but they could. Thus we could multiply the following

- AB or BA with $\{A, B\} \in \mathbb{R}^{3\times 3}$;
- AB with $A \in \mathbb{R}^{3\times 3}$ and $B \in \mathbb{R}^{3\times 4}$;
- AB with $A \in \mathbb{R}^{2\times 3}$ and $B \in \mathbb{R}^{3\times 4}$.

Two general ways to do multiplication exist. Each has computational advantages in different situations. They give the same answer they are just different ways to group the solution.

12.2.1 Inner Product

A full study of the inner product is beyond the scope of this work, but interested students are referred to texts on Hilbert Spaces[2]. An inner product of two vectors of size n[3] a and b is given by $\langle a, b\rangle = a^T b = \sum_{i=1}^{n}(a_i b_i)$. Thus it is the sum of the product of corresponding elements in the vectors.

Example:

$$a = \begin{bmatrix}1 & 2 & 3\end{bmatrix}^T \qquad b = \begin{bmatrix}4 & 5 & 6\end{bmatrix}^T$$

Note that I have defined the vectors in the transposed form to save space. The transpose just means

$$\begin{bmatrix}1 & 2\end{bmatrix}^T = \begin{bmatrix}1\\2\end{bmatrix}. \tag{12.3}$$

Then the inner product of a and b is

$$\begin{aligned}\langle a, b\rangle &= \begin{bmatrix}1 & 2 & 3\end{bmatrix}\begin{bmatrix}4\\5\\6\end{bmatrix}\\ &= 1\times 4 + 2\times 5 + 3\times 6\\ &= 32\end{aligned}$$

Now try it in SciLab. First define a and b by

```
--> a=[1;2;3];
--> b=[4;5;6];
```

[1] Multiplying a matrix A on the right by a matrix B means AB, left multiplication by B would be BA. In matrices it is not true in general that $AB = BA$, or even that one can be calculated even if the other exists. This will make more sense in a few seconds.

[2] Hilbert Spaces are spaces with a defined inner product. They play an important role in control systems theory

[3] A vector of size n is the same as saying the vector is in $\mathbb{R}^n$.

Note that I ended each line with a ";", which tells SciLab to suppress its responses. This is vital when using the programming mode or SciLab will scroll out lots of unneeded info. With the vectors described we just need to take the inner product. To do this we use the $a^T b$ form, which is written

```
--> a'*b
```

The prime(') does the transpose (^{T}) and the multiply then does the inner product.

With vector inner products down we can consider two matrices, $A \in \mathbb{R}^{4\times 2}$ and $B \in \mathbb{R}^{2\times 3}$.

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \\ a_{3,1} & a_{3,2} \\ a_{4,1} & a_{4,2} \end{bmatrix} \qquad B = \begin{bmatrix} b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,1} & b_{2,2} & b_{2,3} \end{bmatrix}$$

We will consider partitioning the matrices into vectors that can be multiplied. We define

- The first row of A to be $a_1 = \begin{bmatrix} a_{1,1} & a_{1,2} \end{bmatrix}$.
- The second row of A to be $a_2 = \begin{bmatrix} a_{2,1} & a_{2,2} \end{bmatrix}$.
- The third row of A to be $a_3 = \begin{bmatrix} a_{3,1} & a_{3,2} \end{bmatrix}$.
- The fourth row of A to be $a_4 = \begin{bmatrix} a_{4,1} & a_{4,2} \end{bmatrix}$.
- The first column of B to be $b_1 = \begin{bmatrix} b_{1,1} & b_{2,1} \end{bmatrix}^T$.
- The second column of B to be $b_2 = \begin{bmatrix} b_{1,2} & b_{2,2} \end{bmatrix}^T$.
- The third column of B to be $b_3 = \begin{bmatrix} b_{1,3} & b_{2,3} \end{bmatrix}^T$.

