[6.3] Improving floating-point solutions to simultaneous equations

Round-off errors in matrix calculations can create errors larger than you might expect when finding solutions to simultaneous equations with *simult().* The closer the matrix is to being singular, the worse the error becomes. In many cases this error can be reduced, as follows.

Suppose we want to find the solution vector *x* in

$$A \cdot x = b \qquad [1]$$

We would use

```
simult(A,b)→x
```

However, because of round-off error, this function really returns *x* with an error of *dx*, which results in *b* being in error by *db*, or

$$A \cdot (x + dx) = b + db \qquad [2]$$

Subtracting [1] from [2], we get

$$A \cdot dx = db \qquad [3]$$

Solve [2] for db, and substitute into [3] to get

$$A \cdot dx = A \cdot (x + dx) - b \qquad [4]$$

Since we know everything on the right-hand side of [4], we can solve [4] for dx:

$$dx = A^{-1} \cdot A \cdot (x + dx) - b \qquad [5]$$

So, if we subtract *dx* from the original solution (x+dx), we get a better estimate of the real solution x.

Usually, equation [5] is evaluated in double precision, with the intent of getting results that are at least accurate to single precision. Even though the 89/92+ do not *have* double precision arithmetic, this process still results in some improvement. It is also common to apply the improvement process several times, to ensure convergence and to find an optimal solution. However, the limited precision of the 89/92+ usually prevent this type of repetitive improvment. Repeating the improvement process results in a solution with more error.

Here is a function that returns the improved solution:

```
simulti(A,b)
func
local x

simult(A,b)→x
x-(A^⁻1*(A*x-b))

Endfunc
```

As an example, consider using *simulti()* to find the coefficients for the Langrangian interpolating polynomial. This is just the polynomial that fits through the given points. A polynomial of degree n-1 can be fit through n points. This function returns the coefficients as a list, given the x-y point coordinates in a matrix:

```
polyfiti(xyd)
func
©Find coefficients of nth-order polynomial, given n+1 xy points
© 18marØØ/dburkett@infinet.com

local a,k,n,xd

rowdim(xyd)→n

seq(k,k,n-1,Ø,⁻1)→a
seq(mat▸list(submat(xyd,1,1,n,1))[k]^a,k,1,n)→a

mat▸list(simulti(a,submat(xyd,1,2,n,2)))

Endfunc
```

This table shows the results of the fit for fitting a function with and without improvement. The function is f(x) = tan(x), *x* in radians. *x* ranges from 0.4 to 1.4, with data points every 0.1 radians. This means that 11 points were fit to a 10th-order polynomial.

Without improvement:	RMS residual:	3.5E-8
	Maximum residual:	3.5E-6
	Minimum residual:	3.4E-10
With improvement:	RMS residual:	7.4E-11
	Maximum residual:	5.9E-9
	Minimum residual:	-7.8E-10

The residual is the difference between the actual b-values and the calculated b-values. The RMS residual is a deviation measurement of all the residuals, and the minimum and maximum residuals are the most extreme residuals for all the fit points.

For this function and data, the improvement results in several orders of magnitude in both the RMS residuals and the extreme residuals. Without the improvement, the calculated results are only accurate to about five significant digits. With the improvement, the results are accurate to at least 8 significant digits.