A Guide to Matrix+ for MA1522

MatrixPlus_v3, UserGuide_v1
github.com/reidenong/MatrixPlusForMA1522

In essence, Matrix+ is a wrapper class for matrices in MatLab, and it provides extra functionalities to the existing matrices in MatLab. I wrote this class to cut down on time in both implementing and interpreting code in exam conditions.

```
%% Regular Matlab
>> A(2, :) = A(2, :) - 4*A(1, :);
>> A(3, :) = 3 * A(3, :)
>> A([1,2], :) = A([2,1],:)
>> null([A'*A A'*b])
```

```
%% Matrix+
>> A = A.rowOp("R2 += 4R1")
        .rowOp("R3 *= 3")
        .rowOp("R2 <-> R1")
>> A.leastSquare(b)
```

Matrix+ provides intuitive, easy to use functions that provide more natural ways to manipulate matrices. The class architecture also allows for easy chaining of functions, which is useful for keeping track of complicated expressions.

Contents

1. Matrix+ as a wrapper class

For those unfamiliar with OOP, Matrix+ wraps around MatLab's "vanilla" matrix, and provides extra functions and capabilities that I have found useful while behaving almost identically to their vanilla counterparts as a immutable matrix.

Note that these functions are also available as standalone functions that can work with vanilla matrices. However, the Matrix+ wrapper provides a more convenient way to access these functions.

Creating a Matrix

In short, there is a difference between a Matrix+ matrix and a vanilla matrix.

```
>> a = [1 0 0; 0 1 0; 0 0 1];    % vanilla matrix
>> A = Matrix(a);                % Matrix+ matrix
```

We can create a Matrix+ matrix by simply using `Matrix()` on a normal matrix. Our Matrix+ Object can then access all of our Matrix+ functions. Some common functions and their Matrix+ counterparts are shown below:

Vanilla `a`	Matrix+ `A`
`rref(a)`	`A.rref()`
`det(a)`	`A.det()`
`inv(a)`	`A.inv()`
`rank(a)`	`A.rank()`
`null(a)`	`A.null()`
`adjoint(a)`	`A.adj()`
`transpose(a)`	`A.transpose()`
`sym(a)`	`A.sym()`
`norm(a)`	`A.norm()`

We can convert between Matrix+ and vanilla matrices easily. Matrix+ matrices display exhibit a `M+` notation at the upper left hand side.

```
>> a = [1 2; 3 4];
>> A = Matrix(a)    % creates a Matrix+ object
A =
M+
       1             2
       3             4

>> b = A.get();     % returns the vanilla matrix
```

2. Matrix Manipulation and Operations

Extracting Rows and Columns

Matrix+ provides a convnient way to extract rows and columns from a matrix, through means of `getRow()` and `getCol()`. These functions return a Matrix+ object.

```
>> A = Matrix([1 2 3; 4 5 6; 7 8 9]);
>> row1 = A.getRow(1)

row1 =
M+
       1              2              3

>> col2 = A.getCol(2)

col2 =
M+
       2
       5
       8
```

$+ - * \hat{}$ operands

Matrix+ matrices behave exactly like vanilla matrices. A list of these operations are provided here. For all Matrix+ Matrices `A` and `B`,

`A + B`	Matrix addition
`A * B`	Matrix Multiplication
`A * c`	Scalar Multiplication
`A^c`	Matrix Powers, ie. A*A ...
`A.^c`	Element wise powers
`[A B]`	Horizontal Concatenation
`[A; B]`	Vertical Concatenation

Currently there is limited support for operations between Matrix+ matrices and vanilla matrices, eg. `A + a`. While it should work in most cases, ie. I haven't come across a case where it has yet to to work, I would recommend keeping to Matrix+ matrices to be safe for now, until I can rigorously extend the support for vanilla matrices.

3. Elementary Row Operations

Matrix+ provides a intuitive way to conduct row operations on a given matrix. This is through the use of `rowOp()`, which allows the client to use verbose arguments to describe the row operation, making it easier to comprehend. Note that all `()` arguments are in string form, and requires the use of `" "`.