then

$$A = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix} \qquad B = \begin{bmatrix} b_1 & b_2 & b_3 \end{bmatrix} \tag{12.4}$$

and so the product

$$\begin{aligned} AB &= \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix} \begin{bmatrix} b_1 & b_2 & b_3 \end{bmatrix} \\ &= \begin{bmatrix} a_1 b_1 & a_1 b_2 & a_1 b_3 \\ a_2 b_1 & a_2 b_2 & a_2 b_3 \\ a_3 b_1 & a_3 b_2 & a_3 b_3 \\ a_4 b_1 & a_4 b_2 & a_4 b_3 \end{bmatrix} \\ &= \begin{bmatrix} \sum_{i=1}^{2} a_{1,i} b_{i,1} & \sum_{i=1}^{2} a_{1,i} b_{i,2} & \sum_{i=1}^{2} a_{1,i} b_{i,3} \\ \sum_{i=1}^{2} a_{2,i} b_{i,1} & \sum_{i=1}^{2} a_{2,i} b_{i,2} & \sum_{i=1}^{2} a_{2,i} b_{i,3} \\ \sum_{i=1}^{2} a_{3,i} b_{i,1} & \sum_{i=1}^{2} a_{3,i} b_{i,2} & \sum_{i=1}^{2} a_{3,i} b_{i,3} \\ \sum_{i=1}^{2} a_{4,i} b_{i,1} & \sum_{i=1}^{2} a_{4,i} b_{i,2} & \sum_{i=1}^{2} a_{4,i} b_{i,3} \end{bmatrix} \end{aligned}$$

By hand the easiest way to do this is to line up the two matrices to multiply as follows

$$\begin{bmatrix} & & \\ & & \\ & & \\ & & \end{bmatrix} \begin{bmatrix} b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,1} & b_{2,2} & b_{2,3} \end{bmatrix}$$
$$\begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \\ a_{3,1} & a_{3,2} \\ a_{4,1} & a_{4,2} \end{bmatrix} \begin{bmatrix} \square & \square & \square \\ \square & \square & \square \\ \square & \square & \square \\ \square & \square & \square \end{bmatrix}$$

Then calculate each component by lining up the row and column and multiplying the corresponding terms in them.

$$\begin{bmatrix} & & \\ & & \\ & & \\ & & \end{bmatrix} \begin{bmatrix} b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,1} & b_{2,2} & b_{2,3} \end{bmatrix}$$
$$\begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \\ a_{3,1} & a_{3,2} \\ a_{4,1} & a_{4,2} \end{bmatrix} \begin{bmatrix} a_{1,1}b_{1,1} + a_{1,2}b_{2,1} & \square & \square \\ \square & \square & \square \\ \square & \square & \square \\ \square & \square & \square \end{bmatrix}$$

The second element is

$$\begin{bmatrix} & & \\ & & \\ & & \\ & & \end{bmatrix} \begin{bmatrix} b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,1} & b_{2,2} & b_{2,3} \end{bmatrix}$$
$$\begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \\ a_{3,1} & a_{3,2} \\ a_{4,1} & a_{4,2} \end{bmatrix} \begin{bmatrix} a_{1,1}b_{1,1} + a_{1,2}b_{2,1} & \square & \square \\ a_{2,1}b_{1,1} + a_{2,2}b_{2,1} & \square & \square \\ \square & \square & \square \\ \square & \square & \square \end{bmatrix}$$

The third element is

$$\begin{bmatrix} & & \\ & & \\ & & \\ & & \end{bmatrix} \begin{bmatrix} b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,1} & b_{2,2} & b_{2,3} \end{bmatrix}$$
$$\begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \\ a_{3,1} & a_{3,2} \\ a_{4,1} & a_{4,2} \end{bmatrix} \begin{bmatrix} a_{1,1}b_{1,1} + a_{1,2}b_{2,1} & \square & \square \\ a_{2,1}b_{1,1} + a_{2,2}b_{2,1} & \square & \square \\ a_{3,1}b_{1,1} + a_{3,2}b_{2,1} & \square & \square \\ \square & \square & \square \end{bmatrix}$$

The fourth element is

$$\begin{bmatrix} & & \\ & & \\ & & \\ & & \end{bmatrix} \begin{bmatrix} b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,1} & b_{2,2} & b_{2,3} \end{bmatrix}$$
$$\begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \\ a_{3,1} & a_{3,2} \\ a_{4,1} & a_{4,2} \end{bmatrix} \begin{bmatrix} a_{1,1}b_{1,1} + a_{1,2}b_{2,1} & \square & \square \\ a_{2,1}b_{1,1} + a_{2,2}b_{2,1} & \square & \square \\ a_{3,1}b_{1,1} + a_{3,2}b_{2,1} & \square & \square \\ a_{4,1}b_{1,1} + a_{4,2}b_{2,1} & \square & \square \end{bmatrix}$$

Then repeat the process till every box is filled.

