The following MATLAB M-file `adapt` uses the basic trapezium rule for R_1 and for R_2, the twice-repeated trapezium rule. The function is recursive, that is, it calls itself.

```
function [q, fcount] = adapt(f,a,b,tol,fs)
%ADAPT  Adaptive quadrature using the trapezium rule.
%       [Q,FCOUNT] = ADAPT(F,A,B,TOL,FS) uses adaptive quadrature based
%       on the trapezium rule to approximate the integral of the
%       function specified by F from A to B.
%       F must accept a vector input argument and return a vector
%       of function values.
%       FS is the 1-by-3 vector [f(a) f(m) f(b)], where m is the
%       midpoint of [a,b].
%       FS is used on recursive calls but not in the original call.
%       TOL is an absolute accuracy tolerance, which defaults to 1e-3.
%
%       FCOUNT is the total number of function evaluations required.
%       Q is the estimated integral

global xpts                % To record where F is evaluated.
fcount = 0;
if nargin < 4, tol = 1e-3; end
if nargin < 5              % On first call need to set up FS.
   xpts = [a (a+b)/2 b];
   fs = feval(f,xpts);
   fcount = 3;             % Count function evaluations.
end
T1 =  (b-a)*(fs(1)+fs(3))/2;         % Trapezium rule.
T2 =  (b-a)*(fs(1)+2*fs(2)+fs(3))/4; % Trapezium rule twice.
if abs(T1-T2) <= tol
   q = T2;
else
   xnew = [a+(b-a)/4 a+3*(b-a)/4];
   xpts = [xpts xnew];
   fnew = feval(f,xnew);
   fcount = fcount + 2;
   fsleft  = [fs(1) fnew(1) fs(2)]; fsright = [fs(2) fnew(2) fs(3)];
   [q1,fcount1] = adapt(f,a,(a+b)/2,tol/2,fsleft);
   [q2,fcount2] = adapt(f,(a+b)/2,b,tol/2,fsright);
   q = q1 + q2; fcount = fcount + fcount1 + fcount2;
end
```

A few comments:

1. The role of `fs` is to pass on function values when we subdivide an interval, to avoid repeating function evaluations.

2. When a subdivision is made half the permitted error is allowed in each new interval.

3. `xpts` is declared as a global variable, so that a single copy of it is kept that is accessible within each invocation of `adapt` and accessible also to the main workspace. (The use of global variables is generally bad programming practice).

4. `adapt` illustrates well the principles of adaptive quadrature, but is far from efficient and robust. For greater efficiency better basic quadrature rules need to be used, and for practical use tests are needed to stop the computation if the intervals become too small.

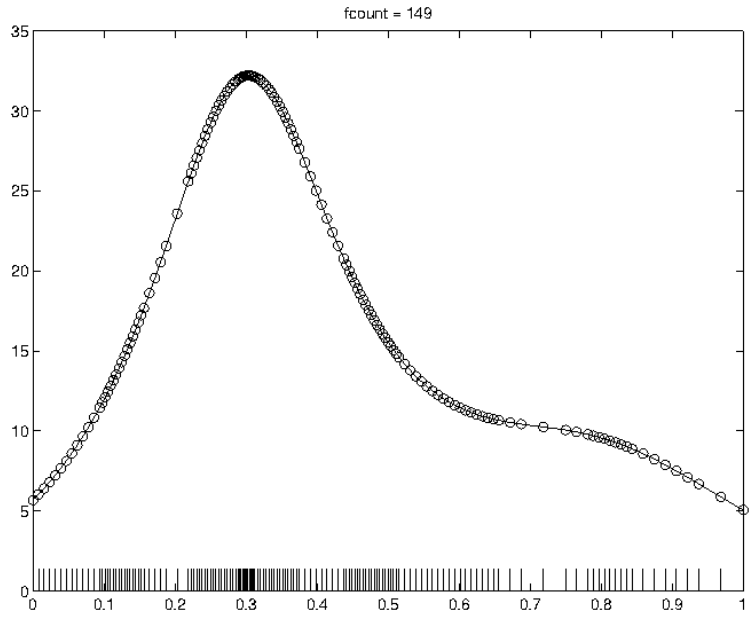

Figure 1: Example of `adapt`, showing where the integral is evaluated.

The following script illustrates `adapt` on a function with a hump:

```
%ADAPT_EX        Script M-file to illustrate ADAPT.

fun = inline('1./((x-0.3).^2+0.03) + 1./((x-0.8).^2+0.1)-4');
a = 0; b = 1; tol = 8e-3;
global xpts      % Give access to grid, for later plotting.
[q,fcount] = adapt(fun,a,b,tol)
x = linspace(a,b,100); fx = fun(x); fxpts = fun(xpts);
plot(x,fx,xpts,fxpts,'o')
title(['fcount = ' num2str(fcount)]); hold on
% Plot x-points chosen by ADAPT as ticks on x-axis.
z = [0 max(fxpts)*4.25/100];
for i=1:max(size(xpts))
    plot( [ xpts(i); xpts(i) ], z )
end
hold off
```

The script produces the plot in Figure 1. Note how the computational effort is correctly concentrated near the hump at $x = 0.3$.