```
## Copyright (C) 1994-2011 John W. Eaton
## Copyright (C) 2007 Ben Abbott
##
## This file is part of Octave.
##
## Octave 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 3 of the License, or (at
## your option) any later version.
##
## Octave 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.  See the GNU
## General Public License for more details.
##
## You should have received a copy of the GNU General Public License
## along with Octave; see the file COPYING.  If not, see
## <http://www.gnu.org/licenses/>.

## -*- texinfo -*-
## @deftypefn {Function File} {[@var{r}, @var{p}, @var{k}, @var{e}] =} residue (@var{b}, @var{a})
## Compute the partial fraction expansion for the quotient of the
## polynomials, @var{b} and @var{a}.
## @tex
## $$
## {B(s)\over A(s)} = \sum_{m=1}^M {r_m\over (s-p_m)^e_m}
##   + \sum_{i=1}^N k_i s^{N-i}.
## $$
## @end tex
## @ifnottex
##
## @example
## @group
##  B(s)    M       r(m)         N
##  ---- = SUM -------------  + SUM k(i)*s^(N-i)
##  A(s)   m=1 (s-p(m))^e(m)    i=1
## @end group
## @end example
##
## @end ifnottex
## @noindent
## where @math{M} is the number of poles (the length of the @var{r},
## @var{p}, and @var{e}), the @var{k} vector is a polynomial of order @math{N-1}
## representing the direct contribution, and the @var{e} vector specifies
## the multiplicity of the m-th residue's pole.
##
## For example,
##
## @example
## @group
## b = [1, 1, 1];
## a = [1, -5, 8, -4];
## [r, p, k, e] = residue (b, a);
##      @result{} r = [-2; 7; 3]
##      @result{} p = [2; 2; 1]
##      @result{} k = [](0x0)
##      @result{} e = [1; 2; 1]
## @end group
## @end example
##
## @noindent
## which represents the following partial fraction expansion
## @tex
## $$
## {s^2+s+1\over s^3-5s^2+8s-4} = {-2\over s-2} + {7\over (s-2)^2} + {3\over s-1}
## $$
## @end tex
## @ifnottex
##
## @example
## @group
##         s^2 + s + 1       -2        7        3
##    ------------------- = ----- + ------- + -----
##    s^3 - 5s^2 + 8s - 4   (s-2)   (s-2)^2   (s-1)
## @end group
## @end example
##
## @end ifnottex
##
## @deftypefnx {Function File} {[@var{b}, @var{a}] =} residue (@var{r}, @var{p}, @var{k})
## @deftypefnx {Function File} {[@var{b}, @var{a}] =} residue (@var{r}, @var{p}, @var{k}, @var{e})
## Compute the reconstituted quotient of polynomials,
## @var{b}(s)/@var{a}(s), from the partial fraction expansion;
## represented by the residues, poles, and a direct polynomial specified
## by @var{r}, @var{p} and @var{k}, and the pole multiplicity @var{e}.
##
```

```
## If the multiplicity, @var{e}, is not explicitly specified the multiplicity is
## determined by the function @code{mpoles}.
##
## For example:
##
## @example
## @group
## r = [-2; 7; 3];
## p = [2; 2; 1];
## k = [1, 0];
## [b, a] = residue (r, p, k);
##      @result{} b = [1, -5, 9, -3, 1]
##      @result{} a = [1, -5, 8, -4]
##
## where mpoles is used to determine e = [1; 2; 1]
##
## @end group
## @end example
##
## Alternatively the multiplicity may be defined explicitly, for example,
##
## @example
## @group
## r = [7; 3; -2];
## p = [2; 1; 2];
## k = [1, 0];
## e = [2; 1; 1];
## [b, a] = residue (r, p, k, e);
##      @result{} b = [1, -5, 9, -3, 1]
##      @result{} a = [1, -5, 8, -4]
## @end group
## @end example
##
## @noindent
## which represents the following partial fraction expansion
## @tex
## $$
## {-2\over s-2} + {7\over (s-2)^2} + {3\over s-1} + s = {s^4-5s^3+9s^2-3s+1\over s^3-5s^2+8s-4}
## $$
## @end tex
## @ifnottex
##
## @example
## @group
##     -2        7        3         s^4 - 5s^3 + 9s^2 - 3s + 1
##    ----- + ------- + ----- + s = --------------------------
##    (s-2)   (s-2)^2   (s-1)          s^3 - 5s^2 + 8s - 4
## @end group
## @end example
##
## @end ifnottex
## @seealso{poly, roots, conv, deconv, mpoles, polyval, polyderiv, polyint}
## @end deftypefn

## Author: Tony Richardson <arichard@stark.cc.oh.us>
## Author: Ben Abbott <bpabbott@mac.com>
## Created: June 1994
## Adapted-By: jwe

%%%%%%%%%%% old_code %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%function [r, p, k, e] = residue (b, a, varargin)