12.3 Matrix Classification

12.3.1 Basic Properties

Property	Meaning
Diagonal	$A_{ij} = 0 \forall i \neq j$
Lower Triangular	$A_{ij} = 0 \forall i < j$
Upper Triangular	$A_{ij} = 0 \forall i > j$
Tridiagonal	$A_{ij} = 0 \forall \lvert i-j \rvert \leq 1$
Pentediagonal	$A_{ij} = 0 \forall \lvert i-j \rvert \leq 3$
Lower Hessenberg	$A_{ij} = 0 \forall i+1 > j$
Upper Hessenberg	$A_{ij} = 0 \forall i > j+1$
Symmetric	$A = A^T$
Normal	$AA^H = A^H A$
Idempotent	$A^2 = A$
Orthogonal	$A^{-1} = A^T$
Unitary	$A^{-1} = A^H$
Positive Definite ($A > 0$)	$x^H Ax > 0 \forall x \in \mathbb{C}^n$

12.3.2 Definite

Positive Definite

A matrix, $A \in \mathbb{C}^{n \times n}$ is defined as positive definite if $\forall x \in \mathbb{C}^n$, $x^H Ax > 0$. Usually we further restrict this to symmetric matrices, as they are what comes up in practice and they are easier to handle (more algorithms, easier proofs, etc.), but they are not the only matrices which fit the definition[4].

Positive definite matrices are non-singular, and thus invertible. In some sense they are the nicest matrices to work with.

Theorem 2 *Let A be any symmetric positive definite matrix of size n, with smallest singular value σ_n. Let q_1 and q_2 be two orthonormal n-vectors and $1 > \alpha > 0$ be a scalar. The matrix $A + n\alpha\sigma_n q_1 q_2^H$ is non-symmetric positive definite.*

Proof:

Let Q be an orthogonal matrix with the first two columns q_1 and q_2 respectively. Q forms a basis for the n-vectors, so $\forall y \in \mathbb{C}^n \exists x \in \mathbb{C}^n$, $y = Qx$. Since Q is unitary it is invertible, so it is 1-1 and onto, thus every x has a unique y and vice versa. Let the singular value decomposition of A be $U\Sigma U^H$. Now consider

$$\begin{aligned} y^H(A + n\alpha\sigma_n q_1 q_2^H)y &= y^H Ay + n\alpha\sigma_n y^H q_1 q_2^H y && (12.5)\\ &= x^H Q^H AQx + n\alpha\sigma_n x^H Q^H q_1 q_2^H Qx && (12.6)\\ &= x^H Q^H U\Sigma U^H Qx + n\alpha\sigma_n x_1 x_2 && (12.7)\\ &= x^H V^H \Sigma V x + n\alpha\sigma_n x_1 x_2 && (12.8) \end{aligned}$$

[4]It is worth noting that there are non-symmetric positive definite matrices. I have never seen anyone categorize how to create one, so I decided to do it in Theorem 2 on page 121.

for $V = U^H Q$. Note that $V^H \Sigma V$ must be symmetric positive definite, thus $x^H V^H \Sigma V x \geq \|x\|_2^2 \sigma_n$. If x_1 or x_2 are zero then it is trivial that $x^H V^H \Sigma V x + n\alpha\sigma_n x_1 x_2 > 0$, so it only remains to prove positive definiteness when both are not zero. That means

$$\begin{aligned} y^H(A + n\alpha\sigma_n q_1 q_2^H)y &\geq \|x\|_2^2\sigma_n + n\alpha\sigma_n x_1 x_2 && (12.9)\\ &\geq n\|x\|_\infty^2\sigma_n + n\alpha\sigma_n x_1 x_2 && (12.10)\\ &\geq n|x_1 x_2|\sigma_n + n\alpha\sigma_n x_1 x_2 && (12.11) \end{aligned}$$

