Ruby - Bug #1532

Improved matrix.rb [patch]

05/29/2009 04:18 AM - marcandre (Marc-Andre Lafortune)

Status:	Closed		
Priority:	Normal		
Assignee:	keiju (Keiju Ishitsuka)		
Target version:	1.9.2		
ruby -v:	ruby 1.9.2dev (2009-05-24 trunk 23554) [i386-darwin9.7.0]	**Backport:**	

Description

=begin
The fact that the 'matrix' library is included in Ruby is very helpful.
There are currently some open bugs, and some improvements could make it easier to use.

Propositions:

1. Matrix#trace should raise an ErrDimensionMismatch if the matrix is not square

Mathematically speaking, the trace is only defined for square matrices (ref: wikipedia, mathworld)
Currently, Matrix[[1,2]].trace raises a NoMethodError: undefined method `[]' for nil:NilClass
Note that Matrix[[1,2]].transpose.trace returns 1, although in mathematically, trace(M) = trace(transpose(M))

Raising an ErrDimensionMismatch would bring #trace in line with #inverse

1. Matrix creation methods should perform checks and conversion so that values are stored as an Array of Arrays.

Currently, no checking is done, so the following will all be constructed without an error or a warning:
Matrix[:hello]
Matrix[nil]
Matrix[4], etc...

Later on, confusing results or strange errors will happen. E.g.:
Matrix[42].transpose # ==> Matrix[[42], [0], [0], [0]]

A TypeError should be raised if the argument is not convertible to an Array of Arrays.

Moreover, non arrays that should be converted using :to_ary, to be consistent
with the builtin library methods that accept array arguments (e.g. Array#transpose)

Finally, conversion from Vector to arrays should be always be performed. Currently,
a = Matrix[[1,2],[3,4]]   # => Matrix[[1, 2], [3, 4]]
b = Matrix.rows([a.row(0), a.row(1)]) # => Matrix[Vector[1, 2], Vector[3, 4]]
a == b # => false
It would be more useful, intuitive and mathematically correct if a == b and b.to_s == "Matrix[[1, 2], [3, 4]]"

The same is true for Vector creation methods. For example currently:
Vector.elements(42, false).size # ==> 4
Vector.elements(Vector[1,2,3]) == Vector.elements([1,2,3]) # ==> false
It would be more useful, intuitive and correct if the first example raises an error and the second returns true

1. Matrix creators should enforce that a matrix is rectangular.

Currently, a matrix with irregular rows can be created, e.g. x = Matrix[[1,2],[3]].
Mathematically speaking, that is not a matrix.
Basically none of the Matrix methods are of any use for non-rectangular matrices.
Moreover, many strange errors can occur later on. For example, x.inverse will
raise a "NoMethodError: undefined method `*' for nil:NilClass"

It would be helpful to catch these cases at creation time.
Many creation methods don't have to make any extra checks (e.g. Matrix.scalar), and
all methods of Matrix can bypass this extra check when they have created the arrays themselves

(e.g. Matrix#*). There would be a small cost for creation using Matrix.[] and Matrix.rows,
although it is in O(rows) while most other operations are usually in O(rows x columns),
so the performance difference would be minimal.

1. Matrix should deal with empty matrices.

Currently, empty matrices like Matrix[] cause problem.
For example Matrix[].determinant raises an error, so does Matrix[] * Matrix[].

Moreover, if h = Matrix[[], []], then currently h.transpose.transpose != h

While an alternative would be to raise and error, the best solution is to handle
empty matrices properly, both 0x0, 0xn and nx0, as does MatLab, GNU Octave, etc...
See doc and references to the literature in:
http://www.gnu.org/software/octave/doc/interpreter/Empty-Matrices.html

1. Out of bound indices should be dealt with.

a) Matrix#[row,col] should behave in a consisten way if either row or col is out of bounds.
Currently it returns nil vs raises an obscure error (See redmine #1518.)

b) Matrix[[1]].row(2) raises an obscure error, while Matrix[[1]].column(2) returns Vector[nil, nil]

c) In a similar vein, Matrix[[1]].minor(2..2,1..1) currently raises an error but not
Matrix[[1]].minor(1..1,2..2)

