Contents

mm FORTH M 9 Linear Algebra Linear Algebra :ontonts

§1 Simultaneous linear equations 551 Theory of linear algebraic equations §§2 Eigenvalue problems 52 Solving linear equations §§1 Cramer's rule §§2 Pivotal Elimination 553 Testing §§4 Implementing pivotal elimination §§5 The program 556 Timing 53 Matrix inversion §§1 Linear transformations 5552 Matrix multiplication §§3 Matrix inversion §§4 Why invert matrices, anyway? §§5 An example

54 LU decomposition 214 214 215 218 218 221 225 227 227 242 242 243 244 245 245 246 213 214

51 Simultaneous linear equations

§§1 Theory oi linear algebraic equations

ChapterB-LhearNgebra Sewncronr'

Two common problems in scientific programming are the numerical solution of simultaneous linear algebraic equation and computing the inverse of a given square matrix. This is sue". an important subject As an exercise in FORTH program development we shall now write programs to solve linear equations and invert matrices.

Simultaneous linear equations

We begin with a dose of mathematics (linear algebra), and then develop programs. Several actual programming session are reproduced *in toto* – mistakes and all– to give the flavor of program development and debugging 1n FORTH.

Theory oi linear algebraic equations

We begin by stating the problem. Given the equations

$$\sum_{n=0}^{N-1} A_{mn} x_n = b_m \ . \tag{1}$$

– more compactly written $\vec{A} \cdot \vec{x} = \vec{b}$– with known coefficient matrix A_{mn} and known inhomogeneous term b_m: under what conditions can we find unique values of x_n that simultaneoust satisfy the N equations 1?

The theory of simultaneous linear equations tells us that if not all the b_m 's are 0, the necessary and sufficient condition for solvability is that the determinant of the matrix $\vec{A}$ should not be 0. Contrariwise, if $det(\vec{A}) = 0$, a solution with $\vec{x} \neq 0$ *can* be found when $\vec{b} = 0$.

Eigenvalue problems

Many physical systems can be represented by systems of linear equat'ons. Masses on springs, pendula, electrical circuits, structures , and molecules are examples. Such systems often can oscillate sinusoidally. If the amplitude of oscillation remains bounded, such motions are called stable. Conversely, sometimes the motions of physical systems are unbounded the amplitude of any small disturbance will increase exponentially with time. An example is a pencil balanced on its point. Exponentially growing motions are called -for obvious reasons - unstable.

Clearly it can be vital to know whether a system is stable or unstable. If stable, we want to know its possible frequencies of free oscillation; whereas for unstable systems we want to know how rapidly disturbances increase in magnitude. Both these problems can be expressed as the question: do linear equations of the form

K-i'= Air-if (2)

have solutions? Here A is generally a complex number, mllcd the eigenvalue (or characteristic value) of Eq. 2, and fi'is often called the mass matrix. Frequently K is the unit matrix I,

1, m=n I''''-0,m=tn, (3)

but in any case, I must be positive-definite (we define this below). A nontrivial solution, x is 0, of the equation

K-ii'=o (4)

exists if and only if "I" = o . This fact is useful in solving eigen- value problems such as Eq. 2 above.

2.

eJuanvNoueiasa-Almm.

216 CMpterO-LklearNgabi-a Sclenwlcronnd

The secular equation (or determinantal equation)

IIIm=o (5)

is a polynomial of degree N inl, hence has N roots (either real ori co.mplex) When p'= T, these roots are called the eigenvalues (or' 'characteristic values") of the matrix K

Eigenvalue problems arising in physical contexts usually involve a restricted class of matrices, called real-symmetric or Hermitiant

(after the French mathematician Hermite) matrices, for which.

A;,, =Am . (The superscript . denotes complex conjugation −_ :

see Ch. 7.) All the eigenvalues of Hermitian matrices are and: numbers. How do we know? We simply consider Eq. 3 and its complex conjugate:

2AM x,— = 112 pm xn ; (6a)

2 x:A:'m = 1' 2 x: pi. ; (6b) Equation 6b can be rewritten (using the fact that K and flare. Hermitian)

2 MIA"... = 1' 2 x; Pm (6b') Multiply 6b' by xm and 63 by x; , sum both over at and subtract: this gives

0= (ifi) 2 x;p.,,x,. (7)

However, as noted above, p is positive-definite, Le.

3. This follows from the fundamental theorem of algebra: a polynomial equation, I p(z): a0 + all + azzz +.. .+ anz" =0, of degree n (in a complex variable 2) has exactly a solutions ,

WM"!

QuailWM 217

lt-p-XIZ x,',,p,,..x,,¿0 (8)

I." for any non-zero vector4 x .Thus, Eq. 7 o A. E A, that is, l is real.

ln vibration problems, the eigenvalue A usually stands for the square of the (angular) vibration frequency:)1 2 (oz. Thus, a posi- tive eigenvalue). corresponds to a (double) real value, 1w, of the angular frequenq. Real frequencies correspond to sinusoidal vibration with time-dependence sin(alr) or cos(wi).

Conversely, a negative). corresponds to an imaginary frequency, ti al and hence to a solution that grows exponentially in time, as

sin(l' (0!) = isinh(wr) (9)

0050' air) = cosh(alt)

There are many techniques for finding eigenvalues of matrices. If only the largest few are needed, the simplest method is itera-

tion: make an initial guess x (0) and let

Axol-l)

,(n) =

Assuming the largest eigenvalue is unique,the sequence of vec-

tors xm , n = 1, 2,..., is guaranteed to converge to the vector corresponding to that eigenvalue, usually after just a few itera- lions.

