Numerical Error and Migration

Bert Jacobs

Abstract

Finite difference migration routines require the solution of a sparse system of linear equations. If a bad algorithm for finding this solution is used then large errors will be introduced into the migration. In the worst of cases, these errors can be large enough to ruin the stability of an otherwise stable problem. This can be prevented by using Gaussian elimination with partial pivoting.

Numerical Error at Low Frequencies

In the course of an attempt to use a "bullet-proof" pentadiagonal imaging procedure a mysterious instability was discovered. Since such routines are guaranteed on mathematical grounds to be stable it was concluded that the problem was numerical. The problem disappeared when either the lowest temporal frequencies were ignored or when a different linear equation solver was used. It was therefore concluded that the problem lay in the use of a complex, pentadiagonal version of the routine *tri* which can be found in *FGDP*.

Estimates of Relative Error

At any given z-step and for a given frequency ω, part of the process of downward continuation of a wavefield involves the solution of a banded, complex system of linear equations. If the matrix of coefficients is denoted by A and we are given a known vector b then we are called upon to solve for x in a system $Ax = b$. Usually x cannot be found in a finite precision computational machine. The computed result can be considered to be the sum of x and a perturbation δx.

To quantify the error requires the introduction of a norm. A convenient one is the 2-norm. We denote the 2-norm of the complex vector x by

$$\|x\| = (x^H x)^{1/2}$$

where x^H is the Hermitian conjugate transpose of x.

Relative error is a bit tricky, since we might consider either $\|\delta x\| / \|x\|$ or $\|\delta x / x\|$. In this paper the former will be used, so that

$$E_{rel} = \frac{\|\delta x\|}{\|x\|}$$

This leaves us with the problem of getting good estimates of the true values of x and δx. One way of doing this which works most of the time is to compute estimates of x with both double precision and single precision linear equation solvers. Denoting the two estimates by x_d and x_s, respectively, then an estimate of E_{rel} can be had by implementing

$$E_{rel}(\omega) = \frac{\|x_s(\omega) - x_d(\omega)\|}{\|x_d(\omega)\|}$$

This was done within a downward continuation procedure so that the error estimates would be reasonable ones for seismic problems. The estimates are not the upper bounds usually considered in numerical analysis.

A Comparison of Linear Equation Methods

Four linear equation solvers were tested for their suitability as part of a pentadiagonal (two higher in order than the 45-degree equation) downward continuation algorithm. One of the linear equation subroutines tested was the subroutine *pentc* from SEP-26, page 180. Gaussian elimination for banded, complex matrices was implemented as well. Algorithms of this type, with and without partial pivoting, were described by Thorson in SEP-20. Complex, banded Householder elimination was also tested, with an algorithm adapted from Stewart (1973). It is found in Appendix A.

The relative estimates for these four algorithms are presented in Figure 1. In all four cases, the only significant errors were made in frequencies below 8 Hertz. Commonly, such frequencies are not propagated, but it is worthwhile to design our algorithms so that they are insensitive to choices of an input parameter such as cutoff frequency.

Householder elimination is more expensive and has larger relative errors at low frequencies than any other of the methods examined. This is probably because it involves a larger number of computations than its competitors. Its use in downward continuation algorithms is not recommended. Gaussian elimination without pivoting and *pentc* behave roughly the same. Since *pentc* is a form of LU decomposition which does not employ pivoting, this conclusion is not surprising. Gaussian elimination with partial pivoting has better error behavior at low frequencies than the other three algorithms. The improvement is not that spectacular, but it is enough to stabilize the downward continuation of the data which caused explosions when *pentc* was used instead.

