Ruby master - Feature #15571

Add methods: iroot, root, and roots

01/29/2019 05:35 PM - jzakiya (Jabari Zakiya)

Status:	Open
Priority:	Normal
Assignee:	
Target version:	

Description

Proposal
The rubygem roots provides a few methods to find the numerical roots
of real, complex, and integer numbers. This proposal requests including the
following three (3) methods into Ruby.

https://rubygems.org/gems/roots
https://github.com/jzakiya/roots

iroot: provide the accurate integer nth root value of any size integer

```
2.6.0 :002 > require 'roots'
 => true


2.6.0 :010 > n = 12345678901234567890
 => 12345678901234567890

2.6.0 :011 > Integer.sqrt n
 => 3513641828

2.6.0 :012 > n.iroot 2
 => 3513641828

2.6.0 :013 > n.iroot 3
 => 2311204

2.6.0 :014 > n.iroot 4
 => 59275
```

root: provide the accurate real value nth root of a real, complex, or integer numbers
roots: provide a collection of all real|complex nth values

```
2.6.0 :020 > n = 12345678901234567890
 => 12345678901234567890

2.6.0 :021 > Math.sqrt n
 => 3513641828.820144

2.6.0 :022 > n**(0.5)
 => 3513641828.820144

2.6.0 :023 > n.root 2
 => 3513641828.820144

2.6.0 :024 > n**(1.0/3)
 => 2311204.2409018343

2.6.0 :025 > n.root 3
 => 2311204.24090183

2.6.0 :026 > n.root 3,1
 => (2311204.24090183+0.0i)

2.6.0 :027 > n.root 3,2
```

```
 => (-1155602.12045092+2001561.58595532i)

2.6.0 :028 > n.root 3,3
 => (-1155602.12045092-2001561.58595532i)

2.6.0 :029 > n.roots 3
 => [(2311204.24090183+0.0i), (-1155602.12045092+2001561.58595532i), (-1155602.12045092-2001561.58
595532i)]

2.6.0 :031 > n.roots 3, :real
 => [(2311204.24090183+0.0i)]

2.6.0 :032 > n.roots 3, :complex
 => [(-1155602.12045092+2001561.58595532i), (-1155602.12045092-2001561.58595532i)]

2.6.0 :033 > n.roots 3, :odd
 => [(2311204.24090183+0.0i), (-1155602.12045092-2001561.58595532i)]

2.6.0 :034 > n.roots 3, :even
 => [(-1155602.12045092+2001561.58595532i)]

2.6.0 :035 > (247823 + 398439i).root 4
 => (25.33541017+6.56622124i)

2.6.0 :036 > (247823 + 398439i).roots 4
 => [(25.33541017+6.56622124i), (-6.56622124+25.33541017i), (-25.33541017-6.56622124i), (6.5662212
4-25.33541017i)]
```

Motivation
Ruby 2.5 included the method Integer.sqrt. It accurately returns the
integer squareroot of integers, whereas performing Math.sqrt(n).floor
produced rounding errows once n exceeded a certain threshold.

Whereas Integer.srt solved that problem for squareroots, the same problem
exists for the other nth roots when n reaches a certain (large) value too.
Adding iroot completes providing this functionality for all nth roots.

Adding root and roots adds funcitionality either not currently present
and/or provides it in an easier to use, standard, and more flexible manner.

I created the roots gem to help me do Project Euler (https://projecteuler.net/) problems.
To probably most people|programmers, Ruby is primarily associated with web
development through frameworks like Rails, Sinatra, Hanami, etc. However Ruby
has great utility in math and numerical analysis fields. These methods provide
basic arithmetic primitives upon which higher order functions can be created
without the need to search for third-party packages. They will increase Ruby's
footprint into numerical|analysis fields now dominated by Python and Julia.

Pros

- only 3 methods with no dependices
- fast and numerically accurate (can change shown digits for root(s)
- adds previously unavailable functionality
- provides existing funcionality in an easier to use, standard, and flexible manner
- provides more math primitives to create higher order algorithms
- makes Ruby, out-of-the-box, more useful for doing math, cryptography, etc
- enhances Ruby's reputation as a more math friendly language
- makes programmers doing math Happy! :-)

Cons

- it adds 3 methods to language core
- better names(?)

History

#1 - 01/30/2019 04:18 AM - duerst (Martin Dürst)

- Status changed from Open to Third Party's Issue

This is the (bug/feature) tracker for Ruby the language. Most gems are maintained separately. Issues for bundled gems may occasionally end up here, but the roots gem is independent. In addition, this gem seems to have been created by yourself (see https://github.com/jzakiya/roots), so you sure should know how to send yourself a bug report or feature request.