Ifall the eigenvalues are wanted, then the only choice is to solve the secular equation 5 for all N roots.

52 Solving linear equations

The test-and-development cycle in FORTH is short compared with most languages. It is usually easy to create a working program that subsequently can be tuned for speed, again much

4. Fordaritywenowonitthevcaor"-'anddyad"0'symbdsfi'omvectorsandmauiwe

emvuouoim-me.

218

ChapterQ-LlnaarNgebra SclentlflcFORmi

more rapidly than with other languages. For me this is the chief 1 benefit of the language.

"I Cramer's rule

Cramer's rule is a constructive method for solving linear equa

tions by computing determinants. It is completely impractical as a computer algorithm because it requires 0(N!) steps to solve N linear equations, whereas pivotal elimination (that we look at below) requires 0(N3) steps, a much smaller number. Neverthe less Cramer's rule is of theoretical interest because it is a closed form solution.

Consider a square NXN matrix A Pretend for the moment we' know how to compute the determinant of an (N-1)x(N-1)' matrix. The determinant of A is defined to be

Nl det(A) = 2 A...,,a,,,,

n-O

where the amn's are called co-factors of the matrix elementsA... and are in fact determinants of specially selected (N-1)x(N-1) sub-matrices of A. '

The sub-matrices are chosen by striking out of A the n'th column and m'th row (leaving an (N-1)X(N-1) matrix). To illustrate, consider the 3 X3 matrix

1 A = a (11) 1

4M0

and produce the co-factor of A 12:

.21-(-1)2+'

1 §5 (12) 1 6

We also attach the factor (-l)"'+" to the determinant of the submatn'x when we compute a". .

K's-DO

A determinant changes sign when any two rows or any two

columns are interchanged. Thus, a determinant with two iden- tical rows or columns is exactly zeros. What would happen to Eq. 11 if instead of putting A," in the sum we put A" where krtm? By inspection we realize that this is the same as evaluating a determinant in which two rows are the same, hence we get zero. Thus Eq. 11 can be rewritten more generally

Nl 20AM anrn = llA ll 6km - (13)

This feature of determinants lets us solve the linear equation A 'x = b by construction: try

1 N-l

xii = detm E: anmbm- (14)

We see from Eq. 13 that Eq. 14 solves the equation. Equation 14 also makes clear why the solution cannot be found if det(A) = 0.

A determinant also vanishes when a row is a linear combination of any of the other rows. Suppose row 0 can be written

5.

Theonlynumberequaltoitsnegatiwiso.

OJulanVNobletflz-Alrldusmarvcd.

Chapter 9 Linear Algebra Scientific FORTH

N-l "or E 2 fins amt;

m=i

that is, the O'th equation can be derived from the other N-l equations, hence it contains no new information. We do not really have N equations for N unknowns, but at most N-1 equations. The N unknowns therefore cannot be completely specified, and the determinant tells us this by vanishing.

As an example, we now use Cramer's rule to evaluate the deter- minant of Eq. 11. We will write

——A——=Aooaoo+Amaio+Aozazo 24 in,"

4 mini-v

The determinant of a 1 x 1 matrix is just the matrix element, hence

32 1

320: 1

em = 2-6 + (1)-4-1 = 8 am = (18 4) = 14 320 = (3 2) = 1 "A II = 1-8 + 0-(14)+ 5-1 =13. How many operations does it take to evaluate a determinant? We

see that a determinant of order N requires N determinants of order N-l to be evaluated, as well as N multiplications and N-l

WWW-l

Glaciers-Wm 221

additions. If the addition time plus the multiplication time is r, then

TN N(r+T,.-,).

It is easy to seem the solution to this is

- Nire.

" 1 TN=leE 3" "an

In other words, the time required to solve N linear equations by Cramer's rule increases so rapidly with N as to render the method thoroughly impractical.

§§2 PlvotalEllmlnatlon

The algorithm we shall use for solving linear equations is elimina- tion, just as we were taught in high school algebra. However we modify it to take into account the experience of 40 years's solving linear equations on digital computers (not to mention a previous 20 years's worth on mechanical calculators!), to minimize the buildup of round-off error and consequent loss of precision .

The necessary additional step involves pivoting selecting the largest element in a given column to normalize all the other

6. Professors always say this, hee, heel See R. Sedgewick, Algorithm (AddisomWesky Publishing Company, Reading. MA 1983).

5"3.'I

N! means Nx(N-1)x(N-2)x... x2x1. The base ofthe "natural" logarithmsis e =Z7182318... Acomputerstoresanumberwithfinite precision say6-7decimalplaceswith32-bit floating-

pointnumbers.'l'hisisenoughformanypurposes,upcdaflyinscienceandengineedngwhere thedataare rarely measured to betterthan 1numbers, about lO'zinmagnitude, are multiplied. Their productisoforder lO'and'sknownto aixsignificantfigures.Nowaddittoathirdnumberoforderunit number:10".l..ater,afourthnumber alsooforderunity issubtractedfromth'nsum'lhe resultwillbeanumberoforderlo'mutnowknownonlytot'esignificantfigunMatrixarith meticisfullofmultiplicationsandadditiouaThelessonisdear tominimizethefinevitable) kndwedshmassodatedwihmundvomwmwwmkcepthemayfludesdprodwsand sum-sasekiseupossible.

222

Chapter 9 - Linear Algebra Sclermflc FORTH

elements. This will be clearer with a concrete illustration rather than further description: Consider the 3 X3 system of equations:

105 x0 0 324 1:1 =4 (15) 116 x2 2

We check that the determinant is #0; in fact det(A) = 13. The first step in solving these equations is to transpose rows in A and in b to bring the largest element in the first column to the/loo position.