Now factor this expression as follows

$$\begin{aligned} n|x_1||x_2|\sigma_n + n\alpha\sigma_n x_1 x_2 &= n|x_1||x_2|\sigma_n + n\alpha\sigma_n|x_1||x_2|sign(x_1 x_2) && (12.12)\\ &= (1 + \alpha sign(x_1 x_2))n|x_1||x_2|\sigma_n. && (12.13) \end{aligned}$$

Trivially, $n|x_1||x_2|\sigma_n > 0$, and $1+\alpha sign(x_1 x_2) > 0$, so $n|x_1||x_2|\sigma_n + n\alpha\sigma_n x_1 x_2 > 0$ and thus we have $y^H(A + n\alpha\sigma_n q_1 q_2^H)y > 0$. Since this holds for all y, we have that $A + n\alpha\sigma_n q_1 q_2^H$ is positive definite. The non-symmetry is a direct result of the structure.

$\diamondsuit$ SDG $\diamondsuit$

Appendix A

Using SciLab

A.1 Basics

When you first start SciLab you will see something like

```
                ==========
                 scilab-2.7.2
        Copyright (C) 1989-2003 INRIA/ENPC
                ==========



Startup execution:
  loading initial environment

-->
```

The arrow "−−>" is the command prompt. SciLab, like MatLab is a command line interface to a mathematics programming environment. To get started lets do a calculation.

```
3+(2+5*4)/11
```

SciLab performs the calculation and displays the answer.

```
 ans  =

    5.
```

Now lets define a simple variable.

```
a=2
```

SciLab responds with

```
 a  =

    2.
```

Notice anything similar? The response is almost the same but "ans" has been replaced by the variable name "a". In fact it is even more similar than that. When no assignment ("name=") is given, SciLab automatically assigns the result to the variable "ans". Try using it.

```
a*ans
```

SciLab will tell you that "ans" is now 10. Lets move on and define a matrix. Type the following

```
  A=[1,2;3,4;5,6]
```

and press enter. Commas are used to separate elements and semicolons are used to separate rows. Note that you could also have entered "A" using the alternate notation

```
  A=[[1 2];[3 4];[5 6]]
```

or even (command prompt shown so you won't think something is wrong when it automatically appears, also you do not need to space over like I do to enter the numbers, I just find it easier to read)

```
-->  A=[[1 2]
-->     [3 4]
-->     [5 6]]
```

Thus spaces work like commas and returns work like semicolons. In any case, SciLab should respond by showing you that it has created the matrix variable as follows

```
 A  =

!   1.    2. !
!   3.    4. !
!   5.    6. !
```

The variable "A" is now defined and can be used. For instance we might want to define "B" to be "A+A". Do this by typing

```
  B=A+A
```

SciLab will add the matrices and define "B" to be the result, showing you the answer.

```
 B  =

!   2.     4.  !
!   6.     8.  !
!   10.    12. !
```

This mode is useful for doing simple calculations and testing output. We will refer to it as the interactive mode. Since SciLab has an interactive mode that is command driven, it is reasonable to assume it would have a programming interface (we will refer to it as the programming mode). I will show the use of programming mode later.

A.2 Scilab and Matlab Programming

Scilab is a Matlab look alike. We will use Scilab as it is free and open source, but it is useful to also know about Matlab.

