

Sep 08, 11 5:31

bg_residue_nl.txt

Page 1/10

```

1  ## Copyright (C) 1994-2011 John W. Eaton
2  ## Copyright (C) 2007 Ben Abbott
3  ##
4  ## This file is part of Octave.
5  ##
6  ## Octave is free software; you can redistribute it and/or modify it
7  ## under the terms of the GNU General Public License as published by
8  ## the Free Software Foundation; either version 3 of the License, or (at
9  ## your option) any later version.
10 ##
11 ## Octave is distributed in the hope that it will be useful, but
12 ## WITHOUT ANY WARRANTY; without even the implied warranty of
13 ## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
14 ## General Public License for more details.
15 ##
16 ## You should have received a copy of the GNU General Public License
17 ## along with Octave; see the file COPYING. If not, see
18 ## <http://www.gnu.org/licenses/>.
19
20 ## -*- texinfo -*-
21 ## @deftypefn {Function File} {[@var{r}, @var{p}, @var{k}, @var{e}] =} residue (@var{b}, @var{a})
22 ## Compute the partial fraction expansion for the quotient of the
23 ## polynomials, @var{b} and @var{a}.
24 ## @tex
25 ## $$
26 ## \{B(s)\over A(s)\} = \sum_{m=1}^M \{r_m\over (s-p_m)^{e_m}\}
27 ## + \sum_{i=1}^N k_i s^{N-i}.
28 ## $$
29 ## @end tex
30 ## @ifnottex
31 ##
32 ## @example
33 ## @group
34 ## B(s)      M      r(m)      N
35 ## ---- = SUM ----- + SUM k(i)*s^(N-i)
36 ## A(s)      m=1 (s-p(m))^(e(m))      i=1
37 ## @end group
38 ## @end example
39 ##
40 ## @end ifnottex
41 ## @noindent
42 ## where @math{M} is the number of poles (the length of the @var{r},
43 ## @var{p}, and @var{e}), the @var{k} vector is a polynomial of order @math{N-1}
44 ## representing the direct contribution, and the @var{e} vector specifies
45 ## the multiplicity of the m-th residue's pole.
46 ##
47 ## For example,
48 ##
49 ## @example
50 ## @group
51 ## b = [1, 1, 1];
52 ## a = [1, -5, 8, -4];
53 ## [r, p, k, e] = residue (b, a);
54 ## @result{} r = [-2; 7; 3]
55 ## @result{} p = [2; 2; 1]
56 ## @result{} k = [](0x0)
57 ## @result{} e = [1; 2; 1]
58 ## @end group
59 ## @end example
60 ##
61 ## @noindent
62 ## which represents the following partial fraction expansion
63 ## @tex
64 ## $$
65 ## \{s^2+s+1\over s^3-5s^2+8s-4\} = \{-2\over s-2\} + \{7\over (s-2)^2\} + \{3\over s-1\}
66 ## $$
67 ## @end tex
68 ## @ifnottex
69 ##
70 ## @example
71 ## @group
72 ## s^2 + s + 1      -2      7      3
73 ## ----- = ----- + ----- + -----
74 ## s^3 - 5s^2 + 8s - 4 (s-2) (s-2)^2 (s-1)
75 ## @end group
76 ## @end example
77 ##
78 ## @end ifnottex
79 ##
80 ## @deftypefnx {Function File} {[@var{b}, @var{a}] =} residue (@var{r}, @var{p}, @var{k})
81 ## @deftypefnx {Function File} {[@var{b}, @var{a}] =} residue (@var{r}, @var{p}, @var{k}, @var{e})
82 ## Compute the reconstituted quotient of polynomials,
83 ## @var{b}(s)/@var{a}(s), from the partial fraction expansion;
84 ## represented by the residues, poles, and a direct polynomial specified
85 ## by @var{r}, @var{p} and @var{k}, and the pole multiplicity @var{e}.
86 ##

