The lualinalg Package

Chetan Shirore* and Ajit Kumar

August 4, 2023

1 Introduction

The `lualinalg` package is developed to perform operations on vectors and matrices defined over the field of real or complex numbers inside LaTeX documents. It provides flexible ways for defining and displaying vectors and matrices. No particular environment of LaTeX is required to use commands in the package. The package is written in Lua, and tex file is to be compiled with the LuaLaTeX engine. The time required for calculations is not an issue while compiling with LuaLaTeX. There is no need to install Lua on the user's system as TeX distributions (TeXLive or MikTeX) come bundled with LuaLaTeX. It may also save users' efforts to copy vectors and matrices from other software (which may not be in latex-compatible format) and to use them in a tex file. The vectors and matrices of reasonable size can be handled with ease. The package can be modified or extended by writing custom Lua programs (Section 5).

The package supports fractions; numerators and denominators must be integers. A fraction can be specified with the Lua function: `lfrac`. This function has the syntax `lfrac(n,d,mode)`: n is an integer and d is a non-zero integer. The mode is optional. It can be `fracs` or `nofracs`. The default mode is `fracs`. If fractions are input, the package will display vectors and matrices in fraction mode wherever possible. The package does not attempt to convert floats into fractions. If fractions are expected, then the input should contain fractions. If fractions are input and answers are expected in numbers, the mode can be specified as `nofracs`.

The Lua function `lcomplex` defines the complex numbers. It has the syntax `lcomplex(x,y)`, where x is a real part, and y is an imaginary part. x and y can also be fractions (numerators and denominators should be integers). The package has a command `\imUnit` which provides typesetting for the imaginary unit. Its default value is `\mathrm{i}`. It can be redefined. For example, one can redefine it as `\renewcommand{\imUnit}{j}`.

2 Installation and License

The installation of the `lualinalg` package is similar to the plain latex package, where the `.sty` file is in the LaTeX directory of the texmf tree. The package can be included with `\usepackage{lualinalg}` command in the preamble of the LaTeX document.

The `lualinalg` package is released under the LaTeX Project Public License v1.3c or later. The complete license text is available at http://www.latex-project.org/lppl.txt. It is developed in Lua. Lua is available as a certified open-source software. Its license is simple and liberal, which is compatible with GPL. The package makes use of `complex.lua` file which is available on https://github.com/davidm/lua-matrix/blob/master/lua/complex.lua. It is available under the same licensing as that of Lua. The package also loads the luamaths package, which is available under the LaTeX Project Public License v1.3c or later. This package is loaded to use the standard mathematical functions and for computations on real numbers while performing operations on vectors and matrices.

*Email id: mathsbeauty@gmail.com

3 Defining vectors and performing operations on vectors

3.1 Defining Vectors

Vectors are defined with the `\vectornew` command.

```
\vectorNew{vector name}{coordinates}
```

This command has two compulsory arguments: `vector name` and `coordinates`. Coordinates of vectors are enclosed in curly braces. A comma separates coordinates. The following are a few valid ways of defining vectors.

```
\vectorNew{v1}{{1,2,3,4,5,6}}
\vectorNew{v2}{{3,6,lcomplex(6,6)}}
```

The standard vector of dimension n with i^{th} coordinate 1 can be produced by using the following command.

```
\vectorNew{e}{n,'e',i}
```

For example, the following commands

```
\vectorNew{e_1}{3,'e',1}
\(e_1=\left(\vectorPrint{e}\right)\)
```

output to $e_1 = (1, 0, 0)$.

3.2 Commands for operations on vectors

Table 1 lists commands for operations on vectors.

Command Format	Description
`\vectorPrint[truncate]{vector}`	Prints vector. Accepts one *optional* argument: `truncate`. It specifies the number of digits up to which vector coordinates must be truncated. The value of `truncate` may be 0,1,2,...
`\vectorGetCoordinate{vector}{i}`	Gives the ith coordinate of vector.
`\vectorSetCoordinate {vector}{i}{val}`	Sets the ith coordinate of vector as `val`.
`\vectorCopy{v}{w}`	Defines a new vector v obtained by copying coordinates of vector w.

`\vectorAdd{vector}{v1}{v2}`	Defines a new vector as the addition of vectors $v1$ and $v2$. Both vectors $v1$ and $v2$ should be of the same dimension. The addition is done coordinate-wise.
`\vectorSub{vector}{v1}{v2}`	Defines a new vector as the subtraction of vectors $v1$ and $v2$. Both vectors $v1$ and $v2$ should be of the same dimension. The subtraction is done coordinate-wise.
`\vectorMulNum{vector}{v}{num}`	Defines a new vector obtained by multiplying each coordinate of a vector by number `num`. It can be a real or complex number (scalar).
`\vectorDot{v}{w}`	Gives the dot product of two vectors: v and w. If $v = (v_1, \ldots, v_n)$ and $w = (w_1, \ldots, w_n)$ are defined over the field of real numbers, then it is evaluated as $v_1 \cdot w_1 + \cdots + v_n \cdot w_n$. If they are defined over the field of complex numbers, then it is evaluated as $v_1 \cdot \bar{w_1} + \cdots + v_n \cdot \bar{w_n}$. $\bar{w_i}$ denotes the complex conjugate of complex number w_i.
`\vectorCross{vector}{v}{w}`	Defines a new vector obtained by taking the cross product of vectors v and w of dimension 3. If $v = (v_1, v_2, v_3)$ and $w = (w_1, w_2, w_3)$, then the cross product of these two vectors is the vector $(v_2w_3 - v_3w_2, v_3w_1 - v_1w_3, v_1w_2 - v_2w_1)$.
`\vectorSumNorm{v}`	Calculates the sum norm of a vector v. If $v = (v_1, \ldots, v_n)$ then it is given by $\lvert v_1\rvert + \cdots + \lvert v_n\rvert$.
`\vectorEuclidNorm{v}`	Calculates the Euclidean norm of a vector v. If $v = (v_1, \ldots, v_n)$ then it is given by $\sqrt{\lvert v_1\rvert^2 + \cdots + \lvert v_n\rvert^2}$.
`\vectorpNorm{v}`	Calculates the p ($p > 1$) norm of a vector v. If $v = (v_1, \ldots, v_n)$ then it is given by $\sqrt[p]{\lvert v_1\rvert^p + \cdots + \lvert v_n\rvert^p}$.
`\vectorSupNorm{v}`	Calculates the sup norm of a vector v. If $v = (v_1, \ldots, v_n)$ then it is given by $\max\{\lvert v_1\rvert, \ldots, \lvert v_n\rvert\}$.
`\vectorCreateRandom{v}{n}{a}{b}`	Creates a new vector v of dimension n with coordinates as random numbers from the interval $[a, b]$.

