Ruby master - Feature #14151

Make Matrix#[]= public method

12/03/2017 09:45 AM - greggzst (Grzegorz Jakubiak)

Status:	Closed	
Priority:	Normal	
Assignee:	marcandre (Marc-Andre Lafortune)	
Target version:		

Description

I don't even understand why this method hasn't been public since the beginning. I've come to a point when I have to create a matrix in a specific way using row and column indices and I can't use #build with a block because then indices go from the beginning of matrix whereas I have to from the center of the matrix. So what I wanted to do is to create a zero matrix and the fill it in a proper way but I can't without using #[]=. I know I can reopen class and that's what I'm doing but this just doesn't make sense. If we can change elements in an array like so using #[]= then why matrices can't use that as well?

Associated revisions

Revision 65507 - 11/02/2018 05:52 PM - marcandre (Marc-Andre Lafortune)

lib/matrix.rb: Make Matrix & Vector mutable. Add #[]=, #map!.

Adapted from patch by Grzegorz Jakubiak. [#14151] [Fix GH-1769] [Fix GH-1905]

Revision 65507 - 11/02/2018 05:52 PM - marcandre (Marc-Andre Lafortune)

lib/matrix.rb: Make Matrix & Vector mutable. Add #[]=, #map!.

Adapted from patch by Grzegorz Jakubiak. [#14151] [Fix GH-1769] [Fix GH-1905]

History

#1 - 12/03/2017 10:40 AM - shevegen (Robert A. Heiler)

Matrix seems a bit weird ... it does not allow for a Matrix.new either.

The documentation is also lacking, as if the one who wrote it did not
finish writing the documentation:

```
https://ruby-doc.org/stdlib/libdoc/matrix/rdoc/Matrix.html
```

I guess it is kept closely to mathematics ... but ruby is a practical
programming language so I understand that you want to use the methods
that are available there. It's even more curious because you could
use .send() anyway to use these methods

```
Matrix.send :new, [ [25, 93], [-1, 66] ]
```

which then works; I guess #[] = also works via send(), so I agree,
it should just really work.

#2 - 12/03/2017 03:03 PM - greggzst (Grzegorz Jakubiak)

shevegen (Robert A. Heiler) wrote:

> I guess it is kept closely to mathematics ... but ruby is a practical
> programming language so I understand that you want to use the methods
> that are available there. It's even more curious because you could
> use .send() anyway to use these methods
>
> ```
> Matrix.send :new, [ [25, 93], [-1, 66] ]
> ```
>
> which then works; I guess #[] = also works via send(), so I agree,
> it should just really work.

I completely forgot that private method can be invoked using send so I'm using send now. But still it would be better if it was public.

#3 - 12/08/2017 12:55 PM - greggzst (Grzegorz Jakubiak)

I created pull request related to this feature/issue. - https://github.com/ruby/ruby/pull/1769

#4 - 12/08/2017 06:03 PM - Eregon (Benoit Daloze)

- Assignee set to marcandre (Marc-Andre Lafortune)

marcandre (Marc-Andre Lafortune) can you take a look?

#5 - 12/10/2017 11:59 PM - marcandre (Marc-Andre Lafortune)

- Status changed from Open to Feedback

greggzst (Grzegorz Jakubiak) wrote:

> I don't even understand why this method hasn't been public since the beginning.

The library was originally written such that Matrices are immutable. I can't speak to the exact reason for this choice as I'm not the original author, but I suspect it was to go along with the all the other numeric objects being immutable. Makes it more natural to do functional programming too.

The method []= as it is written is also dangerous as it doesn't do any type checking. Try calling m[1..2, 1..2] = 42 for example...

> from the center of the matrix.

I wonder what is that usecase.

> So what I wanted to do is to create a zero matrix and the fill it in a proper way but I can't without using #[]=.

You can build the values from arrays and call Matrix.[]:

```
values = Array.new(5) { Array.new(5, 0) }
values[2][2] = 42
m = Matrix[values]
```

shevegen (Robert A. Heiler) wrote:

> Matrix seems a bit weird ... it does not allow for a Matrix.new either

Indeed. As documented in the source, the official constructors are Matrix.[], rows, columns, build, ...
Note that the private Matrix.new makes no check on the validity of the given arguments.

> The documentation is also lacking

This is not a very constructive comment, nor is it particularly relevant to this issue. Better would be to open a separate issue, in which you enumerate which methods you fell lack documentation. Even better would be to provide a PR with the missing documentation.

#6 - 12/11/2017 08:27 AM - greggzst (Grzegorz Jakubiak)

marcandre (Marc-Andre Lafortune) wrote:

> The library was originally written such that Matrices are immutable. I can't speak to the exact reason for this choice as I'm not the original author, but I suspect it was to go along with the all the other numeric objects being immutable. Makes it more natural to do functional programming too.

I see your point but Arrays are immutable as well following this idea but you can change any element at any position in given Array and there's nothing wrong with that.