Solutions:
a) To be consistent with array lookup using [], Matrix#[] should return nil for out of bounds elements.
A #fetch method could be added, but the fact that matrices normally won't contain nil or false
makes it easy to deal with out of bounds references, e.g. m[r, c] || 0

b) Contrary to nil, it is not easy nor useful to deal with Vector[nil, nil, ...].
#row, and #column could raise an IndexError, but it is more useful and
more coherent with Array#at, etc... to return nil.

c) The same way Matrix#row and #col can be related to Array#at,
Matrix#minor should have similar semantics to Array#slice. If either starting point
is out of bounds, nil is returned. Otherwise a Matrix is returned, although it can
be smaller than what was requested. This is similar to
[:a, :b].slice(3..10) # => nil
[:a, :b].slice(2..10) # => []
[:a, :b].slice(1..10) # => [:b]
Matrix[[1], [2]].minor(0..10, 2..10) # => nil
Matrix[[1], [2]].minor(0..10, 1..10) # => Matrix[[], []]
Matrix[[1], [2]].minor(1..10, 0..10) # => Matrix[[2]]

1. Matrix#collect, Vector#collect, #collect2, #map2 should return enumerators if no block is given

This would be more useful and is consistent with Array#each, etc...

1. Matrix#hash should have less collisions

Currently, the following matrices have the same hash:
Matrix[]
Matrix[[0,0], [0,0]]
Matrix[[1,0], [0,1]]
Matrix[[42,42], [666,666]]
Matrix[[1,2,3,4], [5,6,1,2], [3,4,5,6]]

Ideally, these should have different hashes, since they are different matrices.

1. Matrix#compare_by_row_vectors, Vector#compare_by and Vector#init_elements should be made private or disappear.

As per the documentation, these are not meant to be used.
As such it would be best if they didn't appear in the list of methods.

The attached patch addresses all these issues.

Moreover, it addresses all matrix-related issues I could find on redmine:
http://redmine.ruby-lang.org/issues/show/1020
http://redmine.ruby-lang.org/issues/show/1515
http://redmine.ruby-lang.org/issues/show/1516
http://redmine.ruby-lang.org/issues/show/1517
http://redmine.ruby-lang.org/issues/show/1518
http://redmine.ruby-lang.org/issues/show/1526
http://redmine.ruby-lang.org/issues/show/1531

Also fixed a bug with #determinant and #determinant_e that would raise an error for some matrices
(for instance any square matrix with m[0][0] == 0, e.g. Matrix[[0,1],[2,3]].determinant # => error raised)

Finally, the following methods are performing faster:
Matrix#collect
Matrix#transpose
Matrix#==
Matrix#eql?
Matrix#hash
Vector#collect
Vector#map2

Note that the branch 'runpaint' of rubyspecs has specs to this patch.
=end

History

#1 - 07/13/2009 11:43 PM - yugui (Yuki Sonoda)

- Assignee set to keiju (Keiju Ishitsuka)

=begin

=end

#2 - 09/17/2009 02:03 PM - marcandre (Marc-Andre Lafortune)

- File f_matrix_access.diff added

- File a_matrix_creation.diff added

- File b_matrix_empty.diff added

- File c_matrix_trace.diff added

- File d_matrix_cleanup.diff added

- File e_matrix_enum.diff added

=begin
Here is my previous patch simplified and broken down in 6 successive patches (a-f)

*** Summary ***

The following changes are completely compatible for code that used the library in a mathematically sound manner. Potential incompatibilities would arise only for usage that does not make mathematical sense.

- API changes *

(a) Argument checking for matrix creations

Current: "although matrices should theoretically be rectangular, this is not enforced by the class" (as per documentation)

Change to: "matrices must be rectangular, otherwise an ErrDimensionMismatch is raised."

This would make the library behave in a sane manner, with error messages that are dependable and follow the established principle of detecting errors early [ruby-core:23664]

Patch attached: a_matrix_creation.diff

(b) Handle edge case Matrices

Like Mathematica, MatLab and others, the library should deal properly with matrices with a number of columns or rows of 0.

Patch attached: b_matrix_empty.diff

(c) Matrix#trace should raise an ErrDimensionMismatch if the matrix is not square

Mathematically speaking, the trace is only defined for square matrices (ref: wikipedia, mathworld)
Raising an ErrDimensionMismatch would bring #trace in line with #inverse

Patch attached: c_matrix_trace.diff

(d) Matrix#compare_by_row_vectors, Vector#compare_by and Vector#init_elements should be made private or disappear.