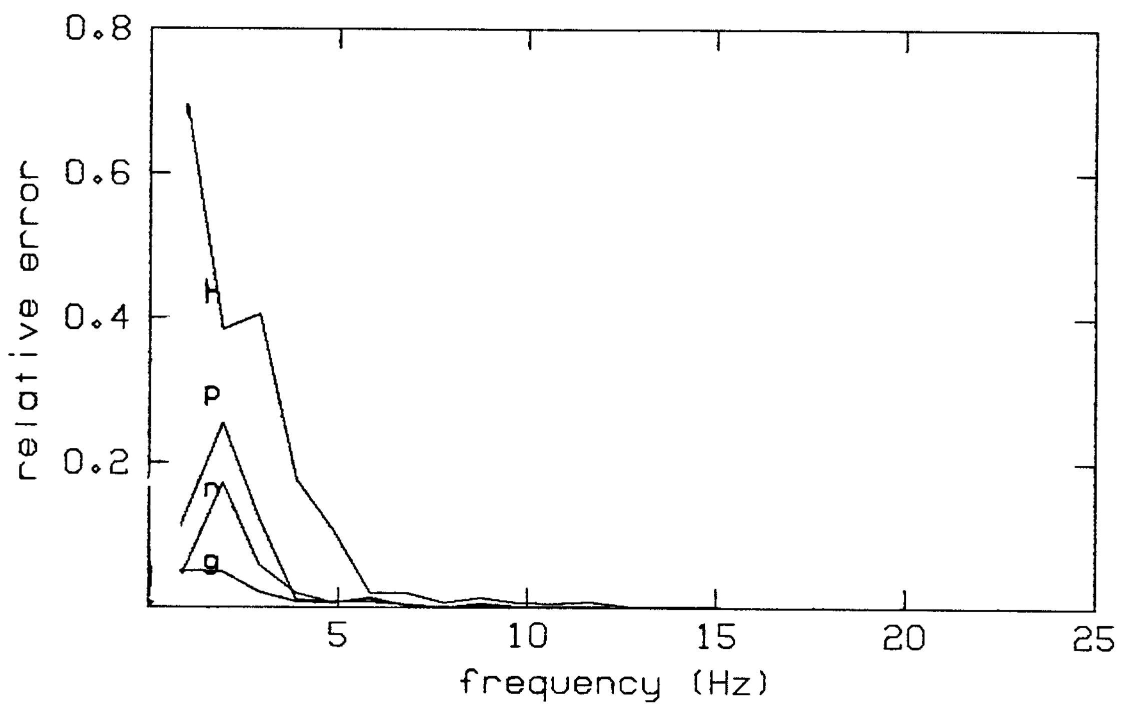

FIG. 1. Relative error versus temporal frequency for a downward continuation problem. Only the first 26 frequencies after DC are plotted. Zero frequency is the blank at the left margin. For curve (h), the linear equation solver was Householder elimination. For curve (p), the subroutine *pentc* from SEP-26. Curve (n) was generated by a Gaussian elimination routine with the pivoting option turned off. Curve (g) is the result of employing Gaussian elimination with partial pivoting.

The bad behavior of Householder's method was a little unexpected in that it's relative error under small perturbations in the input data is more tightly bounded than its competitors. One step in the algorithm which might be expected to cause problem is the scaling employed to prevent overflow and underflow. Removal of the loops which implemented the scaling has some influence on the the relative error estimates for Householder elimination, but the improvement is not large enough to make it competitive with Gaussian elimination with partial pivoting.

REFERENCES

Stewart, G.W., 1973, Introduction to Matrix Computations: New York, Academic Press, Inc.
Thorson, J., Gaussian Elimination on a Banded Matrix: SEP-20, pp. 143-54.

