

Bernie wrote:

> At the present time, I believe that trimming of the polynomial coefficients should be eliminated from the Octave code. The user should decide if he wants to reduce the polynomials in his code.

Jordi wrote:

> I don't think we can do this. It seems awkward and would likely break Matlab compatibility if no coefficients ever get trimmed unless explicitly requested by the user. You really can't suggest a better way to pick this tolerance?

Bernie response:

> At the present time, Matlab compatibility in “residue” is broken because Octave trims the coefficients and Matlab does not.

Jordi wrote:

> Further, even with that suggested change, your test fails because the tolerance is too low for br and b. Is this also a bug with residue or should your test allow for higher tolerance?

Bernie wrote:

> For a 64-bit system, the machine accuracy (eps) is about 2.22x10-16.

Jordi wrote:

> This accuracy is irrelevant of the machine architecture; it's simply the eps of an IEEE 754 double. It's very rare nowadays to compile Octave for machines that do not obey IEEE 754, so it's usually safe that this value is the eps of 1.

Bernie response:

> Octave trims coefficients by using an esp of 1.19x10-7 for 32 bit systems and by using an eps of 2.22x10-16 for 64 bits systems. I do not understand why these values are irrelevant. Moreover, I do not understand why “it's usually safe that this value is the eps of 1”.

Bernie wrote:

> For most poles, the tolerance should be better than 1e-12. But, there are outliers so that the assert test should be done pole-by-pole with variable tolerances. This is rather complicated so I used just 1e-8 as a compromise.

Jordi wrote:

> I think we need a more careful analysis of the error here to decide what we should choose.

Bernie response:

I have developed two new Octave test functions for investigating error tolerances using "bg_residue" and "residue". For "bg_residue", the output is given here:

```
Start of error calculations
         eps=+2.2204460492503131e-16
        b(1)=+1.0000000000000000e+00
       br(1)=+9.9999999999999989e-01
  b(1)-br(1)=+1.1102230246251565e-16  <  eps*10=  +2.2204460492503131e-15 for absolute error
1-b(1)/br(1)=+0.0000000000000000e+00  <  eps*10=  +2.2204460492503131e-15 for relative error
        b(2)=+7.0372976776999999e+06
       br(2)=+7.0372976776999990e+06
  b(2)-br(2)=+9.3132257461547852e-10  <  eps*10^7=+2.2204460492503131e-09 for absolute error
1-b(2)/br(2)=-2.2204460492503131e-16  <  eps*10  =+2.2204460492503131e-15 for relative error
End of error calculations
```

The print statements provide 16 places in the fractional part so that we can easily see the absolute error between **br(1)(**which is calculated using "residue") and **b(1)** (which is calculated without using "residue"). The absolute error [**b(1)-br(1)**] is about 1.1*10^-16 and this is less eps*2. The absolute error [**b(2)-br(2)**] is about 9.3*10^-10 and this is less than eps*10^7. A better measure of the tolerance is the relative error [**1-b(2)/br(2)**] and this is less than eps*10.

For "residue", the output is:

```
Start of error calculations
          eps=+2.2204460492503131e-16
        bx(1)=+7.0372976776999999e+06
        br(1)=+7.0372976776998583e+06
  bx(1)-br(1)=+1.4156103134155273e-07 < eps*10^9=+2.2204460492503131e-07 for absolute error
1-bx(1)/br(1)=-2.0206059048177849e-14 < eps*100 =+2.2204460492503131e-14 for relative error
End of error calculations
```

Here, we can evaluate only one coefficient in the vectors **bx** and **br** since Octave deletes the other coefficient. This deletion (or trimming of the polynomial coefficients) is not done in Matlab.