A.2.1 Matlab

There are two basic ways to interact with Matlab: command line execution, and M-files. Yes there are others such as MEX-files, Simulink, and several interfacing programs, but they are not relevant to us. We will primarily be concerned with the use of M-files, because they are the most helpful. Command line execution is really just for quick operations and checking of segments of code. Matlab syntax is a high level programming language that interacts with a series of numerical libraries (most notably LinPack, EisPack, and BLAS). Like most programming languages we have two types of programs that can be written. A regular program, which is written as you would type commands on the command line, is the most basic type and is often the way you will start homework problems and other projects. Functions, which are sub-programs called by another program (even recursively by other functions), are probably the most useful, as they allow you to extend the language by defining new operations. One of the main goals of this class is for you to walk away with a library of Matlab functions that you can use to do a variety of tasks. So how do you specify which you want? You will get a regular program unless you start the M-file with the command function. The syntax is

function a=name(x,y,..., z)

or

function [a,b,...,c]=name(x,y,..., z)

The second form returns multiple values. Matlab gives us several command structures also: for, while, and if-elseif-else. To see how these work let's use the programs I passed out last time as an example.

Homework: Convert the Fortran program in 3.1 into Matlab syntax. Do problems 9, 13, 14 from section 3.1

Bibliography

[1] S. J. Benbow. Solving Generalized Least-Squares Problems With LSQR. *SIMAX*, 21:165–177, 1999.

[2] Å. Björck. Iterative Refinement of Linear Least Squares Solutions I. *BIT*, 7:257–278, 1967.

[3] Å. Björck. Iterative Refinement of Linear Least Squares Solutions II. *BIT*, 8:8–30, 1968.

[4] Å. Björck. A General Updating Algorithm for Constrained Linear Least Squares Problems. *SISSC*, 5:394–402, 1984.

[5] Å. Björck. Stability Analysis of the Method of Seminormal Equations for Linear Least Squares Problems. *Linear Alg. and Its Applic.*, 88/89:31–48, 1987.

[6] Å. Björck. *Least Squares Methods: Handbook of Numerical Analysis*, volume 1. Elsiver, Holland, 1988.

[7] Å. Björck. *Numerical Methods for Least Squares Problems.* SIAM, Philadelphia, PA, 1996.

[8] Å. Björck and G. H. Golub. Iterative Refinement of Linear Least Squares Solutions by Householder Transformation. *BIT*, 7:322–337, 1967.

[9] S. Boyd and L. Vandenberghe. Advances in Convex Optimization: Theory, Algorithms, and Applications. In *IEEE International Symposium on Information Theory.* IEEE, July 2002. Talk.

[10] S. Boyd and L. Vandenberghe. Convex Optimization. `http://www.stanford.edu/~boyd/cvxbook.html`, December 2002.

[11] S. Chandrasekaran, G. H. Golub, M. Gu, and A. H. Sayed. Parameter Estimation in the Presence of Bounded Data Uncertainties. *SIMAX*, 19(1):235–252, 1998.

[12] S. Chandrasekaran, G. H. Golub, M. Gu, and A. H. Sayed. An Efficient Algorithm for a Bounded Errors-in-Variables Model. *SIMAX*, 20(4):839–859, 1999.

[13] A. K. Cline. An Elimination Method for the Solution of Linear Least Squares Problems. *SINUM*, 10:283–289, 1973.

[14] M. G. Cox. The Least Squares Solution of Overdetermined Linear Equations Having Gand or Augmented Band Structure. *IMA J. Numer. Anal.*, 1:3–22, 1981.

[15] J. Cullum. Ill-posed Deconvolutions: Regularization and Singular Value Decompositions. In *Proceedings 19th IEEE Conference on Decision and Control*, volume 1, pages 29–35, 1980.

[16] G. Cybenko. The Numerical Stability of the Lattice Algorithm for Least Squares Linear Prediction Problems. *BIT*, 24:441–455, 1984.

[17] I. S. Duff. Pivot Selection and Row Ordering in Givens Reduction on Sparse Matrices. *Computing*, 13:239–248, 1974.

[18] M. P. Ekstrom and R. L. Rhoads. On the Application of Eigenvector Expansions to Numerical Deconvolutions. *J. of Comp. Phys.*, 14:395–417, 1974.

[19] L. Eldèn. Algorithms for the Regularization of Ill-Conditioned Least Squares Problems. *BIT*, 17:134–145, 1977.

[20] L. Eldèn. Perturbation Theory for the Least Squares Problem with Linear Equality Constraints. *SINUM*, 17:338–350, 1980.

