```
%% Copyright (C) 2005 Julius O. Smith III
%%
%% This program is free software; you can redistribute it and/or modify it
%% under the terms of the GNU General Public License as published by
%% the Free Software Foundation; either version 2, or (at your option)
%% any later version.
%%
%% This program is distributed in the hope that it will be useful, but
%% WITHOUT ANY WARRANTY; without even the implied warranty of
%% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  The GNU
%% General Public License has more details.
%%
%% You should have received a copy of the GNU General Public License
%% along with this program; see the file COPYING.  If not, see
%% <http://www.gnu.org/licenses/>.

%% -*- texinfo -*-
%% @deftypefn {Function File} {[@var{r}, @var{p}, @var{f}, @var{m}] =} residuez (@var{B}, @var{A})
%% Compute the partial fraction expansion of filter @math{H(z) = B(z)/A(z)}.
%%
%% INPUTS:
%% @var{B} and @var{A} are vectors specifying the digital filter @math{H(z) = B(z)/A(z)}.
%% Say @code{help filter} for documentation of the @var{B} and @var{A}
%% filter coefficients.
%%
%% RETURNED:
%%   @itemize
%%   @item @var{r} = column vector containing the filter-pole residues@*
%%   @item @var{p} = column vector containing the filter poles@*
%%   @item @var{f} = row vector containing the FIR part, if any@*
%%   @item @var{m} = column vector of pole multiplicities
%%   @end itemize
%%
%% EXAMPLES:
%% @example
%%   Say @code{test residuez verbose} to see a number of examples.
%% @end example
%%
%% For the theory of operation, see
%% @indicateurl{http://ccrma.stanford.edu/~jos/filters/residuez.html}
%%
%% @seealso{residue residued}
%% @end deftypefn

function [r, p, f, m] = residuez(B, A, tol)
% RESIDUEZ - return residues, poles, and FIR part of B(z)/A(z)
%
% Let nb = length(b), na = length(a), and N=na-1 = no. of poles.
% If nb<na, then f will be empty, and the returned filter is
%
%            r(1)                     r(N)
% H(z) = ----------------  + ... + ----------------- = R(z)
%        [ 1-p(1)/z ]^m(1)         [ 1-p(N)/z ]^m(N)
%
% If, on the other hand, nb >= na, the FIR part f will not be empty.
% Let M = nb-na+1 = order of f = length(f)-1). Then the returned filter is
%
% H(z) = f(1) + f(2)/z + f(3)/z^2 + ... + f(M+1)/z^M + R(z)
%
% where R(z) is the parallel one-pole filter bank defined above.
% Note, in particular, that the impulse-response of the one-pole
% filter bank is in parallel with that of the the FIR part.  This can
% be wasteful when matching the initial impulse response is important,
% since F(z) can already match the first N terms of the impulse
% response. To obtain a decomposition in which the impulse response of
% the IIR part R(z) starts after that of the FIR part F(z), use RESIDUED.
%
% J.O. Smith, 9/19/05

if nargin==3
  warning("tolerance ignored");
end
NUM = B(:)'; DEN = A(:)';
% Matlab's residue does not return m (since it is implied by p):
[r,p,f,m]=residue(conj(fliplr(NUM)),conj(fliplr(DEN)));
p = 1 ./ p;
r = r .* ((-p) .^m);
if f, f = conj(fliplr(f)); end

%!test
%! B=[1 -2 1]; A=[1 -1];
%! [r,p,f,m] = residuez(B,A);
%! assert(r,0,100*eps);
%! assert(p,1,100*eps);
%! assert(f,[1 -1],100*eps);
%! assert(m,1,100*eps);
```

```
 87
 88  %!test
 89  %! B=1; A=[1 -1j];
 90  %! [r,p,f,m] = residuez(B,A);
 91  %! assert(r,1,100*eps);
 92  %! assert(p,1j,100*eps);
 93  %! assert(f,[],100*eps);
 94  %! assert(m,1,100*eps);
 95
 96  %!test
 97  %! B=1; A=[1 -1 .25];
 98  %! [r,p,f,m] = residuez(B,A);
 99  %! [rs,is] = sort(r);
100  %! assert(rs,[0;1],1e-7);
101  %! assert(p(is),[0.5;0.5],1e-8);
102  %! assert(f,[],100*eps);
103  %! assert(m(is),[1;2],100*eps);
104
105  %!test
106  %! B=1; A=[1 -0.75 .125];
107  %! [r,p,f,m] = residuez(B,A);
108  %! [rs,is] = sort(r);
109  %! assert(rs,[-1;2],100*eps);
110  %! assert(p(is),[0.25;0.5],100*eps);
111  %! assert(f,[],100*eps);
112  %! assert(m(is),[1;1],100*eps);
113
114  %!test
115  %! B=[1,6,2]; A=[1,-2,1];
116  %! [r,p,f,m] = residuez(B,A);
117  %! [rs,is] = sort(r);
118  %! assert(rs,[-10;9],1e-7);
119  %! assert(p(is),[1;1],1e-8);
120  %! assert(f,[2],100*eps);
121  %! assert(m(is),[1;2],100*eps);
122
123  %!test
124  %! B=[6,2]; A=[1,-2,1];
125  %! [r,p,f,m] = residuez(B,A);
126  %! [rs,is] = sort(r);
127  %! assert(rs,[-2;8],1e-7);
128  %! assert(p(is),[1;1],1e-8);
129  %! assert(f,[],100*eps);
130  %! assert(m(is),[1;2],100*eps);
131
132  %!test
133  %! B=[1,6,6,2]; A=[1,-2,1];
134  %! [r,p,f,m] = residuez(B,A);
135  %! [rs,is] = sort(r);
136  %! assert(rs,[-24;15],2e-7);
137  %! assert(p(is),[1;1],1e-8);
138  %! assert(f,[10,2],100*eps);
139  %! assert(m(is),[1;2],100*eps);
140
141  %!test
142  %! B=[1,6,6,2]; A=[1,-(2+j),(1+2j),-j];
143  %! [r,p,f,m] = residuez(B,A);
144  %! [rs,is] = sort(r);
145  %! assert(rs,[-2+2.5j;7.5+7.5j;-4.5-12j],1E-6);
146  %! assert(p(is),[1j;1;1],1E-6);
147  %! assert(f,-2j,1E-6);
148  %! assert(m(is),[1;2;1],1E-6);
149
150  %!test
151  %! B=[1,0,1]; A=[1,0,0,0,0,-1];
152  %! [r,p,f,m] = residuez(B,A);
153  %! [as,is] = sort(angle(p));
154  %! rise = [ ...
155  %!  0.26180339887499 - 0.19021130325903i; ...
156  %!  0.03819660112501 + 0.11755705045849i; ...
157  %!  0.4; ...
158  %!  0.03819660112501 - 0.11755705045849i; ...
159  %!  0.26180339887499 + 0.19021130325903i;];
160  %! pise = [ ...
161  %! -0.80901699437495 - 0.58778525229247i; ...
162  %!  0.30901699437495 - 0.95105651629515i; ...
163  %!  1; ...
164  %!  0.30901699437495 + 0.95105651629515i; ...
165  %! -0.80901699437495 + 0.58778525229247i];
166  %! assert(r(is),rise,100*eps);
167  %! assert(p(is),pise,100*eps);
168  %! assert(f,[],100*eps);
169  %! assert(m,[1;1;1;1;1],100*eps);
```