`\vectorOp{vector}{expression}`	Defines a new vector obtained by evaluating an expression. The expression supports all standard operations such as $+, -, *$.
`\vectorGetAngle{v}{w}`	Gives the angle between two vectors v and w in radians. If v and w are defined over the field of real numbers, then it is evaluated as $\cos^{-1}\left(\frac{v \cdot w}{\lvert v\rvert\lvert w\rvert}\right)$. If they are defined over the field of complex numbers, then it is evaluated as $\cos^{-1}\left(\frac{\Re(v \cdot w)}{\lvert v\rvert\lvert w\rvert}\right)$. Here $v \cdot w$ denotes the dot product of vectors v and w, $\Re(v \cdot w)$ denotes real part of the dot product $v \cdot w$, and $\lvert v\rvert$ and $\lvert w\rvert$ denote Euclidean norms of vectors v and w respectively.
`\vectorParse{vector}`	Parses the coordinates of a vector defined over the field of real numbers. The command helps to plot vectors with different packages.
`\vectorGramSchmidt[brckt, truncate]{list of vectors}`	Performs Gram Schmidt orthogonalisation process on a list of vectors. Accepts two *optional* arguments: `brckt` and `truncate`. The `brckt` is type of parenthesis to be used for displaying vectors. It can be 'round', 'square' or 'curly'. The truncate is number of digits up to which vector coordinates are to be truncated. The value of `truncate` can be 0,1,2,...
`\vectorGramSchmidtSteps[brckt, truncate]{list of vectors}`	Performs Gram Schmidt orthogonalisation process on a list of vectors in a step-by-step manner. Accepts two *optional* arguments: `brckt` and `truncate`. The `brckt` is type of parenthesis to be used for displaying vectors. It can be 'round', 'square' or 'curly'. The truncate is number of digits up to which vector coordinates are to be truncated. The value of `truncate` can be 0,1,2,...

Table 1: Commands for operations on vectors

3.3 Illustrations of commands for operations on vectors

The following commands define vectors v, w, x, and y.

```
\vectorNew{v}{{1,2,lcomplex(3,3)}}
\vectorNew{w}{{3,6,lcomplex(6,6)}}
\vectorNew{x}{{1.12345678,6,lcomplex(6,6)}}
\vectorNew{y}{{1,2,3}}
```

Table 2 illustrates various operations on vectors v, w, x and y.

Commands	Output Produced
`\(v=\left(\vectorPrint{v}\right)\)` `\(w=\left(\vectorPrint{w}\right)\)`	$v = (1, 2, 3 + 3\mathrm{i})$ $w = (3, 6, 6 + 6\mathrm{i})$

`\(x=\left(\vectorPrint [truncate=3]{x}\right)\)`	$x = (1.123, 6, 6 + 6\mathrm{i})$
third coordinate of vector `\(v = \vectorGetCoordinate{v}{3}\)`	third coordinate of vector $v = 3 + 3\mathrm{i}$
`\(\vectorCopy{z}{w}\)` `\(z = \left(\vectorPrint{z}\right)\)`	$z = (3, 6, 6 + 6\mathrm{i})$
new third coordinate of vector `\(z = \vectorSetCoordinate{z}{3}{9.3}\)` `\(z=\left(\vectorPrint{z}\right)\)`	new third coordinate of vector $z = 9.3$ $z = (3, 6, 9.3)$
`\vectorAdd{v1}{v}{w}` `\(v1 = v+w =\left(\vectorPrint{v1}\right)\)`	$v1 = v + w = (4, 8, 9 + 9\mathrm{i})$
`\vectorSub{v2}{v}{w}` `\(v2 = v-w =\left(\vectorPrint{v2}\right)\)`	$v2 = v - w = (-2, -4, -3 - 3\mathrm{i})$
`\vectorMulNum{v3}{v}{complex('3+i')}` `\(v3 = 3v =\left(\vectorPrint{v3}\right)\)`	$v3 = 3v = (3 + \mathrm{i}, 6 + 2\mathrm{i}, 6 + 12\mathrm{i})$
`\(v \cdot w =\vectorDot{v}{w}\)`	$v \cdot w = 51$
`\vectorCross{v4}{v}{w}` `\(v \times w =\left(\vectorPrint{v4}\right)\)`	$v \times w = (-6 - 6\mathrm{i}, 3 + 3\mathrm{i}, 0)$
Sum norm of a vector `\(v = \vectorSumNorm{v}\)`	Sum norm of a vector $v = 7.2426406871193$
Euclidean norm of a vector `\(v = \vectorEuclidNorm{v}\)`	Euclidean norm of a vector $v = 4.7958315233127$
p norm of a vector `\(v = \vectorpNorm{v}{3}\)`	p norm of a vector $v = 4.4031577258332$
Sup norm of a vector `\(v = \vectorSupNorm{v}\)`	Sup norm of a vector $v = 4.2426406871193$
`\vectorCreateRandom{v5}{3}{9}{90}` `\(v5 =\left(\vectorPrint{v5}\right)\)`	$v5 = (27.740082, 59.816315, 79.907013)$
`\vectorOp{v6}{v+w-2*v}` `\(v6 =\left(\vectorPrint{v6}\right)\)`	$v6 = (2, 4, 3 + 3\mathrm{i})$
angle between vector `\(v\)` and `\(w\)` is `\( \vectorGetAngle{v}{w}\).`	angle between vector v and w is 0.18391428733893.
`\vectorParse{y}`	(1,2,3)

Table 2: Illustration of commands for operations on vectors

The package has commands for performing Gram Schmidt Orthogonalisation process. It can also produce the computations in a step-by step manner.