%%%%
%%%%  if (nargin < 2 || nargin > 4)
%%%%%%%%%%% new_code %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [r, p, k, e] = bg_residue (b, a, varargin)
  global       G_FID;
  global       G_DEBUG;
  bg_enable_notes=1;   %%% set to 1 to print notes in files
  if (nargin < 2 || nargin > 5)
%%%%%%%%%%% end of new_code %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    print_usage ();
  endif

%%%%%%%%%%% new_code %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    if (nargin>2 && ischar(varargin{nargin-2}))
        MODE=varargin{nargin-2};
        nargin=nargin-1;
    else
        MODE="MODE_S";
    end
%%%%%%%%%%% end of new_code %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  toler = .001;

  if (nargin >= 3)
    if (nargin >= 4)
      e = varargin{2};
```

```
    else
      e = [];
    endif
    ## The inputs are the residue, pole, and direct part. Solve for the
    ## corresponding numerator and denominator polynomials
%%%%%%%%%%% old_code %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%    [r, p] = rresidue (b, a, varargin{1}, toler, e);
%%%%%%%%%%% new_code %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%  Here the role of r and p are interchanged with a and b.
                                        if G_DEBUG
                                            fprintf(G_FID,'Inputs to bg_rresidue are:\n');
                                            bg_print('r',b);
                                            bg_print('p',a);
                                            fprintf(G_FID,'MODE =%s\n',MODE);
                                            fprintf(G_FID,'toler =%+17.10e\n',toler);
                                            fprintf(G_FID,'---------------------------------------- ');
                                            fprintf(G_FID,'Into [a,b]=bg_rresidue');
                                            fprintf(G_FID,'(r,p,varargin{1},toler,e,MODE);\n');
                                        end
%%  Here the role of r and p are interchanged with b and a.
    [r, p] = bg_rresidue (b, a, varargin{1}, toler, e, MODE);
%%  Here the role of r and p are interchanged with b and a.
                                        if G_DEBUG
                                            fprintf(G_FID,'-------------------------------------- ');
                                            fprintf(G_FID,'Out of [a,b]=bg_rresidue');
                                            fprintf(G_FID,'(r,p,varargin{1},toler,e,MODE);\n');
                                            fprintf(G_FID,'Outputs from bg_rresidue are:\n');
                                            bg_print('b',r);
                                            bg_print('a',p);
                                        end
%%  Here the role of r and p are interchanged with b and a.
%%%%%%%%%%% end of new_code %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    return
  endif

  ## Make sure both polynomials are in reduced form.

  a = polyreduce (a);
  b = polyreduce (b);

  b = b / a(1);
  a = a / a(1);

  la = length (a);
  lb = length (b);

  ## Handle special cases here.

  if (la == 0 || lb == 0)
    k = r = p = e = [];
    return;
  elseif (la == 1)
    k = b / a;
    r = p = e = [];
    return;
  endif

  ## Find the poles.

  p = roots (a);
  lp = length (p);

  ## Sort poles so that multiplicity loop will work.

  [e, indx] = mpoles (p, toler, 1);
  p = p (indx);

  ## For each group of pole multiplicity, set the value of each
  ## pole to the average of the group. This reduces the error in
  ## the resulting poles.

  p_group = cumsum (e == 1);
  for ng = 1:p_group(end)
    m = find (p_group == ng);
    p(m) = mean (p(m));
  endfor

  ## Find the direct term if there is one.

  if (lb >= la)
    ## Also return the reduced numerator.
    [k, b] = deconv (b, a);
    lb = length (b);
  else
    k = [];
  endif
```

```