[21] L. Eldèn. A Weighted Pseudoinverse, Generalize Singular Values, and Constrained Least Squares Problems. *BIT*, 22:487–502, 1983.

[22] L. Eldèn. An Algorithm for the Regularization of Ill-Conditioned, Banded Least Squares Problems. *SISSC*, 5:237–254, 1984.

[23] L. Eldèn. A Note on the Computation of the Generalized Cross-Validation Function for Ill-Conditioned Least Squares Problems. *BIT*, 24:467–472, 1985.

[24] H. W. Engl and H. Gfrerer. A Posteriori Parameter Choice for General Regulariztion Methods for Solving Linear Ill-posed Problems. *Appl. Numer. Math.*, 4:395–417, 1988.

[25] G. E. Forsythe and G. H. Golub. On the Stationary Values of a Second-Degree Polynomial on the Unit Sphere. *SIAM J. App. Math.*, 14:1050–1068, 1965.

[26] J. N. Franklin. Minimum Principles for Ill-posed Problems. *SIAM J. Math. Anal.*, 9:638–650, 1978.

[27] W. Gander. Least Squares with a Quadratic Constraint. *Numer. Math.*, 36:291–307, 1981.

[28] C. F. Gauss and G. W. Stewart. *Theory of the Combination of Observations Least Subject to Errors.* SIAM, Philadelphia, PA, 1995.

[29] J. A. George and M. T. Heath. Solution of Sparse Linear Least Squares Problems Using Givens Rotations. *Lin. Alg. and Its Applic.*, 34:69–83, 1980.

[30] H. Gfrerer. An A Posteriori Parameter Choice for Ordinary and Iterated Tikhonov Regularization of Ill-Posed Problems Leading to Optimal Convergence Rates. *Math. Comp.*, 49:507–522, 1987.

[31] L. El Ghaoui and G. Galafiore. Robust Filtering for Discrete-Time Systems with Bounded Noise and Parametric Uncertainty. *ITAC*, 46:1084–1089, 2001.

[32] L. El Ghaoui. and H. Lebret. Robust Solutions to Least-Squares Problems with Uncertain Data. *SIMAX*, 18(4):1035–1064, 1997.

[33] G. H. Golub, P. C. Hansen, and D. P. O'Leary. Tikhonov Regularization and Total Least Squares. *SIMAX*, 30(1):185–194, 1999.

[34] G. H. Golub, M. Heath, and G. Wahba. Generalized Cross-Validation as a Method for Chosing a Good Ridge Parameter. *Technometrics*, 21:215–223, 1979.

[35] G. H. Golub and C. F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, Md, 1996.

[36] G. H. Golub and J. H. Wilkinson. Note on Iterative Refinement of Least Squares Solutions. *Numerical Math.*, 9:139–148, 1966.

[37] M. Hanke and T. Raus. A General Heuristic for Choosing the Regularization Parameter in Ill-Posed Problems. *SIAM J. Sci. Comput.*, 7:956–972, 1996.

[38] P. C. Hansen. Analysis of Discrete Ill-posed Problems by Means of the L-curve. *Siam Review*, 34:561–580, 1992.

[39] P. C. Hansen. *Rank-Deficient and Discrete Ill-Posed Problems*. SIAM, Philadelphia, PA, 1998.

[40] A. E. Hoerl and R. W. Kennard. Ridge Regression: Biased Estimation for Nonorthogonal Problems. *Technometrics*, 12:55–67, 1970.

[41] P. D. Hough and S. A. Vavasis. Complete Orthogonal Decomposition For Weighted Least Squares. *SIMAX*, 18:369–392, 1997.

[42] M. E. Kilmer and D. P. O'Leary. Choosing Regularization Parameters in Iterative Methods for Ill-posed Problems. *SIMAX*, 22:1204–1221, 2001.

[43] S. Kourouklis and C. C. Paige. A Constrained Least Squares Approach to the General Gauss-Markov Linear Model. *J. Amer Stat. Assoc.*, 76:620–625, 1981.

[44] C. L. Lawson and R. J. Hanson. *Solving Least Squares Problems*. SIAM, Philadelphia, PA, 1995.