Row swap

For a swap between rows X and Y, we use the following syntax `rowOp("X <-> Y")`.

```
>> A = Matrix([1 2 3; 4 5 6; 7 8 9])
>> A = A.rowOp("R2 <-> R3")

    Row Swap:
        R2 <-> R3

A =
M+
       1              2              3
       7              8              9
       4              5              6
```

Row Scalar Multiplication

To multiply row X by scalar factor c, we do `rowOp("RX *= c")`.

```
>> A = Matrix([1 2 3; 4 5 6; 7 8 9]);
>> A = A.rowOp("R2 *= 3")

    Scalar Multiplication:
        R2 -> 3 * R2

A =
M+
       1              2              3
      12             15             18
       7              8              9
```

Row Addition

To add row X to row Y with factor c, we do `rowOp("RX += cRY")`.

```
>> A = Matrix([1 2 3; 4 5 6; 7 8 9]);
>> A = A.rowOp("R2 += 10 R3")

    Row Addition:
        R2 += 10 * R3

A =
M+
```

```
       1             2             3
      74            85            96
       7             8             9
```

Stacking functions

We can stack multiple row operations together, and execute them in one go, similar to how we stack operations in functional programming.

```
>> A = Matrix([1 2 3; 4 5 6; 7 8 9]);
>> A = A.rowOp("R2 += 10 R3").rowOp("R2 <-> R3").rowOp("R2 *= 3")

    Row Sum:
        R2 += 10 * R3

    Row Swap:
        R2 <-> R3

    Scalar Multiplication:
        R2 -> 3 * R2

A =
M+
       1             2             3
      21            24            27
      74            85            96
```

Standalone functions

In the event that one would rather use vanilla matrices, there are standalone functions that we can use. With vanilla matrix a, we can use the following functions to get vanilla matrix result by:

Row swap

```
result = swap(matrix a, int row1, int row2)
```

Row Scalar Multiplication

```
result = mult(matrix a, int row, int factor)
```

Row Addition

```
result = rsum(matrix a, int row1, int row2, int factor)
% returns row1 + factor * row2
```

4. Orthogonal functions

Gram-Schmidt

`gramSchmidt(matrix A)` uses QR Decomposition to turn a collection of linearly independent column vectors into a basis of orthonormal vectors. Column vectors `u1`, `u2`, … are pased in as a matrix `U` where `U = (u1 u2 u3 ... )`. Product is not guaranteed when vectors passed in are not already linearly independent, and the program will throw a warning.

MA1522 Lecture 6, AY23/24 S1

Q: Convert $S = \left\{ \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ 2 \end{pmatrix} \right\}$ into an orthonormal basis for $\mathbb{R}^3$.

A: We use the Gram-Schmidt process.

```
>> A = Matrix.of([1 1 1; 2 1 1; 1 1 2]);
>> A = A.gramSchmidt()
v1 has factor 6^(1/2)/6

1
2
1

v2 has factor 3^(1/2)/3

 1
-1
 1


v3 has factor 2^(1/2)/2

-1
 0
 1

A =
M+
[6^(1/2)/6,  3^(1/2)/3, -2^(1/2)/2]
[6^(1/2)/3, -3^(1/2)/3,          0]
[6^(1/2)/6,  3^(1/2)/3,  2^(1/2)/2]
```

So $\left\{ \frac{1}{\sqrt{6}} \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix}, \frac{1}{\sqrt{3}} \begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix}, \frac{1}{\sqrt{2}} \begin{pmatrix} -1 \\ 0 \\ 1 \end{pmatrix} \right\}$ is a orthonormal basis for $\mathbb{R}^3$.

In the event where we need the economical Gram Schmidt, we can instead use `gramSchmidtEcon()`.

Dot Product

As of 5-Dec-23, the dot product function has yet to be implemented into Matrix+. However, it is easily accessible through `dot(A.get(), B.get())` returns the dot product of two vectors. Note that the two vectors must be column vectors, and the function will throw an error if the two vectors are not of the same length.

Least Square Solutions

`A.leastSquare(b)` returns the least square solution of the system $Ax = b$.

MA1101 Finals Q3, AY19/20 S1

Q: Let

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & -1 & 1 \end{pmatrix}, b = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \end{pmatrix}$$

(a) Show the linear system $Ax = b$ is inconsistent.
(b) Find the least squares solution of the system in (i).
(c) Find the projection p of b onto the column space of A.
(d) Find the smallest possible value of $\|Av - b\|$ among all vectors $v \in \mathbb{R}^3$
(e) Note that the three columns of A form a orthogonal set. Extend this set to an orthogonal basis for $\mathbb{R}^4$.

A:

(a) We want to show `rref(A | b)` is inconsistent.

```
>> A = Matrix.of([1 0 0; 0 1 1; 1 0 0; 0 -1 1]);
>> b = Matrix.of([1; 1; 0; 0]);
>> Matrix([A b]).rref()
```

As our last result is inconsistent, the system is inconsistent.