  ## Determine if the poles are (effectively) zero.

%%%%%%%%%%% old_code %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%  small = max (abs (p));
%%%%  if (isa (a, "single") || isa (b, "single"))
%%%%    small = max ([small, 1]) * eps ("single") * 1e4 * (1 + numel (p))^2;
%%%%  else
%%%%    small = max ([small, 1]) * eps * 1e4 * (1 + numel (p))^2;
%%%%  endif
%%%%  p(abs (p) < small) = 0;
%%%%%%%%%%% new_code %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                    if G_DEBUG
                                        fprintf(G_FID,’MODE   =%s\n’,MODE);
                                        if (isa (a, "single") || isa (b, "single"))
                                            fprintf(G_FID,’eps("single")=%+17.10e\n’,eps("single"));
                                        else
                                            fprintf(G_FID,’eps    =%+17.10e\n’,eps);
                                        end
                                    endif
    if (isa (a, "single") || isa (b, "single"))
        eps_mod=eps("single");
    else
        eps_mod=eps;
    endif
    if strcmp(MODE,"MODE_S")
                                    if G_DEBUG
                                        fprintf(G_FID,"Eliminate small poles here using\n");
                                        fprintf(G_FID,’ eps_mod = %+17.10e\n’,eps_mod);
                                        fprintf(G_FID,"small = max (abs (p));\n");
                                        fprintf(G_FID,"small = max ([small,1])*eps*1e4*(1 + numel (p))^2
;\n");
                                    end
        small = max (abs (p));
        factor=(1 + numel (p))^2;
                                    if G_DEBUG
                                        fprintf(G_FID,’ small_1   =%+17.10e\n’,small);
                                    end
        small = max ([small, 1]) * eps_mod * 1e4 * factor;
                                    if G_DEBUG
                                        fprintf(G_FID,’ small_2   =%+17.10e\n’,small);
                                        bg_print(’p_1’,   p);
                                        fprintf(G_FID,’small =%+17.10e\n’,small);
                                    end
        p(abs (p) < small) = 0;
                                    if G_DEBUG
                                        bg_print(’p_2’,   p);
                                    end
    elseif strcmp(MODE,"MODE_Z")||strcmp(MODE,"MODE_Z2")
                                    if G_DEBUG
                                        fprintf(G_FID,"Eliminate small poles here using\n");
                                        fprintf(G_FID,’ eps_mod = %+17.10e\n’,eps_mod);
                                    end
        small = max (abs (1./p));
        factor=(1 + numel (p))^2;
                                    if G_DEBUG
                                        fprintf(G_FID," small_1 = %+17.10e\n",small );
                                    end
        small = max ([small, 1]) * eps_mod * 1e4 * factor;
                                    if G_DEBUG
                                        fprintf(G_FID," small_2 = %+17.10e\n",small );
                                    end
        small=small/1000;
                                    if G_DEBUG
                                        fprintf(G_FID,’ small   = %+17.10e\n’,small );
                                        bg_print(’p_1’,1./p);
                                    end
        p(abs (1./p) < small) = 0;
                                    if G_DEBUG
                                        bg_print(’p_2’,1./p);
                                    end
        p = polyreduce (p);
    else
       fprintf("Error: MODE not equal to \"MODE_S\",\"MODE_Z\", or \"MODE_Z2\"\n");
       exit;
    end
                                    if G_DEBUG
                                        if strcmp(MODE,"MODE_S")
                                            bg_print(’p’,p);
                                        elseif strcmp(MODE,"MODE_Z")
                                            bg_print(’p’,1./p);
                                            bg_print(’P’,p);
                                        end
                                    end
%%%%%%%%%%% end of new_code %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```
  ## Determine if the poles are (effectively) real, or imaginary.

  index = (abs (imag (p)) < small);
  p(index) = real (p(index));
  index = (abs (real (p)) < small);
  p(index) = 1i * imag (p(index));

  ## The remainder determines the residues.  The case of one pole
  ## is trivial.

  if (lp == 1)
    r = polyval (b, p);
    return;
  endif

  ## Determine the order of the denominator and remaining numerator.
  ## With the direct term removed the potential order of the numerator
  ## is one less than the order of the denominator.

  aorder = numel (a) - 1;
  border = aorder - 1;