```

Sep 08, 11 5:31

bg_residue_n1.txt

Page 2/10

```

87  ## If the multiplicity, @var{e}, is not explicitly specified the multiplicity is
88  ## determined by the function @code{mpoles}.
89  ##
90  ## For example:
91  ##
92  ## @example
93  ## @group
94  ## r = [-2; 7; 3];
95  ## p = [2; 2; 1];
96  ## k = [1, 0];
97  ## [b, a] = residue (r, p, k);
98  ##     @result{} b = [1, -5, 9, -3, 1]
99  ##     @result{} a = [1, -5, 8, -4]
100 ##
101 ## where mpoles is used to determine e = [1; 2; 1]
102 ##
103 ## @end group
104 ## @end example
105 ##
106 ## Alternatively the multiplicity may be defined explicitly, for example,
107 ##
108 ## @example
109 ## @group
110 ## r = [7; 3; -2];
111 ## p = [2; 1; 2];
112 ## k = [1, 0];
113 ## e = [2; 1; 1];
114 ## [b, a] = residue (r, p, k, e);
115 ##     @result{} b = [1, -5, 9, -3, 1]
116 ##     @result{} a = [1, -5, 8, -4]
117 ## @end group
118 ## @end example
119 ##
120 ## @noindent
121 ## which represents the following partial fraction expansion
122 ## @tex
123 ## $$
124 ## \{-2\over s-2\} + \{7\over (s-2)^2\} + \{3\over s-1\} + s = \{s^4-5s^3+9s^2-3s+1\over s^3-5s^2+8s-4\}
125 ## $$
126 ## @end tex
127 ## @ifnottex
128 ##
129 ## @example
130 ## @group
131 ##      -2          7          3          s^4 - 5s^3 + 9s^2 - 3s + 1
132 ##      ---- + ---- + ---- + s = ----
133 ##      (s-2)   (s-2)^2   (s-1)      s^3 - 5s^2 + 8s - 4
134 ## @end group
135 ## @end example
136 ##
137 ## @end ifnottex
138 ## @seealso{poly, roots, conv, deconv, mpoles, polyval, polyderiv, polyint}
139 ## @end deftypefn
140
141 ## Author: Tony Richardson <arichard@stark.cc.oh.us>
142 ## Author: Ben Abbott <bpabbott@mac.com>
143 ## Created: June 1994
144 ## Adapted-By: jwe
145
146 %%%%%%%%%%% old_code %%%%%%%%%%%
147 %function [r, p, k, e] = residue (b, a, varargin)
148 %%%
149 %%% if (nargin < 2 || nargin > 4)
150 %%%%%%%%%%% new_code %%%%%%%%%%%
151 function [r, p, k, e] = bg_residue (b, a, varargin)
152     global      G_FID;
153     global      G_DEBUG;
154     bg_enable_notes=1;    %% set to 1 to print notes in files
155     if (nargin < 2 || nargin > 5)
156 %%%%%%%%%%% end of new_code %%%%%%%%%%%
157         print_usage ();
158     endif
159
160 %%%%%%%%%%% new_code %%%%%%%%%%%
161     if (nargin>2 && ischar(varargin{nargin-2}))
162         MODE=varargin{nargin-2};
163         nargin=nargin-1;
164     else
165         MODE="MODE_S";
166     end
167 %%%%%%%%%%% end of new_code %%%%%%%%%%%
168     toler = .001;
169
170     if (nargin >= 3)
171         if (nargin >= 4)
172             e = varargin{2};