Listing 1: Gram Schmidt Orthogonalisation process in the lualinalgpackage

```
\vectorNew{v1}{{1,2,3}}
```

```
\vectorNew{v2}{{4,5,6}}
\vectorNew{v3}{{7,8,90}}
\[v1=\left(\vectorPrint{v1}\right)\]
\[v2=\left(\vectorPrint{v2}\right)\]
\[v3=\left(\vectorPrint{v3}\right)\]
Gram Schmidt on \(v1,v2,v3\):
    \vectorGramSchmidt[brckt=round,truncate=3]{{'v1','v2','v3'}}
\vectorGramSchmidtSteps[brckt=round,truncate=3]{{'v1','v2','v3'}}
```

Listing 1 outputs the following.

$$v1 = (1, 2, 3)$$

$$v2 = (4, 5, 6)$$

$$v3 = (7, 8, 90)$$

Gram Schmidt on $v1, v2, v3$: $(0.267, 0.535, 0.802)\,, (0.873, 0.218, -0.436)\,, (0.408, -0.816, 0.408)$
Take given vectors as $v_1, \ldots, v_3$ in order.
Step 1:

$$u_1 = v_1 = (1, 2, 3)$$

$$e_1 = \frac{u_1}{||u_1||} = (0.267, 0.535, 0.802)$$

Step 2:

$$u_2 = v_2 - \sum_{j=1}^{1} proj_{u_j}(v_2) = (1.714, 0.429, -0.857)$$

$$e_2 = \frac{u_2}{||u_2||} = (0.873, 0.218, -0.436)$$

Step 3:

$$u_3 = v_3 - \sum_{j=1}^{2} proj_{u_j}(v_3) = (13.5, -27, 13.5)$$

$$e_3 = \frac{u_3}{||u_3||} = (0.408, -0.816, 0.408)$$

In addition to `\mathRound`, the command `complexRound` is also available. It has the following syntax.

```
\complexRound{complex number}{number of decimal places}
```

This command has two compulsory arguments. The complex number and number of decimal places to which number should be rounded off. For example, `\complexRound{lcomplex(3.3333666, 6.777666)}{3}` outputs to $3.333 + 6.778$i. This command can be nested with other commands in the package.

3.4 Plotting vectors

The `lualinalg` package can be used with other packages that have facility to plot vectors defined over the field of real numbers in 2 or 3 dimensions. Listing 2 illustrates plotting of vectors in 2-D plane by using `lualinalg` and `tikz` package.

Listing 2: Plotting vectors in 2-dimensions with the lualinalg and tikz packages

```
\tdplotsetmaincoords{0}{0}
```

```
\begin{tikzpicture}[scale=1,
    tdplot_main_coords,
    axis/.style={->,blue,thick},
    vector/.style={-stealth,red,very thick},
    vector guide/.style={dashed,red,thick}]
\vectorNew{o}{{0,0}}
\vectorNew{e1}{{4,0}}
\vectorNew{e2}{{0,4}}
\vectorNew{f}{{2,1}}
\vectorNew{g}{{1,2}}
% Axes
\draw [axis] \vectorParse{o}-- \vectorParse{e1} node [below left] {$x$};
\draw [axis] \vectorParse{o}-- \vectorParse{e2} node [right] {$y$};
% Plotting Vectors
\draw [vector] \vectorParse{o} --\vectorParse{f};
\draw [vector] \vectorParse{o} --\vectorParse{g};
\vectorOp{h}{f+g}
\draw [vector] \vectorParse{o} --\vectorParse{h};
\draw [vector,dashed,black] \vectorParse{f} --\vectorParse{h};
\draw [vector,dashed,black] \vectorParse{g} --\vectorParse{h};
% Labels
 \node [below right] at \vectorParse{f} {$f$};
 \node [above left] at \vectorParse{g} {$g$};
\node [above left] at \vectorParse{h} {$f+g$};
 \draw[vector guide, black] \vectorParse{h} -- (\vectorGetCoordinate{h}{1},0) node
     [below] {$x=\vectorGetCoordinate{h}{1}$};
  \draw[vector guide, black] \vectorParse{h} -- (0,\vectorGetCoordinate{h}{2}) node
      [left] {$y=\vectorGetCoordinate{h}{2}$};
\end{tikzpicture}
```

Listing 2 produces figure 1.

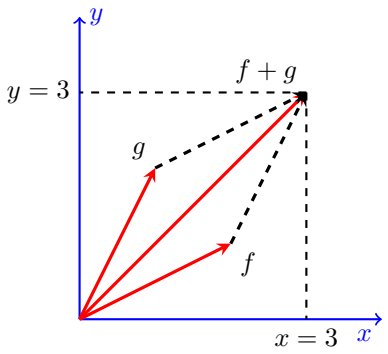

Figure 1: Plotting 2-D Vectors with lualinalg and tikz packages

Listing 3 illustrates plotting of vectors in 3-D plane by using `lualinalg` and `tikz` package.