  ## Construct a system of equations relating the individual
  ## contributions from each residue to the complete numerator.
%%%%%%%%%%% new_code %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                        Count=0;
                                        if G_DEBUG
                                           fprintf(G_FID,'-------------------------------------------------- ');
                                           fprintf(G_FID,'Count=%i;\n',Count);
                                        end
%%%%%%%%%%% end of new_code %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  A = zeros (border+1, border+1);
  B = prepad (reshape (b, [numel(b), 1]), border+1, 0);
  for ip = 1:numel(p)
    ri = zeros (size (p));
    ri(ip) = 1;
%%%%%%%%%%% old_code %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%    A(:,ip) = prepad (rresidue (ri, p, [], toler), border+1, 0).';
%%%%%%%%%%% new_code %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                        if G_DEBUG
                                            fprintf(G_FID,'Inputs to bg_rresidue are:\n');
                                            fprintf(G_FID,'MODE =%s\n',MODE);
                                            fprintf(G_FID,'toler =%+17.10e\n',toler);
                                            if strcmp(MODE,"MODE_S")
                                               bg_print('ri',ri);
                                               bg_print('p',p);
                                               fprintf(G_FID,'-------------------------------------------------- ');
                                               fprintf(G_FID,'Into A(:,ip)=prepad(bg_rresidue');
                                               fprintf(G_FID,'(ri,p,[],toler,[],MODE),border+1,0)\n');
                                            elseif strcmp(MODE,"MODE_Z")||strcmp(MODE,"MODE_Z2")
                                               bg_print('RI',ri);
                                               bg_print('P',p);
                                               fprintf(G_FID,'-------------------------------------------------- ');
                                               fprintf(G_FID,'Into A(:,IP)=prepad(bg_rresidue');
                                               fprintf(G_FID,'(RI,P,[],toler,[],MODE),border+1,0)\n');
                                            end
                                        end
    A(:,ip) = prepad (bg_rresidue (ri, p, [], toler,[], MODE), border+1, 0).';
                                        Count++;
                                        if G_DEBUG
                                            if strcmp(MODE,"MODE_S")
                                                fprintf(G_FID,'------------------------------------------------- ');
                                                fprintf(G_FID,'Out of A(:,ip)=prepad(bg_rresidue');
                                                fprintf(G_FID,'(ri,p,[],toler,[],MODE),border+1,0)\n');
                                            elseif strcmp(MODE,"MODE_Z")||strcmp(MODE,"MODE_Z2")
                                                fprintf(G_FID,'------------------------------------------------- ');
                                                fprintf(G_FID,'Out of A(:,IP)=prepad(bg_rresidue');
                                                fprintf(G_FID,'(RI,P,[],toler,[],MODE),border+1,0)\n');
                                            end
                                            fprintf(G_FID,'-------------------------------------------------- ');
                                            fprintf(G_FID,'Count=%i;\n',Count);
                                            fprintf(G_FID,'Type 1 Calculations for A gives:\n');
                                            bg_print('A',A);
                                            fprintf(G_FID,'This value of A is not used in the following.\n');
                                        end
%%%%%%%%%%% end of new_code %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  endfor

```

```
  ## Solve for the residues.
%%%%%%%%%%% new_code %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    A = cal_M(p,e);
                                    if G_DEBUG
                                        fprintf(G_FID,'Type 2 Calculation for A gives:\n');
                                        bg_print('A',A);
                                    end
%%%%%%%%%%% end of new_code %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  r = A \ B;

endfunction

%%%%%%%%%%% old_code %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%function [pnum, pden, e] = rresidue (r, p, k, toler, e)
%%%%%%%%%%% new_code %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [pnum, pden, e] = bg_rresidue (r, p, k, toler, e, MODE)
%%%%%%%%%%% end of new_code %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  ## Reconstitute the numerator and denominator polynomials from the