Natl-a;

e The x's are not relabeled by this transposition. ,

e We choose the row (n=1 second row) with the largest (in absolute value) element Ano because we are eventually going to divide by it, and want to minimize the accumulation of roundoff error in the floating point arithmetic.

Transposition gives 3 2 4 xo 4 1 0 5 x1 = 0 (16) 1 1 6 x2 2

Now divide row 0 by the neono (in this case, 3) to get 1 2/3 4'3 x0 V3 1 0 5 x1 = 0 (17) 1 1 6 X2 2

Subtract row 0 times Ana from rows with n ¿ 0 1 2/3 4': 10 V: o 2/, 11/3 x, = -Va (13) 0 1'3 W: x, 26

Since —An— ¿ —A21— we do not bother to switch rows 1 and 2, but divide row 1 byAn =-2/3, getting

cranes-mm 223

l '3 V: to V3 0 1 -1V2 xi - 2 (19) 0 V) IV! 12 V3

We now multiply row 1 by A" a 1/3 and subtract it from row 2, and also divide through by A" to get

173 V3 10 V3 0 1 -u/2 x; 8 2 (20) 0 0 1 x; 0

The resulting transformed system of equations now has 0's to the left and below the principal diagonal, and 1's along the diagonal. Its solution is almost trivial, as can be seen by actually writing out the equations:

1Xo+2/3X— +V3X2=4/3 (21.0) 0x0 +1x1+(11/2)X2 = 2 (21.1) Oxo+0x1+1x2=0 (21.2)

That is, from Eq. 21.2, x; = 0. We can back-substitute this in 21.1, then solve for x, to get

x, = 2 (11/2)-(0) = 2

and similarly, from 21.0 we find xo=V32/3(2)+V3(0)=O. We test to, see whether this is the correct solution by direct trial: 1-O+0-2+5-O=0 3-0+2-2 +4-0=4 1-O+2-2+6-O=2

his works we have indeed found the solution.

cmvlioueieaa-Alrlgium.

224

Chapter 9 - Unear Algebra Scientific FORTH

How much time is needed to perform pivotal elimination? We concentrate on the terms that dominate as N - on.

The pivot has to be found once for each row; this takes N-k comparisons for the k'th row. Thus we make N l2 comparisons of 2 real numbers. (For complex matrices we compare the squared moduli of 2 complex numbers, requiring two multiplica- tions and an addition for each modulus.)

We have to divide the k'th (pivot) row by the pivot, at a point in I the calculation when the row contains N-k elements that have not ¯ been reduced to 0. We have to do this for k = 0, 1, ..., N-l, '

requiring =N /2 divisions.

The back-substitution requires 0 steps for rm, 1 multiplication and 1 addition for x", 2 each for it)", etc. That is, it requires

k=0 2 (Nlk) =N2/2 k=N-1

multiplications and additions.

The really time-consuming step is multiplying the k'th row by A", , j ¿ k, and subtracting it from row j. Each such step requires N-k multiplications and subtractions, for j=k+1 to N-l, or (N -k) - (N k 1) multiplications and subtractions. This has to be repeated for k= 0 to N-2, giving approximately N3/3 multiplica-

l l l i i ; i

tions and subtracti30ns. In other words, the leading contribution l

to the time is rN 3/3, which is a lot better than reN! as with Cramer' 5 rule.

Whenweog timize for speed, only the innermost loop re uir-

mg O (N13 /3) operations needs careful tuning; the O (N [2) 3

operations comparing floating point numbers, dividing by the .i

pivot, and back-substituting need not be optimized bgecause for large N they are overshadowed by the innermost loop .

9.

The exception to this general rule, where more complete optimization would pay, would be an application that requires solving many sets of equations of relatively small order.

MM"!

"3

cleansmm 225

Mia

Since we have worked out a specific 3 x 3 set of linear equations, we might as well use it for testing and debugging. Begin with some test words that do the following jobs:

e Create matrix and vector (inhomogeneous term) e Initialize them to the values in the example

e Display the matrix at any stage of the calculation e Display inhomogeneous term at any stage

```
\ Display words
V{ or M{{ stands torwhatanarrayputsonstack
U: G. F. x. ;
:.M (M{{--) FINIT DUP
D.LEN 0 DO CR DUP
D.LEN 0 DO DUP (--M{{ M{{ )
J}{I}} DUP>R G@ G.
LOOP
LOOP DROP ;

:.v (V{-˜) DUP D.LEN
0 DO on DUP l}{0} G@ G. LOOPDROP:

We define a word to put ASCII numbers into matrices:
:GEFF# BL TEXT PAD S->F;
```

This word takes the next string10 from the input stream (set off by ASCII blank, BL) and uses the PIS/FORTH word S ¿ F to convert the string to floating point format and put it on the 87stack.

10. 'l'heword'lEX'l'inputsacountedstringusingtheFORTH-DstandardwordWOIlD,aad placesit"PADJ'hisiswhyPADappeaninthedefinifionofGfl-FlAdefinkionofEXT mid-itbe :'IEX'l'(delirniter–)WORDDUPC@1+ unswxrcuovs; Notethatall words pref-acedwithC arebyteoperatioasbyPORTH convention.

OJilthNoUetm-Mlldusleseived.

Chapter 9 - Unear Algebra Sclentlflc FORTH

```
GET-F# is used in the following:

: <DO-VEC> ODO GET-F# DUP I0} G!
LOOP DROP;

:TAB->VEC DUP D.LEN <DO-VEC> ;

:TAB->MAT DUP D.LEN DUP * <DO-VEC> ;
```

The file EX.3 will be found in the accompanying program disk- ette. The explanatory notes below refer to EX.3.