Listing 3: Plotting vectors in 3-dimensions with the lualinalg and tikz packages

```
\documentclass{article}
\usepackage{tikz,tikz-3dplot,lualinalg}
\begin{document}
\tdplotsetmaincoords{60}{120}
```

```
\begin{tikzpicture}[scale=1,
    tdplot_main_coords,
    axis/.style={->,blue,thick},
    vector/.style={-stealth,red,very thick},
    vector guide/.style={dashed,red,thick}]
\vectorNew{o}{{0,0,0}}
\vectorNew{e1}{{3,0,0}}
\vectorNew{e2}{{0,5,0}}
\vectorNew{e3}{{0,0,4}}
\vectorNew{f}{{2,2,0}}
\vectorNew{g}{{-1,2,2}}
% Axes
\draw [axis] \vectorParse{o}-- \vectorParse{e1} node [below left] {$x$};
\draw [axis] \vectorParse{o}-- \vectorParse{e2} node [right] {$y$};
\draw [axis] \vectorParse{o}-- \vectorParse{e3} node [above] {$z$};
% Plotting Vectors
\draw [vector] \vectorParse{o} --\vectorParse{f};
\draw [vector] \vectorParse{o} --\vectorParse{g};
\vectorOp{h}{f+g}
\draw [vector] \vectorParse{o} --\vectorParse{h};
% Labels
 \node [below right] at \vectorParse{f} {$f$};
 \node [above left] at \vectorParse{g} {$g$};
\node [right] at \vectorParse{h} {$f+g$};
 \draw[vector guide, black] \vectorParse{h} -- (\vectorGetCoordinate{h}{1},0,0) node
     [left] {$x=\vectorGetCoordinate{h}{1}$};
  \draw[vector guide, black] \vectorParse{h} -- (0,\vectorGetCoordinate{h}{2},0) node
      [below] {$y=\vectorGetCoordinate{h}{2}$};
   \draw[vector guide, black] \vectorParse{h} -- (0,0,\vectorGetCoordinate{h}{3}) node
       [left] {$z=\vectorGetCoordinate{h}{3}$};
\end{tikzpicture}
\end{document}
```

Listing 3 produces figure 2.

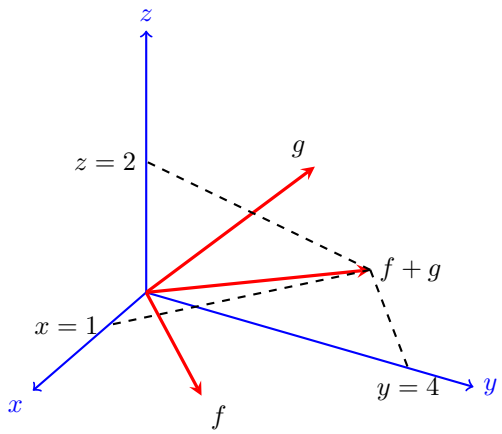

Figure 2: Plotting 3-D Vectors with lualinalg and tikz packages

4 Defining matrices and operations on matrices

Matrices are defined with the `\matrixNew` command.

```
\matrixNew{matrix name}{row entries}
```

This command has two compulsory arguments: `matrix name` and `row entries`. Each row of the matrix is enclosed in curly brackets. A comma separates numbers in rows. Rows are also separated by a comma. The whole matrix is then enclosed in curly brackets. The following are a few valid ways of defining matrices.

```
\def\n{{{1,2,3},{4,5,6},{7,8,lcomplex(9,3)}}}
\def\s{{{1,2,3},{4,5,6},{7,8,10}}}
\matrixNew{m}{\n}
\matrixNew{n}{\s}
% an alternative way
\matrixNew{m}{{{1,2,3},{4,5,6},{7,8,lcomplex(9,3)}}}
\matrixNew{n}{{{1,2,3},{4,5,6},{7,8,10}}}
```

The identity matrix can be defined as well by using the `\matrixNew` command. For example, the following commands

```
\matrixNew{mtx}{3,'I'}
I = \(\matrixPrint{mtx}\)
```

output to $\mathrm{I} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

4.1 Commands for operations on matrices

Table 3 lists all commands for operations on matrices in the `lualinalg` package.

Command Format	Description
Printing Matrices	
`\matrixPrint[type,truncate]{mtx}`	Prints matrix. Accepts two *optional* arguments: `type` and `truncate`. The `type` may be one of the values `pmatrix,bmatrix, vmatrix,Vmatrix`. The default type is `bmatrix`. The `truncate` specifies the number of digits up to which matrix entries are to be truncated. The value of `truncate` may be 0,1,2,....
Some parameters of defined matrices	
`\matrixNumRows{matrix}`	Gives the number of rows in a matrix.
`\matrixNumCols{matrix}`	Gives the number of columns in a matrix.

`\matrixGetElement{matrix}{i}{j}`	Gives an entry of matrix in the ith row and the jth column.
Algebraic operations on matrices	
`\matrixAdd{matrix}{m1}{m2}`	Defines a new matrix as the addition of matrices m1 and m2. The second matrix may have more rows and\or columns.
`\matrixSub{matrix}{m1}{m2}`	Defines a new matrix as the subtraction of matrices m1 and m2. The second matrix may have more rows and\or columns.
`\matrixMulNum{matrix}{number}{m1}`	Defines a new matrix obtained by multiplying each entry of matrix m1 by s real or complex number.
`\matrixMul{matrix}{m1}{m2}`	Defines a new matrix obtained by multiplying matrix m1 by matrix m2. The number of rows in matrix m2 must equal the number of columns in matrix m1.
`\matrixPow{matrix}{m1}{power}`	Defines a new matrix obtained by taking the ith power of matrix m1 (multiplying matrix m1 i times with itself).
`\matrixInvert{matrix}{matrix1}`	Defines a new matrix obtained by taking the inverse of matrix1. It throws an error if matrix is not invertible.
`\matrixTrace{matrix}`	Gives the trace (sum of diagonal entries) of a square matrix. It throws an error if the matrix is not square.
`\matrixConjugate{matrix}{m1}`	Defines a new matrix obtained by taking the complex conjugate of each entry of matrix m1.
`\matrixConjugateT{matrix}{m1}`	Defines a new matrix obtained by taking the transpose of matrix m1 and then the complex conjugate of each matrix entry.