  ## residues, poles, and direct term.

%%%%%%%%%%% old_code %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%  if (nargin < 2 || nargin > 5)
%%%%%%%%%%% new_code %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                        global      G_FID;
                                        global      G_DEBUG;
                                        bg_enable_notes=1;
    if (nargin < 2 || nargin > 6)
%%%%%%%%%%% end of new_code %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    print_usage ();
  endif

  if (nargin < 5)
    e = [];
  endif

  if (nargin < 4)
    toler = [];
  endif

  if (nargin < 3)
    k = [];
  endif

  if numel (e)
    indx = 1:numel(p);
  else
    [e, indx] = mpoles (p, toler, 0);
    p = p (indx);
    r = r (indx);
  endif

  indx = 1:numel(p);

  for n = indx
    pn = [1, -p(n)];
    if n == 1
      pden = pn;
    else
      pden = conv (pden, pn);
    endif
  endfor

  ## D is the order of the denominator
  ## K is the order of the direct polynomial
  ## N is the order of the resulting numerator
  ## pnum(1:(N+1)) is the numerator's polynomial
  ## pden(1:(D+1)) is the denominator's polynomial
  ## pm is the multible pole for the nth residue
  ## pn is the numerator contribution for the nth residue

  D = numel (pden) - 1;
  K = numel (k) - 1;
  N = K + D;
  pnum = zeros (1, N+1);
%%%%%%%%%%% new_code %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                    if G_DEBUG
                                        M = zeros(N,N);  %% initialize matrix M for storing results
                                        %%M = zeros(D,D);  %% initialize matrix M for storing results
                                        bg_print('pden',pden);
                                    end
%%%%%%%%%%% end of new_code %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  for n = indx(abs (r) > 0)
    p1 = [1, -p(n)];
    for m = 1:e(n)
```

```
      if (m == 1)
        pm = p1;
      else
        pm = conv (pm, p1);
      endif
    endfor
    pn = deconv (pden, pm);
%%%%%%%%%%%% new_code %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                        if G_DEBUG
                                            for l = 1:length(pn) M(l,n)=pn(l);endfor %% save values for reference
                                        end
%%%%%%%%%%%% end of new_code %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    pn = r(n) * pn;
    pnum = pnum + prepad (pn, N+1, 0, 2);
  endfor
%%%%%%%%%%%% new_code %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                        if G_DEBUG
                                            fprintf(G_FID,'Type 1 Calculations for M give:\n');
                                            bg_print('M',M);
                                            fprintf(G_FID,'Type 1 Calculations for pnum give:\n');
                                            bg_print('pnum',pnum);
                                        end
     M = cal_M(p,e);
                                        if G_DEBUG
                                            fprintf(G_FID,'Type 2 Calculations for M give:\n');
                                            bg_print('M',M);
                                        end
     B=M*r;            %% This is a better way
     pnum=transpose(B); %% to calculate pnum
     add=zeros(1,N+1-numel(pnum));
     pnum=cat(2,add,pnum);
                                        if G_DEBUG
                                            fprintf(G_FID,'Type 2 Calculations for pnum give:\n');
                                            bg_print('pnum_1',pnum);
                                            bg_print('pden_1',pden);
                                        end
%%%%%%%%%%%% end of new_code %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  ## Add the direct term.

  if (numel (k))
    pnum = pnum + conv (pden, k);
  endif

  ## Check for leading zeros and trim the polynomial coefficients.
%%%%%%%%%%%% old_code %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%  if (isa (r, "single") || isa (p, "single") || isa (k, "single"))
%%%%    small = max ([max(abs(pden)), max(abs(pnum)), 1]) * eps ("single");
%%%%  else
%%%%    small = max ([max(abs(pden)), max(abs(pnum)), 1]) * eps;
%%%%  endif
%%%%
%%%%  pnum(abs (pnum) < small) = 0;
%%%%  pden(abs (pden) < small) = 0;
%%%%
%%%%  pnum = polyreduce (pnum);
%%%%  pden = polyreduce (pden);
%%%%%%%%%%%% new_code %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    if (isa (r, "single") || isa (p, "single") || isa (k, "single"))
        eps_mod = eps ("single");
    else
        eps_mod = eps;
    endif
    if strcmp(MODE,"MODE_S")
                                        if G_DEBUG
                                            fprintf(G_FID,'MODE   =%s\n',MODE);
                                            fprintf(G_FID,'eps_mod    =%+17.10e\n',eps_mod);