Notes:

e The word .(emits everything up to a terminating) .

e HS/FORT'I-I, because of its segmented dictionary, uses a word TASK to define a task-name, and FORGET-TASK to FORGET eve hing following the task-name. The FORTH word FOR- GE fails in HS/FORTH when it has to reach too deeply into the dictionary.

e Ex.3 uses the word }row{ defined below.

This is what it looks like when we load EX.3:

FLOAD EX.3 Loading EX.3 ReadadimenslonSvectorand 3113 matrbtlror-natablethendispiaythem Now displaying vector. V .V CR 0.0000000 4.0000000 2.0000000 Now displayhg matrbc A .M CR 1.0000000 0.000(1)") SIXJOOCDO 3.0000000 2.0000000 4.0000000 1.0000000 1.0000000 6.0000000 say Exa FORGET-TASKto graceitfly FORGET these words olt

WM"!

Mil-Wm 227

"4 WWW

Now we have to define the FORTH words that will implement this algo-rithm. In pseudooode, we can express pivotal elimina- tion as shown in Fig. 9-1 below.

ch

A B SOLVE (solutiontoAxsaleltinB) BEGIN

setn=0

findpivot(rowm ¿ n)

swaprowsman

divide row n by pivot: A'M. =AMJAM

subtract: A'... =A..A..,.A',.. increment n n N = UNTIL Back-substitute: x" = b'n 2 Am... xm(12)

m¿n

DONE

Fig. 9-1 Pseudocode for pivotal ellmlnaa'on

"5 The program

Whenever we write a complex program we are faced with the problem "Where do we begin?" FORTH emphasizes the con-

struction of components. and generally cares little which component we define first.

For example. we must swap two rows of a matrix. The most obvious way moves data:

row1 - ¿ temp row2 - ¿ row1

temp -¿ row2

Although data moves are relatively fast, up to N2/2 swaps may be needed, each row containing N data; row-swapping is an O (N 3) operation which must be avoided. lndirectio setting up a list of row pointers and exchanging those. uses 0 (N) time.

OWVNobletmAlmm.

228

Chapter 9 - Llnear Algebra Scientific FOR'n-l

We anticipate computing the address of an element of A in a , DO loop via the phrase i

AJI (Jis row,lis col)

Suppose we have an array of row pointers and a word row that looks them up. Then the (swapped) element would be

A J row I .

To implement this syntax we define11

:swapper (n - -) l CREATE 0 DO I , LOOP DOES¿ OVER + + @ ; this is a defining word

We would define the array of row indices via

A D.LEN swapper row create array row

We have initilized the vector row by filling it with integers from 0 to N-l while defining it.

Suppose we wanted to re-initialize the currently vectored row- 5 index array: this is accomplished via

A D.LEN 'row refill .i

where

ramm. ,. . .m'. .

: refill (n adr - -) SWAP 0 DO I 2* DDUP + ! 2 +LOOP DROP;

11.

mm"... 'mnLk,fi-u_ .. aims . Vannr

The HS/FORTH words VAR and IS define a data structure that can be changed, like a FORTH VARIABLE: 0 VAR X 3 IS X but has the run-time behavior of a CONSTANT: X . 3 olt.

```
:ADRS >R 2'OVER + SWAPR> 2' +;
(amn-a+2ma+2n)
OVAR ?SWAP \to keep track of swaps

:}SWAP (amn-)DDUP -
IF DDROP DROP \noswap - deanup

ELSE ADRS DDUP (.-121 2)
@SWAP@ (-'12[21[1I)
nor! SWAP 1

-1 ?SWAP XOR IS ?SWAP \ "' ?SWAP
THEN ;

Test this with a 3-dimensional row-index array:

3 swapper }row{

:TEST ODOI}row{ CR. LOOP;
3 TEST

0

1

2 ok

Now swap rows 1 and 2:

'}row{ 1 2 }SWAP 3TEST
o

2
10k

and back again:

'}row{ 1 2 }SWAP 3 TEST
0

1
20k
```

Next, we need a word to find the pivot element in a given column, searching from a given n. The steps in this procedure are shown in Fig. 9-2 on page 230 below.

We anticipate using generic operations Git defined in Chapter 5

to perform fetch, store and other useful manipulations on the matrix elements, without specifying their type until mn-time. lt

```
:A{{ col# }}PIVOT

N col# 1+ DO



Yes IF
No 1 Put larger on fstack
set I.PIV = I
t__l THEN
. LOOP
; \ exit




M{{ | }{ 00"" }}I > M{{ col# }{ 00h" }}I 7
```

Fig. 9-2 Pseudocode for finding the pivot element

is thus useful to have a place to store the type (other than the second cell of the array data structure). We arrange this via

OVART

The rest of the definition is rendered self-explanatory by the vertical commenting style (a must for long12 words):

```
OVAR Length \Iength of matrix
```

```
OVAR COL \currentcol#
OVAR I.PIV
OVAR a{{
: INITIALIZE (M{{ -- ::--)
IS a{{
a type@ IS T
a LEN@ IS Length ;

\ |.PIV is used to return the result
\ a{{ stores the adress of M{{

\ initialize T

\ length - > Length
```

12. While Brodie (11", p.180) quotes Charles Moore. FORTH's inventor, as offering l-line defini- tions as the goal in FORTH programming, sometimes this is just not possible.

i

a place to store data type

i

```
2))PIVOT (M{{ col-- ::--)

IS COL ZECOL IS LPN

INITIAU

fl: COL row COL}} (--aog.ofl[M{% oot,ool}}]t)

> 8 SF >F (87:--|1st.olt)

R> COL1+ \Sloop|infgLCOL+1

DO I .
a{{ I }row{ COL }} (- - seg.ofl[M{flI,col}}] t)
>FS GABS FS>F(87:  |old.olt Ineweltl)
xoup F< \test' newett > olden
IF FSWAP IISI.PIV THEN FDROP

LOOP \ondloop

FDROP ; \deanupfstack
```

```
\ Usage: A{{ 2 "PIVOT
```

Mates:

0 We avoid filling us: the parameter stack with addresses that have to be Eagle , by putting arguments in named storage locations (J). We anticifiate setting all the parameters in the beginning using IN! 21". which can be extended later if necessary.