#2 - 01/30/2019 04:26 AM - duerst (Martin Dürst)

- Status changed from Third Party's Issue to Open

Sorry, I misread this. The proposal is about including some of the methods from the gem into Ruby itself. Looking at the number of downloads on rubygems.org (8,836 for the latest version), I'm not exactly convinced this needs to be in Ruby itself.

#3 - 01/30/2019 06:17 PM - jzakiya (Jabari Zakiya)

First, I think you should see the number of downloads for the roots gem in a different context. Instead of saying it only has 8837 downloads, you should say, Wow, over 8800 people found it useful enough to download to solve a problem for a niche use case for Ruby. And for the majority of the life of roots it only had the two methods root and roots, for real|complex numbers, until I added iroot and iroot2 in 2017.

And I hope the Facebooktizing (how many likes, or popularity) of a proposal hasn't become the defining criteria for its approval, over its merits, utility, and goodness.

I hope you take the time to actually access the underlying problem with Ruby these methods corrects. Ruby 2.5 added Integer.sqrt based on the underlying mathematical deficiencies (errors) it's implementation had that I raised. See the Medium post about it, and the discussion thread that lead to its ultimate inclusion.

https://medium.com/@atul9/using-ruby-2-5s-new-integer-sqrt-cd9cb5955e12

https://bugs.ruby-lang.org/issues/13219

To state simply, the implementation deficiency I raised to computing integer squareroots still exists for all other integer roots. It is an inherent structural characteristic when using floating point arithmetic to approximate these values.

The method iroot **fixes** that problem for all nth integer roots like Integer.sqrt does just for integer squareroots. In fact, its implementation in roots is just the generalization of the Newton method settled on to implement Integer.sqrt.

The methods roots and roots also provides missing functionality dealing with real and complex numbers, and fixes some errors, and unexpected results.

Examples
For any root n you have n possible distinct root values. All non-real roots occur as complex conjugate pairs. Currently, their is no easy and standard way Ruby provides you to see all the n roots, or any particular one, or know their order, or quadrant it exists in.

Also, when taking an odd nth root of a negative real|integer you expect to get the calculator answer, which is the negative value of the real integer root of the positive value.

Here's what you currently get with straight Ruby.

```
2.6.0  :>  27 ** (1.0/3) =>  3.0
2.6.0 : > -27 ** (1.0/3) => -3.0

2.6.0 : >  1_000 ** (1.0/3) =>  9.999999999999998
2.6.0 : > -1_000 ** (1.0/3) => -9.999999999999998

2.6.0 : >  1_000_000 ** (1.0/3) =>  99.99999999999997
2.6.0 : > -1_000_000 ** (1.0/3) => -99.99999999999997

Now observe what happens when assigning values to variables.

2.6.0 : > n =  1_000_000; n ** (1.0/3) => 99.99999999999997
2.6.0 : > n = -1_000_000; n ** (1.0/3) => (50.0+86.60254037844383i) #  first princple CCW root

Houston, we have a problem!

2.6.0 : > n =  1_000_000; n.root 3  =>  100.0
2.6.0 : > n = -1_000_000; n.root 3  => -100.0

2.6.0 : > n =  1_000_000; n.iroot 3 =>  100
2.6.0 : > n = -1_000_000; n.iroot 3 => -100

2.6.0 : > n =  1_000_000; n.roots 3 => [(100.0+0.0i), (-50.0+86.60254038i), (-50.0-86.60254038i)]
2.6.0 : > n = -1_000_000; n.roots 3 => [(50.0+86.60254038i), (-100.0+0.0i), (50.0-86.60254038i)]
```

We see for negative real values assigned to a variable, we don't get the expected **calculator** answer for ood roots of negative reals|integers.

This is totally unexpected, inconsistent, and not desired.

The Crux of the Matter

Every Ruby version includes new methods. Many methods are just aliases (like kernel.self to kernel.then) and some come from Rails, or other sources or inspirations. The point is they are seen as necessary to fix bugs, or provide useful functionality, or just for syntactical sugar.

Here, I empirically show, again, explicit deficiencies in the implementation of some fundamental mathematical operations.

I imagine, most people coming to Ruby and trying to do real math|algorithms that use these operations, and seeing they get wrong|unexpected results, will|have just moved on to languages that will give them correct results. Because I love Ruby, I created a gem to fix them, notified people of the deficiency with squareroots, which ended with Integer.sqrt included in 2.5.

Now, I'm requesting you finishing fixing this problem for all other roots.

The issue, I hope you see, is about the integrity of the language not providing known incorrect results.
I could provide as many error cases as there are possible numbers, but one should be enough.

The absolute benefits of these methods makes doing math|science more accurate, standard, and easy for programmers,
which can only enhance Ruby's reputation and utility in these fields.