Complex#rationalize and to_r with approximate zeros

Currently, Complex#rationalize and Complex#to_r raise a RangeError if the imaginary part is nonzero or is a Float. Note that a BigDecimal(0) is accepted, though:

```
Complex(1, 0).to_r          # => Rational(1,1)
Complex(1, BigDecimal("0.0")).to_r # => Rational(1,1)
Complex(1, 0.0).to_r        # => RangeError
```

This is inconsistent. I recommend not raising an error for 0.0 (Float or BigDecimal). Any objection?

Related issues:
Blocked by Ruby master - Feature #5321: Introducing Numeric#exact? and Numeric#inexact?  Rejected

History

#1 - 08/10/2011 07:20 PM - mrkn (Kenta Murata)
- Assignee set to mrkn (Kenta Murata)

0.0 doesn't exactly represent zero. It may be 0.0+-10.0**(Float::MIN_10_EXP-17). BigDecimal(0) doesn't exactly represent zero, too.

I believe this issue should be resolved by introducing Numeric#exact? and/or Numeric#inexact? methods.

#2 - 03/18/2012 06:46 PM - shyouhei (Shyouhei Urabe)
- Status changed from Open to Assigned

#3 - 01/17/2020 01:18 AM - jeremyevans0 (Jeremy Evans)

As Numeric#exact? has been rejected, and BigDecimal is not a core class, I'm not sure what to do about this issue. mrkn (Kenta Murata) seems to recommend RangeError for Complex(1, BigDecimal("0.0")).to_r (the same as with Complex(1, 0.0).to_r). I think the bigdecimal library would have to override Kernel#Complex for that behavior. Would that be considered acceptable?

#4 - 01/17/2020 03:00 AM - mame (Yusuke Endoh)

We discussed #5321 at the dev-meeting. Whether a value is exact or inexact, is not decidable based on a class. For example, a literal 0.0 may be considered as precise and exact. But, if the same value is gained from an inaccurate calculation, it may be considered as inexact.

In regard to this particular issue, Naruse pointed out that we already have Float#to_r. So I think that it is reasonable that Complex(1, 0.0).to_r returns Rational(1, 1) instead of RangeError.