0 We have used a word from the COMPLEX arithmetic lexicon, namely XDUP(FOVER FOVER) to double-copy the contents

of the fstack, since the test F¡ drops both arguments and leaves a flag.

0 There is little to be gained by optimizing (and if we did we should have had to avoid the generic Gx words because they cannot be optimized by the PIS/FORTH recursive-descent o timizer) because only? /2 time is used, ne ligible (for large

compared with the [3 in the innermost oop.

The Gx operations are found in the file GLIBJ-TH.

To test the word "PIVOT we can add some lines to the test file:

oncn .(weammmmo.1.2)

on

```
G! .(A{{O})PIVOT mv .) BL EMIT A{{0)}PNOT IPN.
on .(A{{1})PNOT IPIV .) BL em A{{1}}PNOT IPN .
on .(A{{2})PIVOT IPN .) BL am A({2}}PNOT new.
```

one:

The result is the additional screen output

```
A{{ o }}PIVOT IPIV 1
A{{1}}P|VOT IPIV 1
A{{ 2 }}PIVOT IPIV 2
```

OR

From our pseudocode expression (see Fig. 9-1, p. 227) of the pivotal elimination algorithm we see that the next operation is to multiply a row by a constant. To do this we define

```
OVAR ISTEP
\ include phrase

T #BYTES DROP IS ISTEP
\ in INITIALIZE
```

```
: DO(ROW*X)LOOP (seg off Lfin I.beg - - :: x - - x)
DO DDUP I + DDUP T >FS G'NP TFS>
ISTEP /LOOP DDROP ;

\ G*NP means "(generic) multiply, no pop"

:}}ROW*X M{{ row - :: x  x)
UNDER }row{ 0 A (- - r seg offt)
DROP ROT (- -seg off r)
ISTEP * Length ISTEP * SWAP \Ioop indices
DO(ROW*X)LOOP ; \Ioop

\Ex: A{{ 2 }ROW*X
```

Mates:

0 While DO(ROW'X)LO0P and ??ROW'X clear the param- eter stack in approved FORTH fashion, they leave the constant multiplier x on the fstack. That is, we antimpate next multipl - ing the corresponding row of the inhomogeneous term b (in the matrix equation A-x = b) by the same constant x.

e We assume INITIALIZE (includin INITJSTEP) has been invoked before "PIVOT or "110 'X.

o Anticipating the (possible) need to optimize, we factor the loop right out of the word. We also com me the addresses fast by putting base addresses on the stac and then incrementing them by the cell-size, in bytes.

Subtracting row 1 (times a constant) from row I is quite similar to multiplying a row by a constant. The process can be broken down into the pseudocoded actions in Fig, 9-3 below.

```
 no (K-JTOL)
x e@

M[I,K] e@ G'NP (::- - x M[l,K]'X)
M{J,K] 6@ GR- (::--XM[J,K]-X'M[I,K])
M{J.K] Gl

LOOP
```

Fig. "Steps In nm-rwx

A first attempt might look as follows:

```
\auxiliarywords

:4DUP DOVER DOVER ;

(abcd--abcdabcd)

: + + (51.0152.02n   1.01 +n 52.02+n)
DUP>R + ROT R> + -ROT;

:}}R1-R2"X (r1r2-- ::x--x)

> R > R

a R> row 0 DROP \M r10
aila@imioli one. \Miiaoli
Length iSTEP " \ L'lSTEP (upper limit)

R> ISTEP * \ r2'ISTEP (lower limit)

DO 4DUP I ++ \duplicateaincbaseadrs

T >FS G'NP (2:  xx*M[r2,k])
DDUP T >FS GR-
T >FS \ I result
ISTEP [LOOP DDROP DDROP ;
\ INITIALIZE is assumed
```

As with HROW'X we avoid the execution-speed penalty of calculating addresses within the loop by not using the phrase

However, we are doing a lot of unnecessary work inside the loop by making 5 choices based on datatype. There are also a lot , unnecessary moves to/from the ifstack. This is an example where speed has been sacrificed to compactness of code: one program solves equations of any ("scientific") data format REAL'4, REAL-8, COMPLEX'8 and COMPIJEX' 16. i

Conversely, we can both eliminate the extra work and accelerate? execution simply by defining a separate inner loop for each data; type, letting the choice take place once, outside the inner loopX This multiplies the needed code fourfold (or more-fold, if we? optimize).

I

The 4 inner loops are

```
: Re.LP (\S1.01 52.02 L'istep r2*istep - - :2 x - - x)
FS>F >R
DO 4DUP I + + R32@L F*NP
DDUP R32@L FR- R32IL
4 /LOOP R> F>FS ;

: DRe.LP ($1.01 $2.02 L'istep r2'istep - - z: x - - x)
FS>F >R
DO 4DUP l + + R64@L F*NP
DDUP R64@L FR- R64!L
8 /LOOP R> F>FS ;

: X.LP (51.01 52.02 L*istep r2*istep - - z: x - - x)
FS>F >R
DO 4DUP l + + CP@L X'NP
DDUP CP@L CPR- CPIL
8 /LOOP R> F>FS ;

: DX.LP (\S1.01 52.02 L'istep r2'istep - - :: x - - x)
FS>F >R
DO 4DUP I + + DCP@L X'NP
DDUP DCP@L CPR- DCPIL
16 /LOOP R> F>FS;
```

Not surprisingly, the loops contain some duplicate code. but this- ' is a small price to pay for the speed increase. A significant furthe r3

increase can be obtained easily. using the HS/FOR'I'H recursive- descent optimizer to define these inner loops, or by redefining the loops in assembler (for ultimate speed).