                                            fprintf(G_FID,"Reduce a and b here\n");
                                        endif
        small = max ([max(abs(pden)), max(abs(pnum)), 1]) * eps_mod/1000;
                                        if G_DEBUG
                                            fprintf(G_FID,'small =%+17.10e\n',small);
                                        end
        pnum(abs (pnum) < small) = 0;
        pden(abs (pden) < small) = 0;
        pnum = polyreduce (pnum);
        pden = polyreduce (pden);
    elseif strcmp(MODE,"MODE_Z")||strcmp(MODE,"MODE_Z2")
                                        if G_DEBUG
                                            fprintf(G_FID,'MODE   =%s\n',MODE);
                                            fprintf(G_FID,'eps_mod=%+17.10e\n',eps_mod);
                                        end
        small = max(abs(pnum)) * eps_mod;
        pnum(abs (pnum) < small) = 0;
                                        if G_DEBUG
```

```
                                            fprintf(G_FID,'Modify pnum using: small =%+17.10e\n',small);
                                            bg_print('pnum_2',pnum);
                                        end
        if (isa (r, "single") || isa (p, "single") || isa (k, "single"))
            large =  1/(min(abs(pden))) / eps ("single");
        else
            large =  1/(min(abs(pden))) / eps;
        endif
        if strcmp(MODE,"MODE_Z")
            pden(abs (pden) > large) = 0;
        elseif strcmp(MODE,"MODE_Z2")
            pden(abs (pden) < small) = 0;
        end
                                        if G_DEBUG
                                            if strcmp(MODE,"MODE_Z")
                                                fprintf(G_FID,'Modify pden using: large =%+17.10e\n',large);
                                                pden(abs (pden) > large) = 0;
                                            elseif strcmp(MODE,"MODE_Z2")
                                                fprintf(G_FID,'Modify pden using: small =%+17.10e\n',small);
                                                pden(abs (pden) < small) = 0;
                                            end
                                            bg_print('pden_2',pden);
                                            fprintf(G_FID,"Reduce polynominals giving:\n");
                                        end
        pnum = polyreduce (pnum);
        pden = polyreduce (pden);
    else
       fprintf("Error: MODE not equal to \"MODE_S\",\"MODE_Z\", or \"MODE_Z2\"\n");
       exit;
    end
                                        if G_DEBUG
                                            bg_print('pnum',pnum);
                                            bg_print('pden',pden);
                                        end
%%%%%%%%%%% end of new_code %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

endfunction

%%%%%%%%%%% new_code %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function  M = cal_M(p,e)
    N = length(p); %% Define N as the number of poles
    C = cell(N,1); %% Define cell C to contain N vectors
    %%%%%%%%%%%%%%%%%%% PART 1: Calculation of the initial values for vectors C(2) through C(N)
    x=[1 -p(1)];
    if (e(2)==1)
        C{2}=x;
    else
        C{2}=[1];
    endif
    for i=3:N
        x=conv( x,[1 -p(i-1)]);
        if (e(i)==1)
             C{i}=x;
        else
             C{i}=C{i-(e(i)-1)};
        endif
    endfor
    %%%%%%%%%%%%%%%%%%%%% PART 2: Calculation of the final values for vectors C(N) through C(1)
    x=[1 -p(N)];
    if (N>2)
        C{N-1}=conv(C{N-1},x);
    else
        C{N-1}=x;
    end
    for i=N-2:-1:2;
        x=conv(x,[1 -p(i+1)]);
        C{i}=conv(C{i},x);
    endfor
    if (N>2)
        x=conv(x,[1 -p(2)]);
        if (e(1)>1)
            x=deconv(x,[1 -p(N)]);
        end
        C{1}=x;
    end
    %%%%%%%%%%%%%%%%%%%%%%%%%%% PART 3: Calculation of matrix M using vectors C(1) through C(N)
    M = zeros(N,N);
    for j=1:N
        x=C{j};
        len_x=length(x);
        offset=N-len_x;
        for i=1:offset
            M(i,j)=0;
        endfor
        for i=1:N-offset
            M(i+offset,j)=x(i);
```

```
        endfor
    endfor
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%% End of Calculation of matrix M
endfunction
%%%%%%%%%%% end of new_code %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%!test
%! b = [1, 1, 1];
%! a = [1, -5, 8, -4];
%%--------- old_code ------------------------------------------------------------------
%%                                                   %! [r, p, k, e] =    residue (b, a);
%%--------- new_code ------------------------------------------------------------------
%! [r, p, k, e] = bg_residue (b, a);
%%--------- end of new_code -----------------------------------------------------------
%! assert (abs (r - [-2; 7; 3]) < 1e-12
%!   && abs (p - [2; 2; 1]) < 1e-12