```

Sep 08, 11 5:31

bg_residue_nl.txt

Page 3/10

```

173     else
174         e = [];
175     endif
176     ## The inputs are the residue, pole, and direct part. Solve for the
177     ## corresponding numerator and denominator polynomials
178     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
179     % [r, p] = rresidue (b, a, varargin{1}, toler, e);
180     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
181     % Here the role of r and p are interchanged with a and b.
182     if G_DEBUG
183         fprintf(G_FID,'Inputs to bg_rresidue are:\n');
184         bg_print('r',b);
185         bg_print('p',a);
186         fprintf(G_FID,'MODE =%s\n',MODE);
187         fprintf(G_FID,'toler =%+17.10e\n',toler);
188         fprintf(G_FID,'----- ');
189         fprintf(G_FID,'Into [a,b]=bg_rresidue');
190         fprintf(G_FID,'(r,p,varargin{1},toler,e,MODE);\n');
191     end
192     % Here the role of r and p are interchanged with b and a.
193     [r, p] = bg_rresidue (b, a, varargin{1}, toler, e, MODE);
194     % Here the role of r and p are interchanged with b and a.
195     if G_DEBUG
196         fprintf(G_FID,'----- ');
197         fprintf(G_FID,'Out of [a,b]=bg_rresidue');
198         fprintf(G_FID,'(r,p,varargin{1},toler,e,MODE);\n');
199         fprintf(G_FID,'Outputs from bg_rresidue are:\n');
200         bg_print('b',r);
201         bg_print('a',p);
202     end
203     % Here the role of r and p are interchanged with b and a.
204     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
205     return
206 endif
207
208     ## Make sure both polynomials are in reduced form.
209
210     a = polyreduce (a);
211     b = polyreduce (b);
212
213     b = b / a(1);
214     a = a / a(1);
215
216     la = length (a);
217     lb = length (b);
218
219     ## Handle special cases here.
220
221     if (la == 0 || lb == 0)
222         k = r = p = e = [];
223         return;
224     elseif (la == 1)
225         k = b / a;
226         r = p = e = [];
227         return;
228     endif
229
230     ## Find the poles.
231
232     p = roots (a);
233     lp = length (p);
234
235     ## Sort poles so that multiplicity loop will work.
236
237     [e, indx] = mpoles (p, toler, 1);
238     p = p (indx);
239
240     ## For each group of pole multiplicity, set the value of each
241     ## pole to the average of the group. This reduces the error in
242     ## the resulting poles.
243
244     p_group = cumsum (e == 1);
245     for ng = 1:p_group(end)
246         m = find (p_group == ng);
247         p(m) = mean (p(m));
248     endfor
249
250     ## Find the direct term if there is one.
251
252     if (lb >= la)
253         ## Also return the reduced numerator.
254         [k, b] = deconv (b, a);
255         lb = length (b);
256     else
257         k = [];
258     endif

```

Sep 08, 11 5:31

bg_residue_nl.txt

Page 4/10

```

259
260   ## Determine if the poles are (effectively) zero.
261
262   %%%%%%%%%%% old_code %%%%%%%%%%%
263   %%% small = max (abs (p));
264   %%% if (isa (a, "single") || isa (b, "single"))
265   %%%     small = max ([small, 1]) * eps ("single") * 1e4 * (1 + numel (p))^2;
266   %%% else
267   %%%     small = max ([small, 1]) * eps * 1e4 * (1 + numel (p))^2;
268   %%% endif
269   %%% p(abs (p) < small) = 0;
270   %%%%%%%%%%% new_code %%%%%%%%%%%
271   if G_DEBUG
272       fprintf(G_FID,'MODE   =%s\n',MODE);
273       if (isa (a, "single") || isa (b, "single"))
274           fprintf(G_FID,'eps("single")=%+17.10e\n',eps("single"));
275       else
276           fprintf(G_FID,'eps   =%+17.10e\n',eps);
277       end
278   endif
279   if (isa (a, "single") || isa (b, "single"))
280       eps_mod=eps("single");
281   else
282       eps_mod=eps;
283   endif
284   if strcmp(MODE,"MODE_S")
285       if G_DEBUG
286           fprintf(G_FID,"Eliminate small poles here using\n");
287           fprintf(G_FID,' eps_mod = %+17.10e\n',eps_mod);
288           fprintf(G_FID,"small = max (abs (p));\n");
289           fprintf(G_FID,"small = max ([small,1])*eps*1e4*(1 + numel (p))^2
; \n");
290       end
291       small = max (abs (p));
292       factor=(1 + numel (p))^2;
293       if G_DEBUG
294           fprintf(G_FID,' small_1   =%+17.10e\n',small);
295       end
296       small = max ([small, 1]) * eps_mod * 1e4 * factor;
297       if G_DEBUG
298           fprintf(G_FID,' small_2   =%+17.10e\n',small);
299           bg_print('p_1', p);
300           fprintf(G_FID,'small =%+17.10e\n',small);
301       end
302       p(abs (p) < small) = 0;
303       if G_DEBUG
304           bg_print('p_2', p);
305       end
306   elseif strcmp(MODE,"MODE_Z") || strcmp(MODE,"MODE_Z2")
307       if G_DEBUG
308           fprintf(G_FID,"Eliminate small poles here using\n");
309           fprintf(G_FID,' eps_mod = %+17.10e\n',eps_mod);
310       end
311       small = max (abs (1./p));
312       factor=(1 + numel (p))^2;
313       if G_DEBUG
314           fprintf(G_FID," small_1 = %+17.10e\n",small );
315       end
316       small = max ([small, 1]) * eps_mod * 1e4 * factor;
317       if G_DEBUG
318           fprintf(G_FID," small_2 = %+17.10e\n",small );
319       end
320       small=small/1000;
321       if G_DEBUG
322           fprintf(G_FID,' small   = %+17.10e\n',small );
323           bg_print('p_1',1./p);
324       end
325       p(abs (1./p) < small) = 0;
326       if G_DEBUG
327           bg_print('p_2',1./p);
328       end
329       p = polyreduce (p);
330   else
331       fprintf("Error: MODE not equal to \"MODE_S\", \"MODE_Z\", or \"MODE_Z2\"\n");
332       exit;
333   end
334       if G_DEBUG
335           if strcmp(MODE,"MODE_S")
336               bg_print('p',p);
337           elseif strcmp(MODE,"MODE_Z")
338               bg_print('p',1./p);
339               bg_print('P',p);
340           end
341       end
342   %%%%%%%%%%% end of new_code %%%%%%%%%%%
343