Now we can redefine }}Rl-RZ'X using vectored execution, via HS/FORI'H's CASE: . .. :CASE construct, or the equivalent high- level version given here:

```
: CASE: CREATE
DOES> OVER + + @ EXECUTE ;
: ;CASE [COMPILE] ; ; IMMEDIATE

CASE: DO(R1-R2"X)LOOP
Re.LP DRe.LP X.LP DX.LP ;CASE

2}}R1-R2'X (r1 r2-- ::x--x)

>R >R

2112a mm; 328: {main
Length I.STEP * \ ULSTEP (upper limit)
R> I.STEP ' \ r2'I.STEP (lower limit)
```

```
T DO(R1-R2'X)LOOP DDROP DDROP ;
\ We assume INITIALIZE has been invoked

Another word is needed to perform the same manipulation on
the inhomogeneous term. Since this latter process runs in 0(N )
time we need not concern ourselves unduly with efficiency.

:}V1V2*X (r1 r2-- ::x--)
SWAP >R >R
b{ 0.0 } DROP DDUP \V[0] V[O]
R> irowilSTEP' + >FS G'
R> row ISTEP " + DDUP >FS GR- FS> ;
```

Hats:

0 After }Vl-VZ'X executes, the multiplier x is no longer needed, so we drop it here by using 6' rather than G'NP.

We now combine the words }SWAP, }}PIVOT, }}ROW'X. }Rl-RZ'X and }Vl-VZ'X to implement the triangularization por- tion of pivotal elimination.

Since the Gaussian elimination method makes it very easy to compute the determinant of the matrix as we go, we might as well do so, especially as it is only an 0(N) process. By evaluating the detemiinants of simple cases, we realize the determinant is simp- ly the product of all the pivot elements, multiplied by

(_1) swaps. Hence we need to keep track of swaps, as well as to multiply the determinant (at a given stage) by the next pivot element. The need to do these things has been anticipated in defining }SWAP above.

Computing the determinant also lets us test as we go, that the equations can be solved: if at any stage the determinant is zero, (at least) two of the equations must have been equivalent. Should it be necessary to test the condition of the equations, this too can be found as we proceed, by computing the determinant.

Here is a simple recipe for computing the determinant, with checking to be sure it does not vanish identically. We use the intelligent fstack (ifstack) defined in Ch. 7.

```
DCOMPLEX SCALAR DET \ room for any type
:INIT\_DEr T 'DEI' I \set Type
TG=1 DEI' GI; \setdet=

%1.E-10 FCONSTANT CONDITION

:DETERMINANT (- - 2: x - - x)
DET >FS G*NP
?SWAP IF GNEGATE THEN
```

```
FS.DUP GABS FS>F CONDITION F<
ABOR'l" determinant too small!' DET FS> ;
```

13.

The condition of a system of linear equations refers to how accurately they can be solved. Equa- tions can be hard to solve precisely if the inverse matrix A" has some large elements. Thus, the error vector for a calculated solution, d= -b-A szlc can be small, but the difference between the exact solution and the calculated solution can be large, since (see §9§3 below)

xBar-ct 'cnc =A W6 A test for ill-condition is whether the determinant gets small compared with the precision of the arithmetic. See, 43.3., A. Ralston,A First Course in Numerical Analysis (McGraw-Hill Book Co., New York, 1965).

Now we define the word that triangularizes the matrix:

```
0VARb( \tokeepthesteokehort
: INITIAUZE (M{(V{o- ::--)

le

lSa

3 .TYPE IS T

a D.LEN is Length

a D.TYPE #BYTES DROP ISISTEP

iN .DET ; \setdet=1

:}/PIVOT }row{) DROP DDUP T >FS G' TFS>;
(segofIt--::x--)

:TRIANGULAFIIZE (M{{ V{ - -)

INITIAUZE
Length 0 DO \ loop 1 - by rows
a{{l} PIVOT \findpivotincoil
' )row I l.PlV }SWAP \ exchange rows
a{{ I }row{ I }} >FS \pivot->iistack
DETERMINANT \ mic det
1/6 a{{ i }}ROW'X \row i Ipivot
b{ l }/PIVOT \inhom. term /pivot
Length l1+ DO \ioop2-byrows
```

```
a l}row{J}} >FS \x->ifstack
a lJ}}R1-R2"X

\ row[i] = row[i]-row[j]'x
b{ I J }V1-V2*X
\ same for b{ and drop x
LOOP \ and loop 2
LOOP ; \end loop 1
\Usege: A{{ B{ TRIANGULARIZE
```

Now at last we can back-solve the triangularized equations to find

the unknown x 's. The word for this is BACK-SOLVE, defined as follows:

```
: }BACK-SOLVE (- - )
0 LENGTH 2- DO \ outer loop
T G=0
LENGTH I 1 + DO \ inner loop
a? J }row{ I }} > PS
b I }row{ } >FS G*G+\
LOOP \ inner loop
b{ l ){ } DROP DDUP T >FS \
GR- T FS > \
1 +LOOP ; \outer loop J
```

Putting the entire program together we have the linear equation solver given in the file SOLVEl. Examples of solving dense 3x3 and 4 X4 systems are included for testing purposes.