`\matrixNormOne{matrix}`	Calculates the norm1 of a matrix. For matrix A of size $m \times n$, it is given by $$\|A\|_1 = \max_{1 \leqslant j \leqslant n} \sum_{i=1}^{m} \|a_{ij}\|$$
`\matrixNormInfty{matrix}`	Calculates the infinity norm of a matrix. For matrix A of size $m \times n$, it is given by $$\|A\|_\infty = \max_{1 \leqslant i \leqslant m} \sum_{j=1}^{n} \|a_{ij}\|$$
`\matrixNormMax{matrix}`	Calculates the max norm of a matrix. For matrix A of size $m \times n$, it is given by $$\|A\|_{\max} = \max_{i,j} \|a_{ij}\|$$
`\matrixNormF{matrix}`	Calculates the Frobenius norm of a matrix. For matrix A of size $m \times n$, it is given by $$\|A\|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} \|a_{ij}\|^2}$$
`\matrixRank{matrix}`	Gives the rank of matrix m. It also supports matrices of complex numbers.
`\matrixDet{matrix}`	Gives the determinant of matrix m. It also supports matrices of complex numbers.
`\matrixTranspose{matrix}{m1}`	Defines a new matrix obtained by taking the transpose of matrix m1.
`\matrixSetElement{matrix}{i}{j}{val}`	Set entry of a matrix in the ith row and jth column as `val`.
`\matrixSubmatrix{sm}{m}{i}{j}{k}{l}`	Defines a new matrix sm obtained by taking a submatrix of matrix m. Here i, j denotes the start row and start column, and k, l denotes the end row and end column for obtaining the submatrix.

`\matrixConcatH{matrix}{m1}{m2}`	Defines a new matrix obtained by augmenting matrix m1 with matrix m2 horizontally.
`\matrixConcatV{matrix}{m1}{m2}`	Defines a new matrix obtained by augmenting matrix m1 with matrix m2 vertically.
`\matrixOp{matrix}{expression}`	Defines a new matrix obtained by evaluating an expression. The expression supports all standard operations such as $+, *, \hat{}$.
`\matrixCopy{matrix}{matrix1}`	Defines a new matrix obtained by copying values from matrix1.
`\matrixCreateRandom {m}{i}{j}{k}{l}`	Creates a new matrix m with random numbers. Here i, j denotes the number of rows and columns, and k, l denotes the start and end integers between which random numbers are generated.
Elementary row operations on matrices	
`\matrixSwapRows{mtx}{m1}{i}{j}`	Defines a new matrix mtx obtained by swapping the ith and jth rows of matrix m1.
`\matrixMulRow{matrix}{m}{i}{no}`	Defines a new matrix obtained by multiplying the ith row of matrix1 by a real or complex number.
`\matrixMulAddRow{mtx}{m}{i}{no}{j}`	Defines a new matrix mtx obtained by multiplying the ith row of matrix1 by a real or complex number and adding it to the jth row.
Elementary column operations on matrices	
`\matrixSwapCols{mtx}{m}{i}{j}`	Defines a new matrix mtx obtained by swapping the ith and jth columns of matrix m.
`\matrixMulCol{matrix}{m}{i}{no}`	Defines a new matrix obtained by multiplying the ith column of matrix1 by a real or complex number.

`\matrixMulAddCol{mtx}{m}{i}{no}{j}`	Defines a new matrix mtx obtained by multiplying the ith column of matrix1 by a real or complex number and adding it to the jth column.
Reduced row echelon form of matrix	
`\matrixRREF{matrix}{matrix1}`	Defines a new matrix obtained by taking the reduced row echelon form of matrix1. It supports matrices of complex numbers as well.
`\matrixRREFSteps[type,truncate]` `{matrix}`	Obtains reduced row echelon form of matrix in a step-by-step manner. The command has two optional parameters `type` and `truncate`. It supports matrices with complex numbers as well. `type` may be one of the values `pmatrix,bmatrix, vmatrix,Vmatrix`. The default type is `bmatrix`. `truncate` specifies number of digits up to which matrix entries are to be truncated. `truncate` may be 0,1,2,….
Gauss-Jordan Elimination	
`\matrixGaussJordan{matrix}` `{m1}{m2}`	Defines new matrix obtained by performing Gauss-Jordan elimination on augmented matrix $m1\|m2$.
`\matrixGaussJordanSteps[type,truncate]` `{matrix}{m1}{m2}`	Defines new matrix obtained by performing Gauss-Jordan elimination on augmented matrix $m1\|m2$ in a step-by-step manner. The command has two optional parameters `type` and `truncate`. `type` may be one of the values `pmatrix, bmatrix, vmatrix, Vmatrix`. The default type is `bmatrix`. `truncate` specifies number of digits up to which matrix entries are to be truncated. `truncate` may be 0,1,2,….

Table 3: Commands for operations on matrices

4.2 Illustrations of matrix operations

The following commands define matrices m, n, and r.

```
\def\r{{{1,2,3},{4,5,6},{7,8,lcomplex(9,3)}}}
\def\s{{{1,2,3},{4,5,6},{7,8,10}}}
\def\t{{{1,2,3},{4,5,6},{7,8,9}}}
\def\u{{{1},{2},{3}}}
\def\z{{{lfrac(1,2),lcomplex(2,3),3},{4,5,6},{7,8,9}}}

\matrixNew{m}{\r}
\matrixNew{n}{\s}
```

```
\matrixNew{p}{\t}
\matrixNew{q}{\u}
\matrixNew{r}{\z}
```

Table 4 illustrates various operations on matrices m, n, p, and q.