```

Sep 08, 11 5:31

bg_residue_nl.txt

Page 5/10

```

344     ## Determine if the poles are (effectively) real, or imaginary.
345
346     index = (abs (imag (p)) < small);
347     p(index) = real (p(index));
348     index = (abs (real (p)) < small);
349     p(index) = 1i * imag (p(index));
350
351     ## The remainder determines the residues. The case of one pole
352     ## is trivial.
353
354     if (lp == 1)
355         r = polyval (b, p);
356         return;
357     endif
358
359     ## Determine the order of the denominator and remaining numerator.
360     ## With the direct term removed the potential order of the numerator
361     ## is one less than the order of the denominator.
362
363     aorder = numel (a) - 1;
364     border = aorder - 1;
365
366     ## Construct a system of equations relating the individual
367     ## contributions from each residue to the complete numerator.
368     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
369     Count=0;
370     if G_DEBUG
371         fprintf(G_FID,'----- ' )
;
372         fprintf(G_FID,'Count=%i;\n',Count);
373         end
374     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
375     A = zeros (border+1, border+1);
376     B = prepad (reshape (b, [numel(b), 1]), border+1, 0);
377     for ip = 1:numel(p)
378         ri = zeros (size (p));
379         ri(ip) = 1;
380     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
381     %%% A(:,ip) = prepad (rresidue (ri, p, [], toler), border+1, 0).';
382     %%% new_code %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
383     if G_DEBUG
384         fprintf(G_FID,'Inputs to bg_rresidue are:\n');
385         fprintf(G_FID,'MODE =%s\n',MODE);
386         fprintf(G_FID,'toler =%+17.10e\n',toler);
387         if strcmp(MODE,"MODE_S")
388             bg_print('ri',ri);
389             bg_print('p',p);
390             fprintf(G_FID,'-----
-- ');
391             fprintf(G_FID,'Into A(:,ip)=prepad(bg_rresidue');
392             fprintf(G_FID,'(ri,p,[],toler,[],MODE),border+1,0)\n');
393             elseif strcmp(MODE,"MODE_Z")||strcmp(MODE,"MODE_Z2")
394                 bg_print('RI',ri);
395                 bg_print('P',p);
396                 fprintf(G_FID,'-----
-- ');
397                 fprintf(G_FID,'Into A(:,IP)=prepad(bg_rresidue');
398                 fprintf(G_FID,'(RI,P,[],toler,[],MODE),border+1,0)\n');
399                 end
400             end
401             A(:,ip) = prepad (bg_rresidue (ri, p, [], toler, [], MODE), border+1, 0).';
402             Count++;
403             if G_DEBUG
404                 if strcmp(MODE,"MODE_S")
405                     fprintf(G_FID,'-----
- ');
406                     fprintf(G_FID,'Out of A(:,ip)=prepad(bg_rresidue');
407                     fprintf(G_FID,'(ri,p,[],toler,[],MODE),border+1,0)\n');
408                     elseif strcmp(MODE,"MODE_Z")||strcmp(MODE,"MODE_Z2")
409                         fprintf(G_FID,'-----
- ');
410                         fprintf(G_FID,'Out of A(:,IP)=prepad(bg_rresidue');
411                         fprintf(G_FID,'(RI,P,[],toler,[],MODE),border+1,0)\n');
412                         end
413                     end
414                     fprintf(G_FID,'----- ' )
;
415                     fprintf(G_FID,'Count=%i;\n',Count);
416                     fprintf(G_FID,'Type 1 Calculations for A gives:\n');
417                     bg_print('A',A);
418                     fprintf(G_FID,'This value of A is not used in the following.\n')
;
419                     end
420     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% end of new_code %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
421     endfor
422