§§6 Timing

e should like to know how much time it will take to solve a

given system. (Of course it is also useful to know whether the solution is correct!) We time the solution of 4' sets of N equations, with 4 different values of N. The running time can be expressed as a cubic polynomial in N with undetermined coefficients:

TN=ao+a1N1+azN2+agN3 (19)

Evaluating 19 for 4 different values of N, and measuring the 4 times TNi , we produce 4 inhomogeneous linear equations in 4

unknowns: a, , i = 0, 1, 2, 3 . As luck would have it, we just hap-

pen to have on hand a linear equation solver, and are thus in a position to determine these coefficients numerically.

For this timing and testing chore we need an exactly soluble dense system of linear equations, of arbitrary order. The simplest siqch system involves a rank-1 matrix, that is, a matrix of the form :

14.

We employ the standard notation that a vector u is a column and an adjoint vector VIf ’3 a row. Their outer product, uvi, is a matrix. Given a column v, we construct its adjoint (row) VIt by taking the complex conjugate of each element and placing it in the corresponding position in a

row-vector.

GustavoWNW 23.

A - I - u v’ (20) In terms of matrix elements,

Ai-du-uw’ (20’)

The solution of the system A 1 I: b is simple: using the standard notation

vi-x a (v, x) = 2 v. ’x. (21) we have X. = D: ’i’ (V, X) U. (22)

The coefficient (v, x) is just a (generally complex) number; we compute it from 22 via

(V. X) = (V. Ii) + (V. u) (V. X) (23) Ol’ ˍ (v, b) (V. X) 1 ˍ (v. u) . (24)

x, = o, +

Everythingin 25 is determined, hence the solution is known in closed form.

An example that embodies the above idea is given in the file EX.20, where we make the special choices

Chapter 9 Unear Algebra Scientific FORTH

N even

GAO-s .s 01

We give the times only for the case of a highly optimized inner loop (written in assembler), for the type REAL’4:

Table 9-1 Execution times for various N (9.54 MHz 8086 + 8087 machine)

N Time 20 1.20 50 7.75 75 19.6 100 38.8

The coefficients of the cubic polynomial extracted from the above are (in seconds, 9.54 MHz 8086 machine, machine-coded inner

loop)

a0 = 032727304 a. = -0.01084850 32 = 000241636 33 = 000001539

from them we can extrapolate the time to solve a 350x350 real system to be about 16 minutes. On a 25 MHz 80386/8038’7 ma- chine with 32-bit addrgsing implemented, the time should decrease five- to tenfold .

It is interesting to explore a bit further the extracted value of a3,

. . 1 which is 3- fined above as Re.LP and hand-coded in assembler for ultimate speed). Converting to clock cycles on an IBM-PC compatible (running at 9.54 MHz) we have an average of 440 clock cycles per traversal (of this loop). Recall the operations that are needed: a 32-bit memory fetch, a floating-point multiply, another 32-bit memory fetch, a floating-point subtraction, and a 32-bit store. The initial fetch-and-multiply can be be compressed into a single co-processor operation, as can the fetch-and-subtract. The times in cycles for these basic operations are

the time needed to evaluate the innermost loop (de-

TaUe92Executiontlmesotlnnennostioopoperntions

Operation lime (cpu clock5) memory FMUL 133 memory FSUBR 128 memory FSTP 100

overhead 61

Total 422

15. thnlthinkthatsolvinglmxlflhystems keytomyPh.D.researchin1966 tookthebetter putofanhornonanlnumnheseresuhsseeminaedible.

16. See WP.

§3 Matrix Inversion

§§1 Unear transformations

ChapterO-LlnaarAlgebra SciandflcFOfl

We see from Table 9-2 above that the time computed from t cpu and co-processor specifications (422 clocks) is close to measured time (440 clocks). The slight difference doub comes both from measurement error and from the fact th timing of CISC chips is not an exact art (for example, there a periodic interruptions for dynamic memory refreshment and f the system clock).

Finally, we confirm that little is to be gained by optimizing out loops. Suppose, 5.3., we could halve a; by clever programmin then we should cut the N =350 time from 16 to 13 minutes, the time for N = 1000 by some 20 minutes in 5 hours, or 6

We introduce this subject with a brief discourse on liners algebra of square matrices.

Suppose A is a square (NXN) matrix and x is an N-dimensiomi vector (a column, or NXI matrix). We can think of the symbolic operation

y= A -x (26)

as a linear transformation of the column x to a new column y. In terms of matrix elements and components,

Nl Ym= E Amnxn (27)

n-o

The transformation is linear because if x is the sum of 2 columns (added component by component)

it = x") + is), (23) _

l

we can calculate A 1 either by first adding the two vectors ant: then transforming, written

A-x-A-(am+am) (29)

or we could transform first and then add the transformed vectors. The identity of the results is called the distribatlvs law

A-x-A-(xm+a(2)) "amuse (30) of linear transformations.

"2 Matrix m Now, suppose we had several square matrices, A, B, 0,... We could imagine performing successive linear transformations on a

vectorxvia y= A -x (31a) 2= e -y (31b) w = C -1 (31c)

These can conveniently be written

w=c-[B-(A-x) . (32)

The concept of successive transformations leads to the idea of multiplying two matrices to obtain a third:

D=B-A E=C-D (33) In terms of matrix elements we have, for example N-I Diir ' 1

The important point is that the (matrix) multiplications may be performed in any order, so long as the left-to-right ordering of the factors is maintained:

oJilthNobinm-Alnohtaraaawad.