Commands	Output Produced
Printing matrices	
`\(m=\matrixPrint{m}\)`	$m = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9+3\mathrm{i} \end{bmatrix}$
`\(m=\matrixPrint[type=pmatrix]{m}\)`	$m = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9+3\mathrm{i} \end{pmatrix}$
Some parameters of defined matrices	
`No. or rows in matrix \(m = \matrixNumRows{m}\)`	No. or rows in matrix $m = 3$
`No. or columns in matrix \(m = \matrixNumCols{m}\)`	No. or columns in matrix $m = 3$
`Element of matrix \(m\) at \((3,3) = \matrixGetElement{m}{3}{3}\)`	Element of matrix m at $(3, 3) = 9 + 3\mathrm{i}$
Algebraic operations on matrices	
`\matrixAdd{m1}{m}{p}` `\(m1 = \matrixPrint{m1}\)`	$m1 = \begin{bmatrix} 2 & 4 & 6 \\ 8 & 10 & 12 \\ 14 & 16 & 18+3\mathrm{i} \end{bmatrix}$
`\matrixSub{m2}{m}{p}` `\(m2 = \matrixPrint{m2}\)`	$m2 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 3\mathrm{i} \end{bmatrix}$
`\matrixMulNum{m3}{3}{m}` `\(m3 = \matrixPrint{m3}\)`	$m3 = \begin{bmatrix} 3 & 6 & 9 \\ 12 & 15 & 18 \\ 21 & 24 & 27+9\mathrm{i} \end{bmatrix}$
`\matrixMul{m4}{m}{p}` `\(m4 = \matrixPrint{m4}\)`	$m4 = \begin{bmatrix} 30 & 36 & 42 \\ 66 & 81 & 96 \\ 102+21\mathrm{i} & 126+24\mathrm{i} & 150+27\mathrm{i} \end{bmatrix}$
`\matrixPow{m5}{m}{2}` `\(m5 = \matrixPrint{m5}\)`	$m5 = \begin{bmatrix} 30 & 36 & 42+9\mathrm{i} \\ 66 & 81 & 96+18\mathrm{i} \\ 102+21\mathrm{i} & 126+24\mathrm{i} & 141+54\mathrm{i} \end{bmatrix}$
`\matrixInvert{m6}{m}` `\(m6 = \matrixPrint[truncate=2]{m6}\)`	$m6 = \begin{bmatrix} -1.67-0.33\mathrm{i} & 0.67+0.67\mathrm{i} & -0.33\mathrm{i} \\ 1.33+0.67\mathrm{i} & -0.33-1.33\mathrm{i} & 0.67\mathrm{i} \\ -0.33\mathrm{i} & 0.67\mathrm{i} & -0.33\mathrm{i} \end{bmatrix}$
`Rank of matrix \(m =\matrixRank{m}\)`	Rank of matrix $m = 3$
`Determinant of matrix \(m =\matrixDet{m}\)`	Determinant of matrix $m = -9\mathrm{i}$
`\matrixTranspose{m7}{m}` `\(m7 = \matrixPrint{m7}\)`	$m7 = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9+3\mathrm{i} \end{bmatrix}$

`\matrixSetElement{n}{3}{3}{300}` `\(n= \matrixPrint{n}\)`	$n = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 300 \end{bmatrix}$
`\matrixSubmatrix{m8}{m}{1}{2}{2}{3}` `\(m8 = \matrixPrint{m8}\)`	$m8 = \begin{bmatrix} 2 & 3 \\ 5 & 6 \end{bmatrix}$
`\matrixConcatH{m9}{m}{q}` `\(m9= \matrixPrint{m9}\)`	$m9 = \begin{bmatrix} 1 & 2 & 3 & 1 \\ 4 & 5 & 6 & 2 \\ 7 & 8 & 9+3\mathrm{i} & 3 \end{bmatrix}$
`\matrixConcatV{m10}{m}{n}` `\(m10= \matrixPrint{m10}\)`	$m10 = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9+3\mathrm{i} \\ 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 300 \end{bmatrix}$
`\matrixOp{m11}{m*m+2*m}` `\(\matrixPrint[truncate=4]{m11}\)`	$\begin{bmatrix} 32 & 40 & 48+9\mathrm{i} \\ 74 & 91 & 108+18\mathrm{i} \\ 116+21\mathrm{i} & 142+24\mathrm{i} & 159+60\mathrm{i} \end{bmatrix}$
`\matrixCopy{m12}{m}` `\(m12 = \matrixPrint{m12}\)`	$m12 = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9+3\mathrm{i} \end{bmatrix}$
`trace of matrix \( m = \matrixTrace{m}\)`	trace of matrix $m = 15+3\mathrm{i}$
`\matrixConjugate{mc}{m}` `\(mc = \matrixPrint{mc}\)`	$mc = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9-3\mathrm{i} \end{bmatrix}$
`\matrixConjugateT{mct}{m}` `\(mct = \matrixPrint{mct}\)`	$mct = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9-3\mathrm{i} \end{bmatrix}$
`\(\matrixNormOne{m}\)`	18.486832980505
`\(\matrixNormInfty{m}\)`	24.486832980505
`\(\matrixNormMax{m}\)`	9.4868329805051
`\(\matrixNormF{m}\)`	17.146428199482
Elementary row operations on matrices	
`\matrixSwapRows{m13}{m}{2}{3}` `\(m13 = \matrixPrint{m13}\)`	$m13 = \begin{bmatrix} 1 & 2 & 3 \\ 7 & 8 & 9+3\mathrm{i} \\ 4 & 5 & 6 \end{bmatrix}$
`\matrixMulRow{m14}{m}{3}{300}` `\(m14 = \matrixPrint{m14}\)`	$m14 = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 2100 & 2400 & 2700+900\mathrm{i} \end{bmatrix}$
`\matrixMulAddRow{m15}{m}{2}{10}{3}` `\(m15 = \matrixPrint{m15}\)`	$m15 = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 47 & 58 & 69+3\mathrm{i} \end{bmatrix}$
Elementary column operations on matrices	