```

Sep 08, 11 5:31

bg_residue_nl.txt

Page 6/10

```

423  ## Solve for the residues.
424  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
425  A = cal_M(p,e);
426
427  if G_DEBUG
428      fprintf(G_FID,'Type 2 Calculation for A gives:\n');
429      bg_print('A',A);
430  end
431  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
432  r = A \ B;
433
434  endfunction
435
436  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
437  %function [pnum, pden, e] = rresidue(r, p, k, toler, e)
438  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
439  function [pnum, pden, e] = bg_rrresidue(r, p, k, toler, e, MODE)
440  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
441
442  ## Reconstitute the numerator and denominator polynomials from the
443  ## residues, poles, and direct term.
444
445  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
446  % if (nargin < 2 || nargin > 5)
447  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
448  % new_code %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
449  %                                     global      G_FID;
450  %                                     global      G_DEBUG;
451  %                                     bg_enable_notes=1;
452  % if (nargin < 2 || nargin > 6)
453  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
454  % end of new_code %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
455  % print_usage ();
456  % endif
457
458  if (nargin < 5)
459      e = [];
460  endif
461
462  if (nargin < 4)
463      toler = [];
464  endif
465
466  if (nargin < 3)
467      k = [];
468  endif
469
470  if numel(e)
471      indx = 1:numel(p);
472  else
473      [e, indx] = mpoles(p, toler, 0);
474      p = p(indx);
475      r = r(indx);
476  endif
477
478  indx = 1:numel(p);
479
480  for n = indx
481      pn = [1, -p(n)];
482      if n == 1
483          pden = pn;
484      else
485          pden = conv(pden, pn);
486      endif
487  endfor
488
489  ## D is the order of the denominator
490  ## K is the order of the direct polynomial
491  ## N is the order of the resulting numerator
492  ## pnum(1:(N+1)) is the numerator's polynomial
493  ## pden(1:(D+1)) is the denominator's polynomial
494  ## pm is the multible pole for the nth residue
495  ## pn is the numerator contribution for the nth residue
496
497  D = numel(pden) - 1;
498  K = numel(k) - 1;
499  N = K + D;
500  pnum = zeros(1, N+1);
501  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
502  % new_code %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
503  % if G_DEBUG
504  %     M = zeros(N,N); %% initialize matrix M for storing results
505  %     %M = zeros(D,D); %% initialize matrix M for storing results
506  %     bg_print('pden',pden);
507  % end
508  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
509  % end of new_code %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
510  for n = indx(abs(r) > 0)
511      pl = [1, -p(n)];
512      for m = 1:e(n)