Ci'iaptoro-LInaarAigabra Salem

c- (a -A) -=- (c .n) -A (35)

Equation 35 is known as the associative law of matrix multipli tion. Finally, we note that as hinted above the left-to-ri order of factors in matrix multiplication is significant. That is, i general,

A-B¢B-A (36) 1

We say that, unlike with ordinary or even complex arithme ' matrix multiplication even of s uare matrices does not i general obey the commutative law .

§§3 Matrix Inversion With this introduction, what does it mean to invert a matrix? First of all, the concept can apply only to a square matrix. Given an NXN matrix A, we seek another N XN matrix A with theproper- ty that

A”-AaA-A”=I (37)

where I is the unit matrix defined in the beginning of this chapte r (Eq. 4 1’s on the main diagonal, 0’s everywhere else). We have: implied in Eq. 37 that a matrix that is an inverse with respect to left-multiplication is also an inverse with respect to right-multi- plication. Put another way, we imply that

(r’)” a A (38)

The condition that - given A we can construct A’1 is the same.— as the condition that we should be able to solve the linear equa-(tion

A-I-b;

t namely, det(A) u 0.

5’4 W invert matrices, W We have developed a linear equation solver program already why should we be interested in a matrix inversion program?

Here is why: The time needed to solve a single system (that is, with a given inhomogeneous term) of linear equations is condi- tioned by the number of floating-point multiplications required:

about 143/111): number needed to invert the matrix is about N3, roughly 3X as many, meaning roughly 3x as long to invert as to solve, if the matrix is large. Clearly there is no advantage to inverting unless we want to solve a number of equations with the same coefficient matrix but with different inhomogeneous terms. In this case, we can write

x=A’Lb a”

and just recalculate for each h. Clearly this breaks even relative to solving 3 sets of equations for 3 different b’s and is superior to re-solving for more than 3 Us

s55 Anmmpie

Let us now calculate the inverse of our 3x3 matrix from before. The equation is (let c = A ” be the inverse”)

105 000301 can 100 3 2 4 C10 C” C12 = 0 1 0 (40) 116 CanCa 001

18. Note:ifCisaright-inverse,AC-l,itkalsoa—dl-inverse,CA=l.

OJUIUIVNanim-Alrldasraaarvad.

246

CMptarO-Urngebra SclentlflcFORT

It is easy to see that Eq. 40 is like 3 linear equations, the unkn being each column of the matrix C. The brute-force method I calculate A is then to work on the right-hand-side all at once we triangularize, and back-solve. It is easy to see that gularization by pivotal elimination leads to

1 2/3 Vs can cat can 0 V3 0 0 1 *'1/2 C10 Ctr C12 = '3': V2 0 (41) 0 O 1 C20 C12 022 V13 "V13 13 l

results of back-solving, column by column. This is N times a.

if

1 total time required for brute-force inversion is =5 gN3. By keep

To construct the inverse we replace the right-hand side with thq

much work as back-solving a single set of equations, hence th

ing track of zero elements we could reduce this time by anothel elimination) of 0(N3). However, the brute-force method is un- satisfactory for another reason: it takes twice the storage of more sophisticated algorithms. Modern matrix packages therefore "8:" LU decomposition for both linear equations and matrix inver

sion.

§4 LU decomposition

We now investigate the LU decomposition algorithm". Sup- pose a given matrix A could be rewritten

awam 1m 0 0 lmflm A: 310311 = L'U = 1401110 0 11 (42) 0 0

then the solution of

19.

Sec, 2.3., WH. Press, B.P. F'Iannery, SA. Teukolsky and W.T. Vetterling. Numerical Recipe: (Cambridge University Press, Cambridge, 1986), p. 31H.

owe-mm 241

(L-u)-nL-(u-x) -b (43) can be found in two steps: First. solve Lty- b (44) for y: U-x (45) via looYo = bo MOYO'l'ltIYI =01 (46) 130% + 421Y1+ la)': = D: etc.

which can be solved successively by forward substtution. Next solve 45 successively (by back-substitution) for x:

l'NtN-t XN1 = YNt I'M-2 N-2 XN-z + I'M2 N-1 xNi = YNz (47) etc.

The n'th term of Eq. 46 requires n multiplications and n additions. Since we must sum n from 0 to N-l, we find N(N1)/2 =4 Nz/Z multiplications and additions to solve all of 46. Similarly, solving

47 requires about N2/2 additions and multiplications. Thus, the dominant time in solving must be the time to decompose accord-

ing to Eq. 42.

The decomposition time is =5 N 3/3, and the method for decom- posing is described clearly in Numerical Recipes. The equations to be solved are

N-l

[a luk/'h =Araa (48)

constituting N2 + N equations for N2 unknowns. Thus we may arbitrarily choose I... = 1.

eJuunMNobmm-nmm.

ChaptersUnearNgobra ScbndflcFORTM

The equations 48 are easy to solve if we do so in a sensible ordefi Clearly,

1mk50,m¿k il (49) "knaos k¡n

so we can divide up the work as follows: for each 11, write

ml .umn =Amn kzo Amkflkn ' m = 0,1,...,'l

(50)

n-l 1m =i(A..... 2 Amp") , m =n+1,n+2,...,N-1 -. .unn i=0

Inspection of Eq. 50 makes clear that the terms on the right side! are always computed before they are needed. We can store tln computed elements A." and uh, in place of the corresponding elements of the original matrix (on the diagonals we store 4..

since A", = 1 is known).

To limit roundoff error we again pivot, which amounts to permutl ing so the row with the largest diagonal element is the currenf one. Much of the code developed for the Gauss eliminatios method is applicable, as the file LU.FI'H shows.