include a baseline function implemented as a separate user-defined fit function. With standard curve fitting, you have to write a user-defined function that implements the sum.

Instead, you can use an alternate syntax for the FuncFit operation to fit to a list of fit functions that are summed automatically. The results for each fit function are returned via separate coefficient waves, making it easy to keep the results of each term in the sum separate. The syntax is described in the **FuncFit** operation on page V-273.

The sum-of-fit-functions feature can mix any kind of fit function — built-in, or any of the variants described in **User-Defined Fitting Functions** on page III-250. However, even if you use only built-in fit functions you must provide initial guesses because our initial-guess generating code cannot discern how the various features of the data are partitioned amongst the list of fit functions.

Linear Dependency: A Major Issue

When you fit a list of fit functions you must be careful not to introduce linear dependencies between the coefficients of the list of functions. The most likely such dependency will arise from a constant Y offset used with all of the built-in fit functions, and commonly included in user functions. This Y offset accounts for any offset error that is common when making real-world measurements. For instance, the equation for the built-in exp function is:

$$y_0 + A\exp(-Bx)$$

If you sum two terms, you get two y_0's (primes used for the second copy of the function):

$$y_0 + A\exp(-Bx) + y_0' + A'\exp(-B'x)$$

As long as *B* and *B'* are distinct, the two exponential terms will not be a problem. But y_0 and y'_0 are linearly dependent — they cannot be distinguished mathematically. If you increase one and decrease the other the same amount, the result is exactly the same. This is sure to cause a singular matrix error when you try to do the fit.

The solution is to hold one of the y_0 coefficients. Because each fit function in the list has its own coefficient wave, and you can specify a hold string for each function, this is easy (see **Example: Summed Exponentials** on page III-246).

There are many ways to introduce linear dependence. They are not always so easy to spot!

Constraints Applied to Sums of Fit Functions

The sums of fit functions feature does not include a keyword for specifying constraints on a function-by-function basis. But you can use the normal constraint specifications - you just have to translate the constraint expressions to take account of the list of functions. (If you are interested in using constraints, but don't know about the syntax for constraints, see **Fitting with Constraints** on page III-227.)

Constraint expressions use K*n* to designate fit coefficients where *n* is simply the sequential number of the fit coefficient within the list of coefficients. n starts with zero. In order to reference fit coefficients for a fit function in a list of summed fit functions, you must account for all the fit coefficients in all the fit functions in the list before the fit function you are trying to constrain. For example, this is part of a command that will fit a sum of three exponential terms:

```
FuncFit {{exp,expTerm1},{exp,expTerm2,hold="1"},{exp,expTerm3,hold="1"}} ...
```

In order to constrain coefficients of the second exponential term, you must know first that the built-in exp fit function has three fit coefficients. Here is the equation of the exp fit function:

```
f(x) = y0 + A*exp(-x*invTau)
```

The vertical offset (y0) of the first summed exp term is represented in a constraint expression as "K0", the amplitude as "K1", and invTau as "K2". Constraint expressions for the second exp term continue the count starting with "K3" for y0 (but see the section **Linear Dependency: A Major Issue** on page III-245 and note