```

Sep 08, 11 5:31

bg_residue_nl.txt

Page 7/10

```

509     if (m == 1)
510         pm = p1;
511     else
512         pm = conv (pm, p1);
513     endif
514 endfor
515 pn = deconv (pden, pm);
516 %%%%%%%%%%% new_code %%%%%%%%%%%
517     if G_DEBUG
518         for l = 1:length(pn) M(l,n)=pn(l);endfor %% save values for refe
519 rences
520     end
521 %%%%%%%%%%% end of new_code %%%%%%%%%%%
522     pn = r(n) * pn;
523     pnum = pnum + prepad (pn, N+1, 0, 2);
524 endfor
525 %%%%%%%%%%% new_code %%%%%%%%%%%
526     if G_DEBUG
527         fprintf(G_FID,'Type 1 Calculations for M give:\n');
528         bg_print('M',M);
529         fprintf(G_FID,'Type 1 Calculations for pnum give:\n');
530         bg_print('pnum',pnum);
531     end
532     M = cal_M(p,e);
533     if G_DEBUG
534         fprintf(G_FID,'Type 2 Calculations for M give:\n');
535         bg_print('M',M);
536     end
537     B=M*r;          %% This is a better way
538     pnum=transpose(B); %% to calculate pnum
539     add=zeros(1,N+1-numel(pnum));
540     pnum=cat(2,add,pnum);
541     if G_DEBUG
542         fprintf(G_FID,'Type 2 Calculations for pnum give:\n');
543         bg_print('pnum_1',pnum);
544         bg_print('pden_1',pden);
545     end
546 %%%%%%%%%%% end of new_code %%%%%%%%%%%
547     ## Add the direct term.
548
549     if (numel (k))
550         pnum = pnum + conv (pden, k);
551     endif
552
553     ## Check for leading zeros and trim the polynomial coefficients.
554 %%%%%%%%%%% old_code %%%%%%%%%%%
555     %%% if (isa (r, "single") || isa (p, "single") || isa (k, "single"))
556     %%%     small = max ([max(abs(pden)), max(abs(pnum)), 1]) * eps ("single");
557     %%% else
558     %%%     small = max ([max(abs(pden)), max(abs(pnum)), 1]) * eps;
559     %%% endif
560     %%%
561     %%% pnum(abs (pnum) < small) = 0;
562     %%% pden(abs (pden) < small) = 0;
563     %%%
564     %%% pnum = polyreduce (pnum);
565     %%% pden = polyreduce (pden);
566 %%%%%%%%%%% new_code %%%%%%%%%%%
567     if (isa (r, "single") || isa (p, "single") || isa (k, "single"))
568         eps_mod = eps ("single");
569     else
570         eps_mod = eps;
571     endif
572     if strcmp(MODE,"MODE_S")
573         if G_DEBUG
574             fprintf(G_FID,'MODE   =%s\n',MODE);
575             fprintf(G_FID,'eps_mod   =%+17.10e\n',eps_mod);
576             fprintf(G_FID,"Reduce a and b here\n");
577         endif
578         small = max ([max(abs(pden)), max(abs(pnum)), 1]) * eps_mod/1000;
579         if G_DEBUG
580             fprintf(G_FID,'small =%+17.10e\n',small);
581         end
582         pnum(abs (pnum) < small) = 0;
583         pden(abs (pden) < small) = 0;
584         pnum = polyreduce (pnum);
585         pden = polyreduce (pden);
586     elseif strcmp(MODE,"MODE_Z") || strcmp(MODE,"MODE_Z2")
587         if G_DEBUG
588             fprintf(G_FID,'MODE   =%s\n',MODE);
589             fprintf(G_FID,'eps_mod =%+17.10e\n',eps_mod);
590         end
591         small = max(abs(pnum)) * eps_mod;
592         pnum(abs (pnum) < small) = 0;
593         if G_DEBUG

