Four Issues with Octave

By Bernie Grung (9/8/2011)

Introduction

For many years, I have studied phase-locked loop (PLL) circuits using various simulation techniques including C++, Mathcad, Matlab and Octave. All simulations produce identical results except for Octave. In the following, I will explain the issues encountered with Octave and will present solutions to these issues. Primarily, I will focus attention on two functions in Octave – "residue" and "residuez" -- where "residue" is a subfunction of "residuez". Also, I will suggest that the best final solution will be to rewrite both functions so that they are completely independent of each other.

The Four Issues

There are four basic issues with the using functions "residue" and "residuez" for phase-locked loop (PLL) simulation in Octave: (1) the "trim" issue, (2) the "**z**-pole" issue, (3) the "**A**-matrix" issue, and (4) the "inverse" issue.

The trim issue is associated with lines 339 to 350 of "residue" as given in Figure 1. If we use "residue" for a typical PLL problem, the high-order polynomial coefficients in **pnum** and **pden** will be deleted since the value of **small** is too large. Dividing **small** by 1000 will fix this issue for continuous-time problems. Unfortunately, the same problem exists for "residuez" and the fix for this problem is much harder, as will be apparent after completing a discussion of the second issue.

```
339    ## Check for leading zeros and trim the polynomial coefficients.
340    if (isa (r, "single") || isa (p, "single") || isa (k, "single"))
341      small = max ([max(abs(pden)), max(abs(pnum)), 1]) * eps ("single");
342    else
343      small = max ([max(abs(pden)), max(abs(pnum)), 1]) * eps;
344    endif
345
346    pnum(abs (pnum) < small) = 0;
347    pden(abs (pden) < small) = 0;
348
349    pnum = polyreduce (pnum);
350    pden = polyreduce (pden);
```

Figure 1 The Code Concerning the Trim Issue

The **z**-pole issue is connected directly to lines 73 to 78 of "residuez", as shown in Figure 2. The input row vectors **A** and **B** are first converted to column vectors **NUM** and **DEN**. These vectors are then flipped in the left-right direction and the complex conjugate is taken of each element. This conversion has the effect of replacing each **z**-pole with its inverse. The "residue" function is then called using the modified values of **NUM** and **DEN** as inputs. After many lines of code in "residue", lines 218 to 226 are reached as listed in Figure 3. For continuous-time simulations, this code eliminates the very small poles from consideration. Because of **z**-pole inversion, this same code will eliminate the very large (dominant) poles in a discrete-time analysis, which is a disaster.

```
73  NUM = B(:)'; DEN = A(:)';
74  % Matlab's residue does not return m (since it is implied by p):
75  [r,p,f,m]=residue(conj(fliplr(NUM)),conj(fliplr(DEN)));
76  p = 1 ./ p;
77  r = r .* ((-p) .^m);
78  if f, f = conj(fliplr(f)); end
```

Figure 2 The Basic Source Code in "residuez"

```
218    ## Determine if the poles are (effectively) zero.
219
220    small = max (abs (p));
221    if (isa (a, "single") || isa (b, "single"))
222      small = max ([small, 1]) * eps ("single") * 1e4 * (1 + numel (p))^2;
223    else
224      small = max ([small, 1]) * eps * 1e4 * (1 + numel (p))^2;
225    endif
226    p(abs (p) < small) = 0;
```

Figure 3 The Source Code in "residue" Concerning the z-Pole Issue

Both, the trim issue and the **z**-pole issues indicate that "residue" must be modified for discrete-time problems to prevent the deletion of the dominant elements. To do this, we could employ a new global variable or we could introduce a new input parameter for "residue" function when it is called from the "residuez" function. Both solutions have some undesirable features but they would allow a switch statement to modify "residue" for discrete-time problems.

The **A**-matrix issue is associated with lines 250 to 259 of "residue" as shown in Figure 4. This matrix is defined so that $\mathbf{b}=(\mathbf{A}*\mathbf{r})^T$ where row vector **b** contains the numerator coefficients of the filter and the column vector **r** contains residues of the filter. The various columns of **A** are calculated one at a time using the subfunction "rresidue" of "residue". Inside "rresidue" as shown in Figure 5, the row vector **pden** is calculated from the poles ($\mathbf{p}_1$, $\mathbf{p}_2$,etc) by convolution of all terms of the form $\mathbf{(1-p_i)}$ assuming no repeated poles. Then, **pnum** is calculated using deconvolution which is a very inefficient and slow method. One solution is to eliminate the "deconv" function from the **A**-matrix calculations.

```
250    ## Construct a system of equations relating the individual
251    ## contributions from each residue to the complete numerator.
252
253    A = zeros (border+1, border+1);
254    B = prepad (reshape (b, [numel(b), 1]), border+1, 0);
255    for ip = 1:numel(p)
256      ri = zeros (size (p));
257      ri(ip) = 1;
258      A(:,ip) = prepad (rresidue (ri, p, [], toler), border+1, 0).';
259    endfor
```

Figure 4: The A-Matrix Issue as Contained in "residue".

```
315    D = numel (pden) - 1;
316    K = numel (k) - 1;
317    N = K + D;
318    pnum = zeros (1, N+1);
319    for n = indx(abs (r) > 0)
320      p1 = [1, -p(n)];
321      for m = 1:e(n)
322        if (m == 1)
323          pm = p1;
324        else
325          pm = conv (pm, p1);
326        endif
327      endfor
328      pn = deconv (pden, pm);
329      pn = r(n) * pn;
330      pnum = pnum + prepad (pn, N+1, 0, 2);
331    endfor
```

Figure 5: The A-Matrix Issue as Contained in "rresidue".

The final issue -- the inverse issue -- concerns the fact that there is no inverse function in "residuez". In Matlab, the main function is "[**r,p,k,e**]=residuez(**b,a**)" and its inverse is "[**b,a**]=residuez(**r,p,k,e**)". In Octave, the inverse function could be implemented by explicitly reversing the steps in "residuez" as given in Figure 2. Another approach could be to use "residue" directly but this works only if the "residue" function is modified correctly for discrete time analysis.

Recommendations

Octave has four issues which prevent its use for my production-level phase-locked loop circuits. Two of the issues -- the trim issue and the **z**-pole issue -- are catastrophic. The other two other issues -- the **A**-matrix issue and the inverse issue -- provide faster and more convenient simulations. More details and possible solutions are given in directory "coar39" (see below). In the future, the best solution is to modify "residue" and "residuez" such that these two functions are completely independent of each other and such that they contain the changes suggested above.

More Details and Possible Solutions are in Directory "coar39"

Directory	doc	Documentation in pdf form.
Directory	mat	Contains new Octave functions.
Directory	test	Outputs generated by "coar39.m".
Directory	test_old	Copy of the test directory for future reference.
Text File	coar39.sh	Shell script to run "coar39.m".
Text File	coar39.log	Output log of shell script.
Text File	coar39.m	Main function.
PDF File	coar39_8-functions	"bg_residue" and "bg_residuez" verification.
PDF File	coar39_4-issues	This Document which is entitled: "Four Issues with Octave" by Bernie Grung