%!   && isempty (k)
%!   && e == [1; 2; 1]);
%! k = [1 0];
%! b = conv (k, a) + prepad (b, numel (k) + numel (a) - 1, 0);
%! a = a;
%%--------- old_code ------------------------------------------------------------------
%%                                                    %! [br, ar] = residue (r, p, k);
%%--------- new_code ------------------------------------------------------------------
%! [br, ar] = bg_residue (r, p, k);
%%--------- end of new_code -----------------------------------------------------------
%! assert ((abs (br - b) < 1e-12
%!   && abs (ar - a) < 1e-12));
%%--------- old_code ------------------------------------------------------------------
%%                                                    %! [br, ar] = residue (r, p, k, e);
%%--------- new_code ------------------------------------------------------------------
%! [br, ar] = bg_residue (r, p, k, e);
%%--------- end of new_code -----------------------------------------------------------
%! assert ((abs (br - b) < 1e-12
%!   && abs (ar - a) < 1e-12));

%!test
%! b = [1, 0, 1];
%! a = [1, 0, 18, 0, 81];
%%--------- old_code ------------------------------------------------------------------
%%                                                       %! [r, p, k, e] = residue (b, a);
%%--------- new_code ------------------------------------------------------------------
%! [r, p, k, e] = bg_residue (b, a);
%%--------- end of new_code -----------------------------------------------------------
%! r1 = [-5i; 12; +5i; 12]/54;
%! p1 = [+3i; +3i; -3i; -3i];
%! assert (abs (r - r1) < 1e-12 && abs (p - p1) < 1e-12
%!   && isempty (k)
%!   && e == [1; 2; 1; 2]);
%%--------- old_code ------------------------------------------------------------------
%%                                                       %! [br, ar] = residue (r, p, k);
%%--------- new_code ------------------------------------------------------------------
%! [br, ar] = bg_residue (r, p, k);
%%--------- end of new_code -----------------------------------------------------------
%! assert ((abs (br - b) < 1e-12
%!   && abs (ar - a) < 1e-12));

%!test
%! r = [7; 3; -2];
%! p = [2; 1; 2];
%! k = [1 0];
%! e = [2; 1; 1];
%%--------- old_code ------------------------------------------------------------------
%%                                                       %! [b, a] = residue (r, p, k, e);
%%--------- new_code ------------------------------------------------------------------
%!          [b, a] = bg_residue (r, p, k, e);
%%--------- end of new_code -----------------------------------------------------------
%! assert ((abs (b - [1, -5, 9, -3, 1]) < 1e-12
%!   && abs (a - [1, -5, 8, -4]) < 1e-12));
%%--------- old_code ------------------------------------------------------------------
%%                                                       %! [rr, pr, kr, er] = residue (b, a);
%%--------- new_code ------------------------------------------------------------------
%! [rr, pr, kr, er] = bg_residue (b, a);
%%--------- end of new_code -----------------------------------------------------------
%! [jnk, n] = mpoles(p);
%! assert ((abs (rr - r(n)) < 1e-12
%!   && abs (pr - p(n)) < 1e-12
%!   && abs (kr - k) < 1e-12
%!   && abs (er - e(n)) < 1e-12));

%!test
%! b = [1];
%! a = [1, 10, 25];
%%--------- old_code ------------------------------------------------------------------
%%                                                       %! [r, p, k, e] = residue (b, a);
%%--------- new_code ------------------------------------------------------------------
```

```
%! [r, p, k, e] = bg_residue (b, a);
%%-------- end of new_code --------------------------------------------------------------
%! r1 = [0; 1];
%! p1 = [-5; -5];
%! assert (abs (r - r1) < 1e-12 && abs (p - p1) < 1e-12
%!   && isempty (k)
%!   && e == [1; 2]);
%%-------- old_code ---------------------------------------------------------------------
%%                                                %! [br, ar] = residue (r, p, k);
%%-------- new_code ---------------------------------------------------------------------
%! [br, ar] = bg_residue (r, p, k);
%%-------- end of new_code --------------------------------------------------------------
%! assert ((abs (br - b) < 1e-12
%!   && abs (ar - a) < 1e-12));
%%-------- new_code ---------------------------------------------------------------------
%!test
%! z1=+7.0372976777e+06;
%! p1=-3.1415926536e+09;
%! p2=-4.9964813512e+08;
%! r1=-(1+z1/p1)/(1-p1/p2)/p2/p1  ;
%! r2=-(1+z1/p2)/(1-p2/p1)/p2/p1  ;
%! r3=(1+(p2+p1)/p2/p1*z1)/p2/p1  ;
%! r4=z1/p2/p1                    ;
%! r=[r1;r2;r3;r4];
%! p=[p1;p2;0;0];
%! k=[];
%! e=[1;1;1;2];
%! b=[1,z1];
%! a=[1,-(p1+p2),p1*p2,0,0];
%! [br, ar] = bg_residue (r, p, k, e);
%! assert ((abs (br - b) < 1e-8
%!       && abs (ar - a) < 1e-8));
%%-------- end of new_code --------------------------------------------------------------
```