As per the documentation, these are not meant to be used.

Patch attached: d_matrix_cleanup.diff

- Compatible API changes *

(e) Enumerators should be returned when no blocks are given for methods like #map, etc...

This would bring the library in line with Ruby 1.8.7+ which returns enumerators in such cases

Patch attached: e_matrix_enum.diff

(f) Consistent results when accessing elements with out of bounds indices.

This would make the library in line with Ruby 1.8.7+ which returns enumerators in such cases

Patch attached: f_matrix_access.diff

*** Detailed examples of problems ***

(a) Argument checking for matrix creations

1. Plain wrong arguments:
   Matrix[nil]   # => no expection is raised
   Matrix[:hello] # => no expection is raised
   Matrix[4]   # => no expection is raised

Later on, confusing results or strange errors will happen. My two favorites:

Matrix[:hello].rank # ==> NoMethodError: undefined method `quo' for "e":String

Matrix[42].transpose # ==> Matrix[[0], [1], [0], [1]]
or Matrix[[0], [1], [0], [1], [0], [1], [0], [0]]
(depending on the platform)

1. Most methods will result the wrong result or raise an unintuitive error in case of uneven rows

Matrix[[1,2],[3]].rank # ==> NoMethodError: undefined method `-' for nil:NilClass
Matrix[[0,0],[0,0]] + Matrix[[1,2], [3,4,5,6,7,8]] # ==> does not raise error
Matrix[[1,2],[3]].square? # ==> true

1. Array-like arguments

Moreover, basic conversion to arrays should be attempted, in particular from Vectors:

a = Matrix[[1,2],[3,4]]   # => Matrix[[1, 2], [3, 4]]
b = Matrix.rows([a.row(0), a.row(1)]) # => Matrix[Vector[1, 2], Vector[3, 4]]
a == b # => false

The attached patch matrix_creation.diff changes the behaviour and documentation to insure that arguments passed to the Matrix creators are correct.

(related to redmine issues #1517, 1515)

(b) Handle edge case Matrices

Matrix[].row_size # ==> 0, ok!
Matrix[].column_size # ==> raise an error, should be 0
Matrix[].determinant # ==> raise an error, should be 1
m = Matrix[[], []]
m.transpose.transpose == m # ==> false, should be true for any m

While an alternative would be to raise and error, the best solution is to handle empty matrices properly, both 0x0, 0xn and nx0, as does Mathematica, MatLab, GNU Octave, etc...
See doc and references to the literature in:
http://www.gnu.org/software/octave/doc/interpreter/Empty-Matrices.html

(c) Matrix#trace should raise an ErrDimensionMismatch if the matrix is not square

Matrix[[1],[2]].trace # ==> returns 1, which makes no sense mathematically
Matrix[[1,2]].trace   # ==> raises a NoMethodError: undefined method `[]' for nil:NilClass

(d, e) Enumerators should be returned when no blocks are given for methods like #map, etc...

No additional comment

(f) Consistent results when accessing elements with out of bounds indices.
m = Matrix[[1]] # Example with 1x1 matrix

m[2,0] # ==> NoMethodError: undefined method `[]' for nil:NilClass
m[0,2] # ==> nil

m.row(2)   # ==> TypeError: can't dup NilClass
m.column(2) # ==> Vector[nil]

m.minor(2..2, 0..0) # ==> NoMethodError: undefined method `collect' for nil:NilClass
m.minor(0..0, 2..2) # ==> Matrix[nil]

Accessing elements should behave in a consistent way if either row or col is out of bounds
Moreover, Matrix[nil] is not a matrix!

Solutions:

1. To be consistent with array lookup using [], Matrix#[] should return nil for out of bounds elements.

2. #row, and #column should return nil, to be coherent with with Array#at, etc... and be most useful.