(b)

```
>> A.leastSquare(b)
Solving for A'Ax = A'b, we do rref([A'*A A'*b]) to get solution set:

ans =
M+
[1, 0, 0, 1/2]
[0, 1, 0, 1/2]
[0, 0, 1, 1/2]
```

So $x = \begin{pmatrix} \frac{1}{2} \\ \frac{1}{2} \\ \frac{1}{2} \end{pmatrix}$ is the least square solution.

(c) The projection p of b onto the column space of A is given by $A * \hat{b}$ where $\hat{b}$ is the least square solution of $Ax = b$. So

```
>> A * A.leastSquare(b).getCol(4)
Solving for A'Ax = A'b, we do rref([A'*A A'*b]) to get solution set:

ans =
M+
1/2
  1
1/2
  0
```

So $proj_a = \begin{pmatrix} \frac{1}{2} \\ 1 \\ \frac{1}{2} \\ 0 \end{pmatrix}$.

(d) The smallest possible value of $\|Av - b\|$ is given by $\|proj_a - b\|$, which is

```
>> proj = A * A.leastSquare(b).getCol(4);      % from part(c)
>> Matrix(proj - b).norm()
ans =
  2^(1/2)/2
```

So our answer is $\frac{1}{\sqrt{2}}$.

(e) We can add vector

$$p - b = \begin{pmatrix} -\frac{1}{2} \\ 0 \\ \frac{1}{2} \\ 0 \end{pmatrix}$$

which is orthogonal to the column space of A.

5. Eigenvalues & Eigenvectors

Finding Eigenvalues

`.findEigVals()` allows us to easily find the sorted eigenvalues of a given matrix. This is complemented by `.findEigPoly()`, which gives the characteristic polynomial of the matrix for exam working.

Finding Eigenspaces

Once we have found the relevant eigenvalues from the previous step, we can then use it as a input argument for `.findEigSpace(int eigenvalue)`. This function returns a basis for each eigenspace.

MA2001 Finals Q4, AY22/23 S1
Q: We are given that matrix C has two eigenvalues 0 and 3, and

$$C = \begin{pmatrix} 2 & 1 & -1 & 0 \\ 1 & 2 & 1 & 0 \\ -1 & 1 & 2 & 0 \\ 0 & 0 & 0 & 3 \end{pmatrix}$$

(a) Find an orthonormal basis for the eigenspace E_0 of C. (4 marks)
(b) Find an orthonormal basis for the eigenspace E_3 of C. (4 marks)
(c) Based on the results of (a) and (b), write down the characteristic polynomial of C. (2 marks)
(d) Write down two 4×4 matrices P and D such that P is an orthogonal matrix, D is a diagonal matrix, and $D = P^T CP$. (4 marks)

A:
(a) & (b)

```
>> A = Matrix.of([2 1 -1 0; 1 2 1 0; -1 1 2 0; 0 0 0 3]);
>> A. findEigSpace(0)
ans =
M+
 1
-1
 1
 0

>> A.findEigSpace(3)
ans =
M+
[1, -1, 0]
[1,  0, 0]
[0,  1, 0]
[0,  0, 1]
```

We get our answer by normalising each column vector. Alternatively, this can be done instantly by

```
>> A.findEigSpace(0).gramSchmidtEcon();
>> A.findEigSpace(3).gramSchmidtEcon();
```

Either way, we get the same answer:

$$E_0 = \left\{ \frac{1}{\sqrt{3}} \begin{pmatrix} 1 \\ -1 \\ 1 \\ 0 \end{pmatrix} \right\}$$

$$E_3 = \left\{ \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \frac{1}{\sqrt{6}} \begin{pmatrix} -1 \\ 1 \\ 2 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \right\}$$

(c) $\dim(E_0) = 1$, $\dim(E_3) = 3$, so the characteristic polynomial is $x(x-3)^3$. We can verify this by

```
>> A.findEigPoly()
det(xI - A) =
(x - 3)*(x^3 - 6*x^2 + 9*x)
[x - 3, x - 3, x - 3, x]
```

(d) We can use the basis vectors from (a) and (b) to form the matrix P, and the eigenvalues to form the diagonal matrix D. Then

$$P = \left\{ \frac{1}{\sqrt{3}} \begin{pmatrix} 1 \\ -1 \\ 1 \\ 0 \end{pmatrix}, \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \frac{1}{\sqrt{6}} \begin{pmatrix} -1 \\ 1 \\ 2 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \right\} = \begin{pmatrix} \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{6}} & 0 \\ -\frac{1}{\sqrt{3}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{6}} & 0 \\ \frac{1}{\sqrt{3}} & 0 & \frac{2}{\sqrt{6}} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$D = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 3 \end{pmatrix}$$