```

Sep 08, 11 5:31

bg_residue_nl.txt

Page 8/10

```

594         fprintf(G_FID,'Modify pnum using: small =%+17.10e\n',small);
595         bg_print('pnum_2',pnum);
596         end
597         if (isa(r, "single") || isa(p, "single") || isa(k, "single"))
598             large = 1/(min(abs(pden))) / eps("single");
599         else
600             large = 1/(min(abs(pden))) / eps;
601         endif
602         if strcmp(MODE,"MODE_Z")
603             pden(abs(pden) > large) = 0;
604         elseif strcmp(MODE,"MODE_Z2")
605             pden(abs(pden) < small) = 0;
606         end
607         if G_DEBUG
608             if strcmp(MODE,"MODE_Z")
609                 fprintf(G_FID,'Modify pden using: large =%+17.10e\n',large);
610                 pden(abs(pden) > large) = 0;
611             elseif strcmp(MODE,"MODE_Z2")
612                 fprintf(G_FID,'Modify pden using: small =%+17.10e\n',small);
613                 pden(abs(pden) < small) = 0;
614             end
615             bg_print('pden_2',pden);
616             fprintf(G_FID,"Reduce polynomials giving:\n");
617         end
618         pnum = polyreduce(pnum);
619         pden = polyreduce(pden);
620     else
621         fprintf("Error: MODE not equal to \"MODE_S\", \"MODE_Z\", or \"MODE_Z2\"\n");
622         exit;
623     end
624     if G_DEBUG
625         bg_print('pnum',pnum);
626         bg_print('pden',pden);
627     end
628     %%%%%%%%%%% end of new_code %%%%%%%%%%%
629 endfunction
630
631 %%%%%%%%%%% new_code %%%%%%%%%%%
632 function M = cal_M(p,e)
633     N = length(p); % Define N as the number of poles
634     C = cell(N,1); % Define cell C to contain N vectors
635     %%%%%%%%%%% PART 1: Calculation of the initial values for vectors C(2) through C(N)
636     x=[1 -p(1)];
637     if (e(2)==1)
638         C{2}=x;
639     else
640         C{2}=[1];
641     endif
642     for i=3:N
643         x=conv(x,[1 -p(i-1)]);
644         if (e(i)==1)
645             C{i}=x;
646         else
647             C{i}=C{i-(e(i)-1)};
648         endif
649     endfor
650     %%%%%%%%%%% PART 2: Calculation of the final values for vectors C(N) through C(1)
651     x=[1 -p(N)];
652     if (N>2)
653         C{N-1}=conv(C{N-1},x);
654     else
655         C{N-1}=x;
656     end
657     for i=N-2:-1:2;
658         x=conv(x,[1 -p(i+1)]);
659         C{i}=conv(C{i},x);
660     endfor
661     if (N>2)
662         x=conv(x,[1 -p(2)]);
663         if (e(1)>1)
664             x=deconv(x,[1 -p(N)]);
665         end
666         C{1}=x;
667     end
668     %%%%%%%%%%% PART 3: Calculation of matrix M using vectors C(1) through C(N)
669     M = zeros(N,N);
670     for j=1:N
671         x=C{j};
672         len_x=length(x);
673         offset=N-len_x;
674         for i=1:offset
675             M(i,j)=0;
676         endfor
677         for i=1:N-offset
678             M(i+offset,j)=x(i);
679         endfor