`\matrixSwapCols{m16}{m}{2}{3}` `\(m16 = \matrixPrint{m16}\)`	$m16 = \begin{bmatrix} 1 & 3 & 2 \\ 4 & 6 & 5 \\ 7 & 9+3\mathrm{i} & 8 \end{bmatrix}$
`\matrixMulCol{m17}{m}{3}{300}` `\(m17 = \matrixPrint{m17}\)`	$m17 = \begin{bmatrix} 1 & 2 & 900 \\ 4 & 5 & 1800 \\ 7 & 8 & 2700+900\mathrm{i} \end{bmatrix}$
`\matrixMulAddCol{m18}{m}{2}{10}{3}` `\(m18 = \matrixPrint{m18}\)`	$m18 = \begin{bmatrix} 1 & 2 & 23 \\ 4 & 5 & 56 \\ 7 & 8 & 89+3\mathrm{i} \end{bmatrix}$
Reduced row echelon form of a matrix	
`\matrixRREF{m19}{p}` `\(m19 = \matrixPrint{m19}\)`	$m19 = \begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & 2 \\ 0 & 0 & 0 \end{bmatrix}$
`\matrixRREF{m20}{m}` `\(m20 = \matrixPrint{m20}\)`	$m20 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

Table 4: Illustration of commands for operations on matrices

The package has command `\matrixRREFSteps` to produce step-by-step computation of reduced row echelon form of a matrix. The command `\matrixRREFSteps{r}` outputs the following.

Step 1: Divide row 1 by $\frac{1}{2}$.

$$\begin{bmatrix} 1 & 4+6\mathrm{i} & 6 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Step 2: Multiply row 1 by 4 and subtract it from row 2.

$$\begin{bmatrix} 1 & 4+6\mathrm{i} & 6 \\ 0 & -11-24\mathrm{i} & -18 \\ 7 & 8 & 9 \end{bmatrix}$$

Step 3: Multiply row 1 by 7 and subtract it from row 3.

$$\begin{bmatrix} 1 & 4+6\mathrm{i} & 6 \\ 0 & -11-24\mathrm{i} & -18 \\ 0 & -20-42\mathrm{i} & -33 \end{bmatrix}$$

Step 4: Divide row 2 by $-11-24\mathrm{i}$.

$$\begin{bmatrix} 1 & 4+6\mathrm{i} & 6 \\ 0 & 1 & \frac{198}{697}+\frac{-432}{697}\mathrm{i} \\ 0 & -20-42\mathrm{i} & -33 \end{bmatrix}$$

Step 5: Multiply row 2 by $4 + 6\mathrm{i}$ and subtract it from row 1.

$$\begin{bmatrix} 1 & 0 & \frac{798}{697} + \frac{540}{697}\mathrm{i} \\ 0 & 1 & \frac{198}{697} + \frac{-432}{697}\mathrm{i} \\ 0 & -20 - 42\mathrm{i} & -33 \end{bmatrix}$$

Step 6: Multiply row 2 by $-20 - 42\mathrm{i}$ and subtract it from row 3.

$$\begin{bmatrix} 1 & 0 & \frac{798}{697} + \frac{540}{697}\mathrm{i} \\ 0 & 1 & \frac{198}{697} + \frac{-432}{697}\mathrm{i} \\ 0 & 0 & \frac{-897}{697} + \frac{-324}{697}\mathrm{i} \end{bmatrix}$$

Step 7: Divide row 3 by $\frac{-897}{697} + \frac{-324}{697}\mathrm{i}$.

$$\begin{bmatrix} 1 & 0 & \frac{798}{697} + \frac{540}{697}\mathrm{i} \\ 0 & 1 & \frac{198}{697} + \frac{-432}{697}\mathrm{i} \\ 0 & 0 & 1 \end{bmatrix}$$

Step 8: Multiply row 3 by $\frac{798}{697} + \frac{540}{697}\mathrm{i}$ and subtract it from row 1.

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & \frac{198}{697} + \frac{-432}{697}\mathrm{i} \\ 0 & 0 & 1 \end{bmatrix}$$

Step 9: Multiply row 3 by $\frac{198}{697} + \frac{-432}{697}\mathrm{i}$ and subtract it from row 2.

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The command `\matrixGaussJordan` is used to obtain Gauss-Jordan elimination of an augmented matrix.

```
\def\a{{{lfrac(1,2),1,1},{2,-1,-1},{1,-1,1}}}
\def\b{{{3},{3},{9}}}
\matrixNew{S}{\a}
\matrixNew{T}{\b}
\matrixConcatH{W}{S}{T}
$$W = \matrixPrint{W}$$
\matrixGaussJordan{U}{S}{T}
$$U = \matrixPrint{U}$$
```

The above code produces the following output.

$$W = \begin{bmatrix} \frac{1}{2} & 1 & 1 & 3 \\ 2 & -1 & -1 & 3 \\ 1 & -1 & 1 & 9 \end{bmatrix}$$

$$U = \begin{bmatrix} 1 & 0 & 0 & \frac{12}{5} \\ 0 & 1 & 0 & \frac{-12}{5} \\ 0 & 0 & 1 & \frac{21}{5} \end{bmatrix}$$

The package also has a command `\matrixGaussJordanSteps` to produce step-by-step computation of Gauss-Jordan elimination of an augmented matrix. The command `\matrixGaussJordanSteps{S}{T}` produces the following output.

Step 1: Divide row 1 by $\frac{1}{2}$.

$$\begin{bmatrix} 1 & 2 & 2 & 6 \\ 2 & -1 & -1 & 3 \\ 1 & -1 & 1 & 9 \end{bmatrix}$$

Step 2: Multiply row 1 by 2 and subtract it from row 2.

$$\begin{bmatrix} 1 & 2 & 2 & 6 \\ 0 & -5 & -5 & -9 \\ 1 & -1 & 1 & 9 \end{bmatrix}$$

Step 3: Multiply row 1 by 1 and subtract it from row 3.

$$\begin{bmatrix} 1 & 2 & 2 & 6 \\ 0 & -5 & -5 & -9 \\ 0 & -3 & -1 & 3 \end{bmatrix}$$

Step 4: Divide row 2 by -5.

$$\begin{bmatrix} 1 & 2 & 2 & 6 \\ 0 & 1 & 1 & \frac{9}{5} \\ 0 & -3 & -1 & 3 \end{bmatrix}$$

Step 5: Multiply row 2 by 2 and subtract it from row 1.

