

Guidelines on Writing Scientific Computing Software

Dianne P. O’Leary

January 2005

Every piece of software for scientific computing should be carefully designed and documented.

Design:

- Parameters should be variables, as much as possible. *For example, a subroutine to solve a linear system should work for any size of the matrix, rather than having a size specified.*
- The code should be modular, so that a user can pull out one piece and substitute another when necessary. *For example, a minimization code should call upon a separate function to get a function value, so that it can be used to minimize any function.*
- On the other hand, there is considerable overhead involved in function calls, so each module should involve a substantial computation in order to mask this overhead. *For example, don’t write a separate function to add 5 numbers.*
- “Spaghetti code” should be avoided. In other words, the sequence of instructions should be top-to-bottom (including loops), without a lot of jumps in control.
- The names of variables should be chosen to remind the reader of their purpose. *For example, `lambda` is better than `l` as the name of a Lagrange multiplier.*
- The code should be reasonably efficient. In particular, it should not take an order of magnitude more time or storage than necessary. *For example, if a code to solve a linear system with n variables takes $O(n^4)$ operations or $O(n^3)$ storage, then it is not so useful.*
- And, of course, the program should be correct.

Documentation: Documentation provides you and other potential users of your code an easy source of information about the use and design of the software.

The documentation at the top of the module should provide basic information to help a potential user decide whether the software is of interest. It should include:

- purpose of code *Why: this is certainly the first thing a user wants to know!*
- name of author *Why: someone to whom bugs can be reported and questions asked. (Mathworks is the author of the Matlab codes.)*
- date of original code, and list of later modifications *Why: it gives information such as whether the code is likely to run under the current computer environment and whether it might include the latest advances.*
- description of each input parameter *Why: so that a user knows what information needs to be provided and in what format.*
- description of each output parameter *Why: so that a user knows what information will be yielded.*
- brief description of the method, and references. *Why: to help a user decide whether the method fits his/her needs.*

In-line documentation identifies the major sections of the code and provides some detail on the method used. It is important in specifying the algorithm, identifying bugs, and providing information to someone who might need to modify the software in order to solve a slightly different problem.

Note that the documentation should be an integral part of the code; in other words, it is not enough to include it in a separate document, because a potential user might not have access to that document.

Example: Here is an example of a function that is well-designed and well-documented (although it certainly could be improved).

```
function [x,r,A,condA] = expfit(alpha,t,y)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% function [x,r,A,condA] = expfit(alpha,t,y)
% Given the rate constants alpha, expfit determines the
% linear coefficients in the exponential fitting of the
% data y taken at times t.
% On return:
%   x is the vector of linear coefficients
%   r is the vector of data residuals:
%       observed y - model
%   A is the matrix in the least squares problem
%   condA is the condition number of A
%
% Algorithm:
%   We use the singular value decomposition (SVD) to
%   solve the linear least squares problem
%   min_{x} ||A x - y||_2
%   where
%   A(i,j) = exp(alpha(j) t(i)) for i=1:m, j=1:n,
%   where m is the number of datapoints and n is the
%   number of rate constants.
% Reference: See
%   Gene H. Golub and Charles van Loan,
%   Matrix Computations, Johns Hopkins Press, 1989
%   for information on the SVD and data fitting.
%
% expfit.m Dianne P. O'Leary 02/2004
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Compute the matrix A, its SVD, and its condition number.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for j=1:length(alpha)
    A(:,j) = exp(alpha(j)*t);
end

[m,n] = size(A);
[u,s,v] = svd(A);
if (n>1)
    s = diag(s);
else
    s = s(1,1);
end
condA = s(1)/s(n);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Compute the solution to the least squares problem
%  $x = V S^{-1} U' y$  and the residual  $r$ .
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

w = (u(:,1:n)'*y) ./ s;
x = v*w;
r = y - A*x;

```