We can verify this quickly using Matrix+ by

```
>> P = [A.findEigSpace(0).gramSchmidtEcon()
       A.findEigSpace(3).gramSchmidtEcon()];
>> D = diag([0 3 3 3]);
>> P * D * P.transpose()
```

Diagonalizations

With the previous question in mind, Matrix+ also has a front to end function that diagonalizes a matrix.

```
>> A = Matrix.of([2 1 -1 0; 1 2 1 0; -1 1 2 0; 0 0 0 3]);
>> [P D] = A.findDiag()
    Using [P D] = findDiag(A),
    A = P x D x P^-1 where

P =
M+
[ 1, 1, -1, 0]
[-1, 1,  0, 0]
[ 1, 0,  1, 0]
[ 0, 0,  0, 1]

D =
M+
[0, 0, 0, 0]
[0, 3, 0, 0]
[0, 0, 3, 0]
[0, 0, 0, 3]
```

To complete the answer, we need to find a orthonormal basis for P. This can be done by

```
>> P = P.gramSchmidtEcon();
```

Standalone functions

For vanilla matrices, the following methods are available.

Finding Eigenvalues

```
>> eig(sym(matrix a))
>> findEigPoly(matrix a)
```

Finding Eigenspaces

```
>> findEigSpace(matrix a, int eigenvalue)
```

Diagonalizations

```
>> [P D] = findDiag(matrix a)
```

6. Matrix Decompositions / Factorizations

LU Decomposition

The native `lu()` function in MatLab is pretty much all you need, and I saw no further value in writing my own method.

```
>> [L U] = lu(sym(matrix a))
```

QR Decomposition

QR Decomposition can be done by `qr()`, another native MatLab function. Again, no added value in adding my own version.

```
>> [Q R] = qr(sym(matrix a))
```

Singular Value Decomposition

While SVD has a native function, much of SVD in a exam scenario is in the method of finding the SVD, rather than the result itself. As such, `autoSVD()` is made specially to generate working along the way such that the user does not need to perform further steps to get presentation marks.

```
>> [U S V] = A.autoSVD()
>> [U S V] = autoSVD(matrix a)
```

While `autoSVD()` is pretty straightforward, there might be certain twists in the given question that prevent a direct application of `autoSVD()` or other similar SVD solvers.

MA1522 Finals Q18, AY23/24 S1
Q: Let $\mathbf{A}$ be a 5×3 matrix. Suppose a QR factorization of $\mathbf{A}$ is given by $\mathbf{A} = \mathbf{QR}$, where

$$\boldsymbol{R} = \begin{pmatrix} \sqrt{3} & 0 & 0 \\ 0 & \sqrt{3} & -\frac{1}{\sqrt{3}} \\ 0 & 0 & 2\sqrt{\frac{2}{3}} \end{pmatrix}.$$

(a) Compute $\boldsymbol{A^T A}$. Give exact values and explain how you derive your answers.
(b) Suppose a singular-value decomposition of A is $A = U\Sigma V^T$, where

$$V = \begin{pmatrix} * & 1 & * \\ * & * & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & 0 & * \end{pmatrix}.$$

Find Σ and V. Give exact values. You may use MATLAB to aid in your computation, but explain how you derive your answers.
(c) Let U, Σ, V be given in (b). Given that

$$U = \begin{pmatrix} 0 & \frac{1}{\sqrt{3}} & 0 & -\frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} \\ 0 & 0 & 0 & -\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ 0 & 0 & 1 & 0 & 0 \\ -\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{3}} & 0 & -\frac{1}{\sqrt{12}} & \frac{1}{\sqrt{12}} \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{3}} & 0 & \frac{1}{\sqrt{12}} & -\frac{1}{\sqrt{12}} \end{pmatrix}$$

find the matrix $\mathbf{A}$.