[45] H. Lebret and S. Boyd. Antenna Array Pattern Synthesis Via Convex Optimization. *IEEE Transactions on Signal Processing*, 45(3):526–532, March 1997.

[46] M. Lobo, L. Vandenberghe, S. Boyd, and H. Lebret. Applications of Second-Order Cone Programming. *Linear Algebra and its Applications*, 284:193–228, November 1998. Special Issue on Linear Algebra in Control, Signals and Image Processing.

[47] R. Lorenz and S. Boyd. Robust Minimum Variance Beamforming. Submitted to IEEE Transactions on Signal Processing, October 2001.

[48] D. A. McQuarrie and P. A. Rock. *General Chemistry*. W. H. Freeman and Company, New York, NY, 1987.

[49] J. M. Mendel. *Lessons in Estimation Theory for Signal Processing, Communications, and Control*. Prentice Hall, Englewood Cliffs, NJ, 1995.

[50] V. A. Morozov. On the Solution of Functional Equations by the Method of Regularization. *Soviet Math. Dokl.*, 7:414–417, 1966. cited in [39].

[51] A. Neubauer and O. Scherzer. Regularization for Curve Representations: Uniform Convergence for Discontinuous Solutions of Ill-posed Problems. *SIAM J. Appl. Math.*, 58:1891–1900, 1998.

[52] D. P. O'Leary. Near-Optimal Parameters for Tikhonov and Other Regularization Methods. Technical report, CS Dept., University of Maryland, 1999. CS-TR-4004.

[53] D. P. O'Leary and J. A. Simmons. A Bidiaonalization-Regularization Procedure for Large Scale Discreizations of Ill-Posed Problems. *SISSC*, 2:474–489, 1981.

[54] C. C. Paige. Computer Solution and Perturbation Analysis of Generalized Least Squares Problems. *Math. Comp.*, 33:171–184, 1979.

[55] C. C. Paige. Fast Numerically Stable Computations for Generalized Linear Least Squares Problems. *SINUM*, 16:165–171, 1979.

[56] C. C. Paige. The General Limit Model and the Generalized Singular Value Decomposition. *Lin. Alg. and Its Aplic.*, 70:269–284, 1985.

[57] G. Peters and J. H. Wilkinson. The Least Squares Problem and Pseudo-Inverses. *Comp. J.*, 13:309–316, 1970.

[58] J. E. Pierce and B. W. Rust. Constrained Least Squares Interval Estimation. *SIAM Jour. Sci. Stat. Comput.*, 6:670–683, 1985.

[59] T. Raus. The Principle of the Residual in the Solution of Ill-posed Problems with Nonselfadjoint Operator. *Uchen. Zap. Tartu Gos. Univ.*, 715:12–20, 1985. In Russian. Referenced in [37].

[60] B. W. Rust. Truncating the Singular Value Decomposition for Ill-posed Problems. Technical report, National Institute of Standards and Technology, U.S. Dept. of Commerce, 1998. Tech. Report NISTIR 6131.

[61] K. Schittkowski and J. Stoer. A Factorization Method for the Solution of Constrained Linear Least Squares Problems Allowing for Subsequent Data Changes. *Numer. Math.*, 31:431–463, 1979.

[62] G. W. Stewart. On the Asymptotic Behavior of Scaled Singular Value and QR Decompositions. *Math. Comp.*, 43:483–490, 1984.

[63] G. W. Stewart. On Scaled Projections and Pseudoinverses. *Linear Algebra Appl.*, 112:189–193, 1989.

[64] G. Strang. A Framework for Equilibrium Equations. *SIREV*, 30:283–297, 1988.

[65] M. J. Todd. A Dantzig-Wolfe-Like Variant of Karmarkar's Interior-Point Linear Programming Algorithm. *Oper. Res.*, 38:1006–1018, 1990.

[66] S. A. Vavasis. Stable Numerical Algorithms For Equilibrium Systems. *SIMAX*, 15:1108–1131, 1994.

[67] G. A. Watson. Data Fitting Problems with Bounded Uncertainties in the Data. *SIMAX*, 22(4):1274–1293, 2001.