Ruby master - Bug #10855

[PATCH] Matrix#inverse returns matrix of integers whenever possible

02/15/2015 07:08 PM - LitoNico (Lito Nicolai)

Status:	Open		
Priority:	Normal		
Assignee:	marcandre (Marc-Andre Lafortune)		
Target version:			
ruby -v:	2.3.0	**Backport:**	2.0.0: UNKNOWN, 2.1: UNKNOWN, 2.2: UNKNOWN

Description

Currently, Matrix#inverse returns a matrix of Rationals, even when each element has a denominator of 1. This leads to

```
> x = Matrix.identity 3
=> Matrix[[1, 0, 0],
          [0, 1, 0],
          [0, 0, 1]]

> x.inverse
=> Matrix[[(1/1), (0/1), (0/1)],
      [(0/1), (1/1), (0/1)],
          [(0/1), (0/1), (1/1)]]
```

Even though Matrix.identity.inverse should be identical to Matrix.identity.

This patch guarantees that Matrix#inverse will return a matrix of integers whenever it can. To maintain uniform types across a matrix, the conversion is only performedif *every* element can be converted to an integer.

History

#1 - 02/15/2015 08:15 PM - marcandre (Marc-Andre Lafortune)

- Assignee set to marcandre (Marc-Andre Lafortune)

Interesting.

I'm thinking it might be best to do the conversion even if some entries are not integral. Why do you feel it's best to have uniform types accross a matrix, in particular when would having an Integer instead of a Rational be a problem?

#2 - 02/15/2015 10:23 PM - LitoNico (Lito Nicolai)

Marc-Andre Lafortune wrote:

> Interesting.
>
> I'm thinking it might be best to do the conversion even if some entries are not integral. Why do you feel it's best to have uniform types accross a matrix, in particular when would having an Integer instead of a Rational be a problem?

In the Matrix class, scalar divison is implemented by using the usual / operation, which loses precision on Integers but not on Rationals. If the Matrix is a mix of the two, something like this will happen:

```
> x = Matrix[[(3/1), 3, 3], [3, (3/1), 3], [3, 3, (3/1)]]
=> # as above
> x / 2
=> Matrix[[(3/2), 1, 1], [1, (3/2), 1], [1, 1, (3/2)]]
```

I would find this mixed precision *really* surprising when writing matrix code! Especially because the loss of precision could be hidden across a number of matrix, vector, and scalar multiplications.

Actually, that's a good argument for returning rationals in ordinary matrix scalar division (and changing this patch as you suggest), but that's out of

line compared to what the rest of Ruby does with division.

#3 - 02/16/2015 10:42 AM - Eregon (Benoit Daloze)

Lito Nicolai wrote:

> Marc-Andre Lafortune wrote:
>
> > Interesting.
> >
> > I'm thinking it might be best to do the conversion even if some entries are not integral. Why do you feel it's best to have uniform types accross a matrix, in particular when would having an Integer instead of a Rational be a problem?

It means every operation that follows must go through rational arithmetic which is likely to be slower and more memory hungry, isn't it?
But of course homogeneity also has its value and the code to lower explicitly Rational to Integer is not exactly nice.

> In the Matrix class, scalar divison is implemented by using the usual /
> operation, which loses precision on Integers but not on Rationals. If
> the Matrix is a mix of the two, something like this will happen:
>
> ```
> > x = Matrix[[(3/1), 3, 3], [3, (3/1), 3], [3, 3, (3/1)]]
> => # as above
> > x / 2
> => Matrix[[(3/2), 1, 1], [1, (3/2), 1], [1, 1, (3/2)]]
> ```
>
> I would find this mixed precision *really* surprising when writing matrix code!
> Especially because the loss of precision could be hidden across a number
> of matrix, vector, and scalar multiplications.

I would think this is a bug. Matrix division by a scalar should be exact, no?

#4 - 02/23/2015 12:03 AM - LitoNico (Lito Nicolai)

Hello! Are there any further thoughts or consensus on which path to take with this?

Here are the options:

1. When dividing matrices, if the resulting matrix has any rational numbers in it, it is entirely rational numbers-- even if they have a divisor of 1.
2. When dividing matrices, the result can have a mix of numeric types, even though this can result in a mix of precise (rational) and imprecise (integral) division in the next operation.
3. Scalar division of a matrix by an integer is patched to return a rational if needed, removing the loss of precision, but breaking with the rest of integral division in Ruby.

I'm happy to write up a patch with any of these changes!

Best,
L

Benoit Daloze wrote:

> Lito Nicolai wrote:
>
> > Marc-Andre Lafortune wrote:
> >
> > > Interesting.
> > >
> > > I'm thinking it might be best to do the conversion even if some entries are not integral. Why do you feel it's best to have uniform types accross a matrix, in particular when would having an Integer instead of a Rational be a problem?
>
> It means every operation that follows must go through rational arithmetic which is likely to be slower and more memory hungry, isn't it?
> But of course homogeneity also has its value and the code to lower explicitly Rational to Integer is not exactly nice.
>
> > In the Matrix class, scalar divison is implemented by using the usual /
> > operation, which loses precision on Integers but not on Rationals. If
> > the Matrix is a mix of the two, something like this will happen:
> >
> > ```
> > > x = Matrix[[(3/1), 3, 3], [3, (3/1), 3], [3, 3, (3/1)]]
> > => # as above
> > > x / 2
> > => Matrix[[(3/2), 1, 1], [1, (3/2), 1], [1, 1, (3/2)]]
> > ```

> I would find this mixed precision *really* surprising when writing matrix code! Especially because the loss of precision could be hidden across a number of matrix, vector, and scalar multiplications.

I would think this is a bug. Matrix division by a scalar should be exact, no?

#5 - 02/23/2015 04:38 PM - marcandre (Marc-Andre Lafortune)

TBH, I can't think of any legitimate use of Matrix#/ with integer division. Anyone?

I never really thought of that, but it's a bit odd that there is no natural way to write Matrix.I(3) / 2, say. There's no quo method on Matrix, so one has to do Matrix.I(3) / 2r, Matrix.I(3) * 0.5r or Matrix.diagonal([0.5r] * 3).

I'm very tempted to change / to act like quo.

Files

matrix_inverse_to_integer.patch	2.14 KB	02/15/2015	LitoNico (Lito Nicolai)