A:
(a)

$$A^TA = (QR)^TQR = R^T(Q^TQ)R = R^TR = \begin{pmatrix} 3 & 0 & 0 \\ 0 & 3 & -1 \\ 0 & -1 & 3 \end{pmatrix}.$$

(b) `autoSVD()` faces a issue here. Usually for SVD, given a matrix $\mathbf{A}$, we will find $\boldsymbol{A^TA}$ before proceeding to find its eigenvectors and so on. However, in this case, we are given $\boldsymbol{A^TA}$ instead, which I suspect prevents most people from blindly using solvers. However, a workaround is to directly edit the source code of `autoSVD()` to directly take in $\boldsymbol{A^TA}$ as a input argument.

```
%% in autoSVD.m
function [U S Vt] = autoSVD(matrix)
ori = matrix;
matRank = rank(matrix);
[rows, cols] = size(matrix);
% matrix = matrix'*matrix;    % comment out this line
eigenvalues = eig(sym(matrix))';
.
.
.
```

So now `autoSVD()` will take in $\boldsymbol{A^TA}$ as a input argument, and we can just copy paste the answer and workings from Matrix+.

```
>> AtA = Matrix([3 0 0; 0 3 -1; 0 -1 3])

AtA =
M+
     3     0     0
     0     3    -1
     0    -1     3
```

```
>> [U S Vt] = AtA.autoSVD()
For given matrix, we first find A' * A:
     3     0     0
     0     3    -1
     0    -1     3

For given matrix, we have characteristic polynomial:

det(xI - A) =
x^3 - 9*x^2 + 26*x - 24

[x - 3, x - 4, x - 2]

This gives us sorted eigenvalues:
[4, 3, 2]

We now obtain S as a diagonal matrix of the sqrt of nonzero eigenvalues
padded with 0s
since matrix is rank 3, we have S:
S =

[2,       0,       0]
[0, 3^(1/2),       0]
[0,       0, 2^(1/2)]

Now we find a unit eigenvector for each eigenvalue.
for our #1 eigenvalue of _4_, we have unit ...
.
.   % workings ommitted for User Guide
.
Placing them in a vector V = (v1 v2 v3 ...) while getting the unit vector
for each vector, we have V:

V =
[         0, 1,         0]
[-2^(1/2)/2, 0, 2^(1/2)/2]
[ 2^(1/2)/2, 0, 2^(1/2)/2]

Now we find a orthonormal basis for the solution space.
using formula u_i = 1/sqrt(eigenvalue) * A * v_i, we have

.
.  % workings ommitted for User Guide
.

Now we use the gram Schmidt process to find a orthonormal basis of U

.
.  % workings ommitted for User Guide
.
```

```
SVD algorithm is finished.
  We have A = U * S * VT, where

U =
M+
[         0, 1,         0]
[-2^(1/2)/2, 0, 2^(1/2)/2]
[ 2^(1/2)/2, 0, 2^(1/2)/2]


S =
M+
[2,       0,       0]
[0, 3^(1/2),       0]
[0,       0, 2^(1/2)]


Vt =
M+
[0, -2^(1/2)/2, 2^(1/2)/2]
[1,          0,         0]
[0,  2^(1/2)/2, 2^(1/2)/2]

>> V = Vt.transpose();
```

There are several things to note. Given their definition of $\mathbf{V}$, we want to multiply the first col of $\mathbf{V}$ by -1. This is still logical as both cols are in the eigenspace of eigenvalue 2. We also need to extend Σ to a 5×3 matrix as $\mathbf{A}$ is given to be a 5×3 matrix. So from `autoSVD()` we have

$$V = \begin{pmatrix} 0 & 1 & 0 \\ \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \end{pmatrix}, \Sigma = \begin{pmatrix} 2 & 0 & 0 \\ 0 & \sqrt{3} & 0 \\ 0 & 0 & \sqrt{2} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}.$$

(c) We just copy in the given U and Σ from (b), and we can get A by

```
>> U = Matrix.of([0 1/sqrt(3) 0 -1/sqrt(3) 1/sqrt(3); 0 0 0 -1/sqrt(2) -1/
sqrt(2); 0 0 1 0 0; -1/sqrt(2) -1/sqrt(3) 0 -1/sqrt(12) 1/sqrt(12); -1/
sqrt(2) 1/sqrt(3) 0 1/sqrt(12) -1/sqrt(12)]);
```

```
>> U * S * V.transpose()

ans =
M+
[ 1,  0, 0]
[ 0,  0, 0]
[ 0,  1, 1]
[-1, -1, 1]
[ 1, -1, 1]
```

So we have

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 1 \\ -1 & -1 & 1 \\ 1 & -1 & 1 \end{pmatrix}.$$

Using Matrix+, this question took me < 5 minutes to complete. I would estimate it would have taken > 15 minutes to complete otherwise, which makes sense with a total mark value of 12.