> I wonder what is that usecase.

The general use case is the same as in Arrays apart from that you can change Matrix elements in languages like Matlab as well.

> You can build the values from arrays and call Matrix.[]:
>
> values = Array.new(5) { Array.new(5, 0) }
> values[2][2] = 42
> m = Matrix[values]

It's just when you work with Matrix and you want to change one element at the time it doesn't make any sense doing it by creating arrays cause you have to create the whole Matrix again even though you've got some elements already filled in. This is how I see things.

#7 - 12/11/2017 12:13 PM - greggzst (Grzegorz Jakubiak)

marcandre (Marc-Andre Lafortune) wrote:

> The method []= as it is written is also dangerous as it doesn't do any type checking. Try calling m[1..2, 1..2] = 42 for example...

marcandre (Marc-Andre Lafortune) I called it using send and it didn't change a thing in my 3x3 Matrix. So what you probably mean we'd have to add this functionality so it works as in arrays when you pass a range.

My example (at the end I'm setting one element) :

```
m = Matrix[[1,2,3],[1,2,3],[1,2,3]]
=> Matrix[[1, 2, 3], [1, 2, 3], [1, 2, 3]]
m.send(:[]=, 1..2, 1..2, 5)
=> 5
m
=> Matrix[[1, 2, 3], [1, 2, 3], [1, 2, 3]]
m.send(:set_element, 1..2, 1..2, 5)
=> 5
m
=> Matrix[[1, 2, 3], [1, 2, 3], [1, 2, 3]]
m.send(:[]=, 1, 1, 5)
=> 5
m
=> Matrix[[1, 2, 3], [1, 5, 3], [1, 2, 3]]
```

#8 - 12/11/2017 03:48 PM - marcandre (Marc-Andre Lafortune)

greggzst (Grzegorz Jakubiak) wrote:

> I see your point but Arrays are immutable

Arrays are most definitely mutable. If you freeze it, many methods will no longer work, like []=, map!, compact!, etc. No method of Matrix will raise anything because a Matrix is frozen.

> It's just when you work with Matrix and you want to change one element at the time it doesn't make any sense doing it by creating arrays cause you have to create the whole Matrix again even though you've got some elements already filled in. This is how I see things.

Again, I'm not sure what is the use case. You were talking about building a Matrix from the center, I still don't know in what circumstances you'd want to do that. If you already have a matrix and want a new one with some options modified, you can:

```
arrays = some_matrix.to_a
arrays[2][2] = 42
m = Matrix.rows(arrays, false)
```

Of course it's not as easy as calling []=.

greggzst (Grzegorz Jakubiak) wrote:

> marcandre (Marc-Andre Lafortune) wrote:
>
> > The method []= as it is written is also dangerous as it doesn't do any type checking. Try calling m[1..2, 1..2] = 42 for example...
>
> marcandre (Marc-Andre Lafortune) I called it using send and it didn't change a thing in my 3x3 Matrix. So what you probably mean we'd have to add this functionality so it works as in arrays when you pass a range.

Sorry, I meant that m[1, 1..2] = 42 was dangerous, as you no longer have a well formed matrix. The fact that m[1..2, 1..2] = 42 does nothing is also an issue.
So yes, the method []= as currently written doesn't work correctly for ranges.

#9 - 12/11/2017 04:02 PM - marcandre (Marc-Andre Lafortune)

In summary, I'm ok to make Matrix mutable. I'm updating your PR.

#10 - 12/11/2017 05:12 PM - mame (Yusuke Endoh)

Matrix should be kept immutable, IMO. I guess what OP wants is NArray.

#11 - 12/11/2017 07:39 PM - marcandre (Marc-Andre Lafortune)

Thanks mame for the input.

It's a tough case I think, in particular because matrices can be big and thus expensive to dup.

Do you have any extra argument to keep Matrix immutable?

An alternative solution could be a MutableMatrix class

#12 - 12/12/2017 02:08 AM - mame (Yusuke Endoh)

It is difficult for me to explain that all numeric objects are (basically) immutable because it looks obvious to me... "The Ruby Programming Language" by David Flanagan, Yukihiro Matsumoto, says:

> All numeric objects are immutable; there are no methods that allow you to change the
> value held by the object.

I know Matlab and SciPy people are familiar with mutable matrices. For such people, there is NArray.

#13 - 12/12/2017 10:17 AM - Eregon (Benoit Daloze)

Intuitively I agree all numeric *values* should be immutable.

I'm less sure about "collections" of such numeric values, like Matrix or Vector (not just a fixed number of Integer/Float components like Rational/Complex).
All built-in collections in Ruby (Array, Hash, String) are mutable, and they can be made immutable with #freeze.

#14 - 11/03/2018 05:48 AM - greggzst (Grzegorz Jakubiak)

It's been merged and issue can be closed now :)

#15 - 11/03/2018 03:15 PM - marcandre (Marc-Andre Lafortune)

- Status changed from Feedback to Closed