```

Sep 08, 11 5:31

bg_residue_nl.txt

Page 9/10

```

680         endfor
681     endfor
682     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% End of Calculation of matrix M
683 endfunction
684 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% end of new_code %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
685
686 %!test
687 %! b = [1, 1, 1];
688 %! a = [1, -5, 8, -4];
689 %----- old_code -----
690 %
691 %----- new_code -----
692 %! [r, p, k, e] = bg_residue (b, a);
693 %----- end of new_code -----
694 %! assert (abs (r - [-2; 7; 3]) < 1e-12
695 %!      && abs (p - [2; 2; 1]) < 1e-12
696 %!      && isempty (k)
697 %!      && e == [1; 2; 1]);
698 %! k = [1 0];
699 %! b = conv (k, a) + prepad (b, numel (k) + numel (a) - 1, 0);
700 %! a = a;
701 %----- old_code -----
702 %
703 %----- new_code -----
704 %! [br, ar] = bg_residue (r, p, k);
705 %----- end of new_code -----
706 %! assert ((abs (br - b) < 1e-12
707 %!      && abs (ar - a) < 1e-12));
708 %----- old_code -----
709 %
710 %----- new_code -----
711 %! [br, ar] = bg_residue (r, p, k, e);
712 %----- end of new_code -----
713 %! assert ((abs (br - b) < 1e-12
714 %!      && abs (ar - a) < 1e-12));
715
716 %!test
717 %! b = [1, 0, 1];
718 %! a = [1, 0, 18, 0, 81];
719 %----- old_code -----
720 %
721 %----- new_code -----
722 %! [r, p, k, e] = bg_residue (b, a);
723 %----- end of new_code -----
724 %! r1 = [-5i; 12; +5i; 12]/54;
725 %! p1 = [+3i; +3i; -3i; -3i];
726 %! assert (abs (r - r1) < 1e-12 && abs (p - p1) < 1e-12
727 %!      && isempty (k)
728 %!      && e == [1; 2; 1; 2]);
729 %----- old_code -----
730 %
731 %----- new_code -----
732 %! [br, ar] = bg_residue (r, p, k);
733 %----- end of new_code -----
734 %! assert ((abs (br - b) < 1e-12
735 %!      && abs (ar - a) < 1e-12));
736
737 %!test
738 %! r = [7; 3; -2];
739 %! p = [2; 1; 2];
740 %! k = [1 0];
741 %! e = [2; 1; 1];
742 %----- old_code -----
743 %
744 %----- new_code -----
745 %! [b, a] = bg_residue (r, p, k, e);
746 %----- end of new_code -----
747 %! assert ((abs (b - [1, -5, 9, -3, 1]) < 1e-12
748 %!      && abs (a - [1, -5, 8, -4]) < 1e-12));
749 %----- old_code -----
750 %
751 %----- new_code -----
752 %! [rr, pr, kr, er] = bg_residue (b, a);
753 %----- end of new_code -----
754 %! [jnk, n] = mpoles(p);
755 %! assert ((abs (rr - r(n)) < 1e-12
756 %!      && abs (pr - p(n)) < 1e-12
757 %!      && abs (kr - k) < 1e-12
758 %!      && abs (er - e(n)) < 1e-12));
759
760 %!test
761 %! b = [1];
762 %! a = [1, 10, 25];
763 %----- old_code -----
764 %
765 %----- new_code -----

```

Sep 08, 11 5:31

bg_residue_n1.txt

Page 10/10

```

766 %! [r, p, k, e] = bg_residue (b, a);
767 %%----- end of new_code -----
768 %! r1 = [0; 1];
769 %! p1 = [-5; -5];
770 %! assert (abs (r - r1) < 1e-12 && abs (p - p1) < 1e-12
771 %!      && isempty (k)
772 %!      && e == [1; 2]);
773 %%----- old_code -----
774 %%                                     %! [br, ar] = residue (r, p, k);
775 %%----- new_code -----
776 %! [br, ar] = bg_residue (r, p, k);
777 %%----- end of new_code -----
778 %! assert ((abs (br - b) < 1e-12
779 %!      && abs (ar - a) < 1e-12));
780 %%----- new_code -----
781 %!test
782 %! z1=+7.0372976777e+06;
783 %! p1=-3.1415926536e+09;
784 %! p2=-4.9964813512e+08;
785 %! r1=-(1+z1/p1)/(1-p1/p2)/p2/p1 ;
786 %! r2=-(1+z1/p2)/(1-p2/p1)/p2/p1 ;
787 %! r3=(1+(p2+p1)/p2/p1*z1)/p2/p1 ;
788 %! r4=z1/p2/p1 ;
789 %! r=[r1;r2;r3;r4];
790 %! p=[p1;p2;0;0];
791 %! k=[];
792 %! e=[1;1;1;2];
793 %! b=[1,z1];
794 %! a=[1,-(p1+p2),p1*p2,0,0];
795 %! [br, ar] = bg_residue (r, p, k, e);
796 %! assert ((abs (br - b) < 1e-8
797 %!      && abs (ar - a) < 1e-8));
798 %%----- end of new_code -----

```