$$\begin{bmatrix} 1 & 0 & 0 & \frac{12}{5} \\ 0 & 1 & 1 & \frac{9}{5} \\ 0 & -3 & -1 & 3 \end{bmatrix}$$

Step 6: Multiply row 2 by -3 and subtract it from row 3.

$$\begin{bmatrix} 1 & 0 & 0 & \frac{12}{5} \\ 0 & 1 & 1 & \frac{9}{5} \\ 0 & 0 & 2 & \frac{42}{5} \end{bmatrix}$$

Step 7: Divide row 3 by 2.

$$\begin{bmatrix} 1 & 0 & 0 & \frac{12}{5} \\ 0 & 1 & 1 & \frac{9}{5} \\ 0 & 0 & 1 & \frac{21}{5} \end{bmatrix}$$

Step 8: Multiply row 3 by 0 and subtract it from row 1.

$$\begin{bmatrix} 1 & 0 & 0 & \frac{12}{5} \\ 0 & 1 & 1 & \frac{9}{5} \\ 0 & 0 & 1 & \frac{21}{5} \end{bmatrix}$$

Step 9: Multiply row 3 by 1 and subtract it from row 2.

$$\begin{bmatrix} 1 & 0 & 0 & \frac{12}{5} \\ 0 & 1 & 0 & \frac{-12}{5} \\ 0 & 0 & 1 & \frac{21}{5} \end{bmatrix}$$

5 Customized usage

The commands available in the package can be used for performing further operations on vectors and matrices. The command `\vectorAdd` can be extended to add more than two vectors. The latex document (listing 4) provides some instances of such usage.

Listing 4: Customized usage of the lualinalg package

```
\documentclass{article}
\usepackage{lualinalg}
\begin{document}
\begin{luacode*}
function sumcoordinates(v1)
local sum = 0
for i = 1,#v1 do
    sum = sum + v1[i]
   end
return sum
end

function vector.addmulti(...)
  p=table.pack(...)
  s=vector(p[1])
  for i=2,#p do
s=vector.add(s,vector(p[i]))
end
  return s
end
\end{luacode*}
\vectorNew{v}{{1,2,lcomplex(3,3)}}
The sum of coordinates of vector
```

```
 \(v = \directlua{tex.sprint(tostring( sumcoordinates(vectors['v'])))}\).

\newcommand\vectorAddmulti[2]{%
   \directlua{%
      vectors['#1'] = vector.addmulti(#2)
   }%
}
\vectorNew{w}{{3,6,lcomplex(6,6)}}
\vectorNew{x}{{9,12,lcomplex(12,12)}}
\vectorAddmulti{y}{vectors['v'],vectors['w'],vectors['x']}
The sum of vectors \(v,w \text{ and } x =\left( \vectorPrint{y} \right)\).
\end{document}
```

The latex document (listing 4) outputs the following on compilation.

> The sum of coordinates of vector $v = 6 + 3\mathrm{i}$.
>
> The sum of vectors v, w and $x = (13, 20, 21 + 21\mathrm{i})$.

The command **\matrixAdd** can be extended to add more than two matrices. The latex document (listing 5) provides some instances of such usage.

Listing 5: Customized usage of the lualinalg package

```
\documentclass{article}
\usepackage{lualinalg}
\begin{document}
\begin{luacode}
function squareDiagEntries(m1)
 if #m1 ~= #m1[1] then error( "matrix not square") end
  local sum = 0
for i = 1,#m1 do
   for j = 1,#m1[1] do
    if i == j then sum = sum + (m1[i][j])^2 end
   end
  end
return complex.round(sum)
end

function matrix.addmulti(...)
  p=table.pack(...)
  s=matrix(p[1])
  for i=2,#p do
s=matrix.add(s,matrix(p[i]))
end
  return s
end
\end{luacode}

\def\r{{{1,2,3},{4,5,6},{7,8,lcomplex(9,3)}}}
\matrixNew{m}{\r}
The sum of squares of diagonal entries of matrix
 \(m = \directlua{tex.sprint(tostring(squareDiagEntries(matrices['m'])))}\).

\def\s{{{1,2,3},{4,5,lcomplex(6,6)}}}
```

```
\def\t{{{10,20,30},{40,50,lcomplex(60,60)}}}
\def\u{{{100,200,300},{400,500,lcomplex(600,600)}}}
\matrixNew{m1}{\s}
\matrixNew{m2}{\t}
\matrixNew{m3}{\u}
\newcommand\matrixAddmulti[2]{%
    \directlua{%
        matrices['#1'] = matrix.addmulti(#2)
    }%
}
\matrixAddmulti{m4}{matrices['m1'],matrices['m2'],matrices['m3']}
The sum of matrices \(m1,m2 \text{ and } m3 = \matrixPrint{m4}\).
\end{document}
```

The latex document (listing 5) outputs the following on compilation.

The sum of squares of diagonal entries of matrix $m = 98 + 54\text{i}$.

The sum of matrices $m1, m2$ and $m3 = \begin{bmatrix} 111 & 222 & 333 \\ 444 & 555 & 666 + 666\text{i} \end{bmatrix}$.

6 Known issues and limitations

- The package supports small and big numbers. They can be input in the usual scientific notation. The math library in Lua defines constants with the maximum `math.maxinteger` and the minimum `math.mininteger` values for an integer. The result wraps around when there is a computational operation on integers that would result in a value smaller than the `mininteger` or larger than the `maxinteger`. It means that the computed result is the only number between the `miniinteger` and `maxinteger`.
- The package currently supports only numerical computations. The table in a Lua is a data type that implements an associative array. This feature is used in packages to define and store vectors and matrices. This approach is close to object-oriented programming. It will allow easy conversion of algorithms in packages for symbolic computations. Future package updates will consider algorithm conversions to support symbolic calculations.
- The error handling mechanism in the tool is not robust. There are some custom errors included in the package. However the package mostly depends on error handling mechanism of Lua. The error handling can be strengthened in future updates of the package.