3. The same way Matrix#row and #col can be related to Array#at,
   Matrix#minor should have similar semantics to Array#slice. If either starting point
   is out of bounds, nil is returned. Otherwise a Matrix is returned, although it can
   be smaller than what was requested. This is similar to
   [:a, :b].slice(3..10) # => nil
   [:a, :b].slice(2..10)   # => []
   [:a, :b].slice(1..10)   # => [:b]
   Matrix[[1], [2]].minor(0..10, 2..10) # => nil
   Matrix[[1], [2]].minor(0..10, 1..10) # => Matrix[[], []]
   Matrix[[1], [2]].minor(1..10, 0..10) # => Matrix[[2]]

(see redmine #1518)

Cleanup, optimizations and bug fixes present in the original patch have already been committed.

*** Closing notes ***

Numerous and very basic bugs have been present for a long time (issues #1020, 1531, 2106, 2107, 2108 all fixed in upcoming versions of Ruby). This seem to indicate that very few people, if any, actually use the matrix library.

Nevertheless I believe that fixing it is important. Moreover I think we should not hesitate in modifying the API for edge cases as long as maintain compatibility for the potential few users using the library correctly.

I would be happy and honored to be the maintainer of this library.

=end

#3 - 10/20/2009 10:37 PM - keiju (Keiju Ishitsuka)

- Status changed from Open to Closed

- % Done changed from 0 to 100

=begin
This issue was solved with changeset r25412.
Marc-Andre, thank you for your reporting of the issue.
You have greatfully contributed toward Ruby.
May Ruby be with you.

=end

#4 - 10/21/2009 04:14 AM - marcandre (Marc-Andre Lafortune)

- Status changed from Closed to Open

=begin
Hi

I am very happy that all of my 6 patches have been accepted and applied. Thank you for taking the time to review them.

I noticed 4 differences:

*** Introduction of Matrix.empty ***

Great addition!

I feel the documentation example could be more informative. It also should not show information about the way the matrix library stores the information internally; this information should not be relied upon. (r25422)

I took the liberty to modify the documentation to provide examples that I hope help get a better understanding of empty matrices and can be relied upon.

*** Matrix.columns was modified from my version ***

That new version was causing 2 type of bugs:

Matrix.columns [[],[],[]]
==> Matrix.empty(0, 0), should be Matrix.empty(0, 3)

Matrix.columns nil
==> raises NoMethodError 'transpose' instead of TypeError

I modified Matrix.columns back to my original version to address these. (r25423)

*** to_s, inspect & inspect_org ***

to_s & inspect now refer to Matrix.empty, much nicer than my original proposal.

A problem was that #inspect was still aliased to #to_s, so the code in "def inspect" was actually never called. Fixed in r25423.

(a) I feel that the introduction of #inspect_org is bad. The names of the instance variables are implementation details that should not be public. If you decide nevertheless to keep this method, it should be documented.

*** Reintroduction of #compare_by_row_vectors and #compare_by ***

These two methods were documented as "not meant for public consumption". Now they have no documentation at all.

(b) They should either be removed (they are not used by the implementation anymore) or be documented.

Since these patches were more than just a "bug fix", I also summarized the changes in the ChangeLog. (r25421)

(c) Is there a reason why these changes have not also been made to 1.8.8's lib/matrix?

I will bring Run Paint's RubySpecs up to date.

Thanks

=end

#5 - 10/24/2009 03:32 PM - marcandre (Marc-Andre Lafortune)

- Status changed from Open to Closed

=begin
I'm closing this and moving the what remains to be done to a new issue.

RubySpec for lib/matrix have been brought up to date.

Some additional specs revealed three bugs, fixed in r25450, r25451 and r25452.
=end

Files

matrix.patch	21.2 KB	05/29/2009	marcandre (Marc-Andre Lafortune)
f_matrix_access.diff	1.15 KB	09/17/2009	marcandre (Marc-Andre Lafortune)
a_matrix_creation.diff	6.29 KB	09/17/2009	marcandre (Marc-Andre Lafortune)
b_matrix_empty.diff	3.53 KB	09/17/2009	marcandre (Marc-Andre Lafortune)

c_matrix_trace.diff	310 Bytes	09/17/2009	marcandre (Marc-Andre Lafortune)
d_matrix_cleanup.diff	2.18 KB	09/17/2009	marcandre (Marc-Andre Lafortune)
e_matrix_enum.diff	1.66 KB	09/17/2009	marcandre (Marc-Andre Lafortune)