list of errata (as of 19oct00)
Charles F. Van Loan’s Scientific Computing, 2nd ed

Each item’s location is identified by
page/paragraph or item/line

Those starting with e are not errors but emendation proposals.

2/3/-5: changing --> curving
5/first script/4: omit ;
5/-3: in --> into
e6/display 4/: omit first blank (also later at times)
e7//1: more interesting: x = 3:-1:4
e9/script: avoid use of extra variable a, i.e., use y(1:m+1) = sin(2*pi*x(1:m+1));
11/display 4/: this is not what was used in the earlier script
14/2/2: genuine --> smooth
14/-6: incorrect claim; the script that produced Fig.1.4 is further below.
16/script 2/: in the line preceding, we were promised a single vector-level command to replace the loop in the previous script, namely y = ( (1+x/24)./(1-x/12+(x/384).*x) ).^8;
18/-4/-1: axis by itself --> axis auto
19//2: xnew --> x
20//1,2: should be using \tt for those matlab plotting symbol specs.
e21/second display/: should read plot( x([end 1:end]), y([end 1:end]) ) (i.e., why destroy the original information?)
e22/display -2/: are pixels always square?
e25/script 2/: at this point, the variable a is no longer needed, i.e., use y = [sin(x)
sin(2*x) sin(3*x) sin(4*x)]*[2;3;7;5];, or, better yet, y = sin(x*(1:4))*[2;3;7;5];
26/2/1: row --> rows
26/2/2: we also --> we can also
30/footnote/: I thought there is now a possible label to identify a logical variable, to help with the discontinuity in a(b) which occurs when b == ones(size(a))
e31/script/: avoid inconsistencies by making 500 a variable that is set once
32/last script/-3: use the linestyle ’-*’ to specify simultaneously linefill and marker style, i.e., use here plot(1:n,zeros(1,n),I+1,x(I+1),’-*’)
work with the points rather than their coordinates. In that way, one can be sure that all ops to be done the same on both coordinates are done the same. It also produces cleaner code. Note that ginput is willing to output a 2-vector instead of two scalars if so asked.

places --> place

stick to the model: \( exact = approx + error \)

relerr) --> relerr,'o')

the function being plotted is only defined for integers, hence drawing this linear fill-in is a sham.

< \( x \leq x \) 

\( \epsilon \) Note \( \epsilon /2 \). Note

\( y = 1; \) --> \( y = term; \)

\( y = \text{ones}(x); \), and follow the \( \text{term} = \) ... line by \( y = \text{term}; \), as that reflects the logic of the calculation better and avoids one call to \( \text{ones} \).

now doesn’t work for matrices \( x \). For that, would replace the two lines before the loop by \( n = \text{prod}((\text{size})(x)); y = \text{zeros}((\text{size})(x)); \)

This quantity --> Their sum

\( \sin(x) \) --> \( \sin(10x) \)

out side --> outside

pretty --> Pretty

use \( \text{num2cell} \) instead

Please, here and elsewhere, stick to MATLAB’s way of writing a polynomial in powerform. That would make it easy to use MATLAB’s built-in tools (polyval, polyfit, etc) for working with polynomials.

the quantity \( m \) is computed but never used. (Also, in line following that, could avoid mults by using \( \text{repmat} \) instead.)

why is \( x_0 \) made a column vector? As far as I can see, all works fine without that additional step.

if the output from \( \text{IntervN} \) were just 1 item, incorporating both
the centers x and the coefficients c, then, when using HornerV, we wouldn’t have
to input two items, nor have to remember in which order they must appear
here. There is still way too much FORTRAN thinking in this book.

e88//2: Here (and elsewhere) is an opportunity to use repmat, i.e., to write pVal
= repmat(c(end),size(z)); and so avoid all those multiplications (and learn
about another tool for proper vectorization).

e94/function InterpN2/1,end: skip last line; change first line to function y = In-
terpN2(x,y)

106/function pwL/8: the variable z is nowhere defined.

e106/function pwL/: why not put out the interpolant in the ppform understood
by MATLAB’s ppval. This obviates having to write and teach pwLEval. At a min-
imum, output the entire description of this pl interpolant, making it possible
to supply to pwLEval just one item, rather than the present 3 (which also now
requires me to remember the order in which to supply them).

e108/Locate/: this way of locating a point within a mesh does not vectorize
easily. Better to use the simple scheme employed in MATLAB’s ppval for simul-
taneously locating all the entries of a vector with respect to a given mesh.

e111/-1: comment on the fact that, for a quadratic f, the left-hand side would
give the exact maximum absolute error.

e112/function pwLAdapt/: since x and y are paired, use just one array, xy say, with
two rows, rather than the individual x and y. This cuts down on the number
of input arguments as well as on the number of assignment statements.

112/function pwLAdapt/8: delta are hmin --> delta and hmin are

e112/function pwLAdapt/-4,-3: use end instead of length(xRight)

113/Script File: ShowPWL2/-1: 1, --> 3,

115/P3.1.10/: the website provides no answer here. I recommend the two-liner
used in MATLAB’s ppval. Come to think of it, I recommend simply using ppval.

e116/first display/: the particular form of the cubic is entirely unmotivated. I
recommend, in any case, treating this as a special case of the Newton form of
the polynomial interpolant.

117/Theorem3/4: \([L, R] \rightarrow [x_L, x_R]\)

e128/CubicSpline/: any of these end conditions can be handled by MATLAB’s spline
command: not-a-knot is the default, but clamped/complete is directly avail-
able from spline as well. Any others can be obtained by using the fact that
spline can handle vector-valued data. Thus, if interpolation to y at the n points
in x is wanted, construct first \(s = spline(x,[0 y 0; zeros(1,n+1); zeros(1,n+1)
1]);\) then add the appropriate weighted sum of its second and third component
to its first component to achieve the desired end condition. E.g., to get a peri-
odic spline, compute \(s1 = ppder(s); s1vals = diff(ppval(s1,x([1 end])),1,2);\)
\(s2vals = diff(ppval(ppder(s1),x(1,end))1,,2);\), then get the weights alpha =
and use them to put together the interpolating periodic spline as follows:

\[
\text{s = mkpp(breaks, [1, alpha] * reshape(c, 3, l*k))};
\]

131/3/1,2: \texttt{pp or pp}?

133/P3.3.10/-1: \texttt{Spline} --\texttt{spline}

139/function NVWeights/: here and elsewhere, a good opportunity to use \texttt{switch} and \texttt{case}.

146/function CompQNC/2: the last input argument is not mentioned.

158/Spline Quadrature/: this should be vectorized

159/function SplineQ/: this should be vectorized!

169/first script/: vectorize: \texttt{repmat(1:n,n,1); 1./(ans'+ans-1)};

170/fist script/: cut out the inner loop by \texttt{P(i,2:i) = P(i-1,1:i-1)+P(i-1,2:i)};

171/function Circulant/-2: \texttt{[C(i-1,n) C(i-1,1:n-1)] --> C(i-1, [n,1:n-1])} (A vectorized version of this is quite cute.)

171/-1/1: \texttt{toeplitz --> Toeplitz}

172/1/-1: a nonzero --\texttt{a possibly nonzero}

180/f/: the book, in effect, does the column version of matrix multiplication twice, as \texttt{MatMatSax} and as \texttt{MatMatVec}, and fails to do the row-version, i.e.,

\[
(AB)(i,:) = \sum_k A(i,k)B(k,:).
\]