Appendix A - Complex Householder Elimination for Banded Matrices

The following routine is a complex, banded version of the Householder elimination routine discussed in Stewart (1973). It employs a scaling which prevents overflows and underflows. Wherever a real number is expected as the result of some complex arithmetic, a real floating number is employed.

```
c          Householder triangularization for complex and
c          banded matrices.  The input matrix should be
c          stored as bands according to the recipe in
c          Thorson (SEP-20).
c          Inputs:   a, matrix to be triangularized plus
c                              extra columns of scratch space
c                    m, the bandwidth of "a"
c                    n, the length of the diagonal of "a"
c          The algorithm finds a unitary factor Q and an
c          upper triangular factor R for "a", such that
c          "a=QR."  On input "a" contains m columns of infor-
c          mation.  On output, the vectors needed to con-
c          struct Q are stored in columns 1 through (m+1)/2
c          and the bands of R are stored in columns (m+1)/2+1
c          through m+m/2.  Columns m+1+m/2 through m+2+m/2
c          contain still more parameters needed to construct
c          Q.  Since all the parameters needed for forming
c          the unitary matrix Q are stored away, many right
c          hand sides can be solved for with a single
c          triangularization step.
           subroutine hous(a,m,n)
           integer i,j,k,lim,m,n,r
           real*4 eta,pi,reta,sumsq,vka
           complex*8 a(256,9),cdir,rho,sigma,tau,v(256)
           r = (m + 1)/2
           do 00 i = 1,n
           do 02 j = m+1,m+r-1
           a(i,j) = cmplx(0.0,0.0)
02         continue
00         continue
           do 04 k = 1,n-1
           eta = -1.0
           lim = min(n,k+r-1)
           do 06 i = k,lim
           if(.not.(eta .lt. abs(real(a(i,k-i+r))))) goto 08
           eta = abs(real(a(i,k-i+r)))
08         continue
           if(.not.(eta .lt. abs(aimag(a(i,k-i+r))))) goto 10
           eta = abs(aimag(a(i,k-i+r)))
10         continue
06         continue
           if(.not.(eta .eq. 0.0)) goto 12
           a(k,m+r) = 0.0
           a(k,m+r+1) = 0.0
           goto 13
12         continue
           reta = 1.0/eta
           do 14 i = k,lim
           v(i) = a(i,k-i+r)*reta
           a(i,k-i+r) = v(i)
14         continue
           sumsq = 0.0
           do 16 i = k,lim
           sumsq = sumsq + (real(v(i)))**2+(aimag(v(i)))**2
16         continue
           if(.not.(v(k) .eq. 0.0)) goto 18
           sigma = sqrt(sumsq)
           goto 19
18         continue
           vka = (real(v(k)))**2+(aimag(v(k)))**2
           sumsq = sqrt(sumsq/vka)
           sigma = v(k)*sumsq
19         continue
           rho = -eta*sigma
```

```
        v(k) = v(k) + sigma
        a(k,r) = v(k)
        pi = real(sigma)*real(v(k)) + aimag(sigma)*aimag(v(k))
        pi = 1.0/pi
        a(k,m+r) = pi
        a(k,m+r+1) = rho
        do 20 j = k+1,min(k+m-1,n)
        tau = 0.0
        do 22 i = k,lim
        tau = tau + conjg(v(i))*a(i,j-i+r)
22      continue
        tau = tau*pi
        do 24 i = k,lim
        a(i,j-i+r) = a(i,j-i+r) - tau*v(i)
24      continue
20      continue
13      continue
04      continue
        a(n,m+r+1) = a(n,r)
        return
        end


c       Back substitution.  The inputs are consistent
c       with those of subroutine hous.  The vector
c       "b" is the right hand side on input and the
c       solution vector on input.
        subroutine solve(a,b,m,n)
        integer i,j,k,m,n,r,lim
        real*4 pi
        complex*8 a(256,9),b(1),tau,v(256)
        r = (m+1)/2
        do 26 k = 1,n-1
        lim = min(n,k+r-1)
        pi = a(k,m+r)
        do 28 i = k,lim
        v(i) = a(i,k-i+r)
28      continue
        tau = 0.0
        do 30 i = k,lim
        tau = tau + conjg(v(i))*b(i)
30      continue
        tau = tau*pi
        do 32 i = k,lim
        b(i) = b(i) - tau*v(i)
32      continue
26      continue
        b(n) = b(n)/a(n,m+r+1)
        do 34 k = n-1,1,-1
        do 36 j = k+1,min(k+m-1,n)
        b(k) = b(k) - a(k,j-k+r)*b(j)
36      continue
        b(k) = b(k)/a(k,m+r+1)
34      continue
        return
        end
```