13.2_Sympy_and_Lineal_Algebra

December 2, 2014

Figure 1: BY-SA

Authors : *Sonia Estradé*
José M. Gómez
Ricardo Graciani
Manuel López
Xavier Luri
Josep Sabater

1 13.6 Symbolic linear algebra

Sympy implements operations with matrices and vectors, ie linear algebra. It provides a `Matrix` class equivalent to the *numpy*, but supporting symbolic items.

1.1 Creating Matrices

1.1.1 From a list

A `Matrix` object can be created from a list (or nested list):

```
In [1]: import sympy as sp
        sp.init_printing()

        matrix = sp.Matrix([[1, -1], [3, 4], [0, 2]])
        print( 'Matrix:', matrix )
        print( 'Matrix shape:', matrix.shape )
        matrix
```

```
Matrix: Matrix([[1, -1], [3, 4], [0, 2]])
Matrix shape: (3, 2)
```

Out[1]:

$$\begin{bmatrix} 1 & -1 \\ 3 & 4 \\ 0 & 2 \end{bmatrix}$$

When a `Matrix` is created from a simple list, it produces a column vector. Like in *numpy*, *sympy* matrices can be reshaped.

```
In [2]: import sympy as sp
        sp.init_printing()

        matrix = sp.Matrix([1,2,3])
        matrix
```

Out[2]:

$$\begin{bmatrix}1\\2\\3\end{bmatrix}$$

```
In [3]: matrix = sp.Matrix( range(9)).reshape(3,3)
        matrix
```

Out[3]:

$$\begin{bmatrix}0 & 1 & 2\\3 & 4 & 5\\6 & 7 & 8\end{bmatrix}$$

As mentioned above, the members of a `Matrix` can also be any *sympy* `Symbol`.

```
In [4]: import sympy as sp
        sp.init_printing()

        matrix = sp.Matrix( sp.var('x y z') )
        matrix
```

Out[4]:

$$\begin{bmatrix}x\\y\\z\end{bmatrix}$$

1.1.2 Using the functions zeros(), ones(), eye(), diag()

Like in *numpy*, they produce a matrix full of zeros or ones, the identity matrix or a diagonal matrix.

```
In [5]: import sympy as sp
        sp.init_printing()

        matrix0 = sp.zeros(2, 3)
        matrix1 = sp.ones(3, 2)
        matrixI = sp.eye(3)
        matrixD = sp.diag(1, 2, 3, 4, 5)

        matrix0, matrix1, matrixI, matrixD
```

Out[5]:

$$\left(\begin{bmatrix}0 & 0 & 0\\0 & 0 & 0\end{bmatrix}, \begin{bmatrix}1 & 1\\1 & 1\\1 & 1\end{bmatrix}, \begin{bmatrix}1 & 0 & 0\\0 & 1 & 0\\0 & 0 & 1\end{bmatrix}, \begin{bmatrix}1 & 0 & 0 & 0 & 0\\0 & 2 & 0 & 0 & 0\\0 & 0 & 3 & 0 & 0\\0 & 0 & 0 & 4 & 0\\0 & 0 & 0 & 0 & 5\end{bmatrix}\right)$$

`sp.diag()` also admits matrices as arguments, placing each of them diagonally, completing with zeros as needed:

```
In [6]: sp.diag( matrix, matrix0, matrix1, matrixI, matrixD )
```

Out[6]:

$$\begin{bmatrix} x & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ y & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ z & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 5 \end{bmatrix}$$

1.1.3 Using a function

The function should accept 2 arguments, row and column, and the matrix is filled with the values returned by the function.

```
In [7]: import sympy as sp
        sp.init_printing()

        def matrixFunction(row,col):
            # The output is the row squared with the column added
            return row**2 + col

        matrixF = sp.Matrix(4, 4, matrixFunction)
        matrixF
```

Out[7]:

$$\begin{bmatrix} 0 & 1 & 2 & 3 \\ 1 & 2 & 3 & 4 \\ 4 & 5 & 6 & 7 \\ 9 & 10 & 11 & 12 \end{bmatrix}$$

1.2 Accesing the elements of a Matrix

The elements of Matrix objects can be access with `[row, col]` or `[row range, col range]`, as used in vector slicing.

```
In [8]: import sympy as sp
        sp.init_printing()

        matrix = sp.Matrix([[1, -1, 0], [2, 3, 4], [0, 2, 7]])

        matrix
```

Out[8]:

$$\begin{bmatrix}1 & -1 & 0\\2 & 3 & 4\\0 & 2 & 7\end{bmatrix}$$

```
In [9]: print( 'Element 1,2:', matrix[1,2])
        matrix[0:2,0:3]
```

```
Element 1,2: 4
```

Out[9]:

$$\begin{bmatrix}1 & -1 & 0\\2 & 3 & 4\end{bmatrix}$$

Complete rows and columns can be accessed using the methods `row()` and `col()`:

```
In [10]: import sympy as sp
         sp.init_printing()

         matrix = sp.Matrix([[1, -1, 0], [2, 3, 4], [0, 2, 7]])

         matrix
```

Out[10]:

$$\begin{bmatrix}1 & -1 & 0\\2 & 3 & 4\\0 & 2 & 7\end{bmatrix}$$

```
In [11]: matrix.row(1)
```

Out[11]:

$$\begin{bmatrix}2 & 3 & 4\end{bmatrix}$$

```
In [12]: matrix.col(2)
```

Out[12]:

$$\begin{bmatrix}0\\4\\7\end{bmatrix}$$

Notice that they return a matrix object of the appropriate shape.
The transposed matrix it is also easily accesible using the **T** data member of the object:

```
In [13]: matrix, matrix.T
```

Out[13]:

$$\left(\begin{bmatrix}1 & -1 & 0\\2 & 3 & 4\\0 & 2 & 7\end{bmatrix}, \begin{bmatrix}1 & 2 & 0\\-1 & 3 & 2\\0 & 4 & 7\end{bmatrix}\right)$$

1.3 Basic operations

The basic operators +, -, *, /, ** are defined for Matrix objects as the usual matrix operations.

In [14]:
```
import sympy as sp
sp.init_printing()

# Definim les matrius
M = sp.Matrix([[1, -1, 0], [2, 3, 4], [0, 2, 7]])
N = sp.Matrix([[5, 6, 2], [8, 7, 7], [5, 1, 1]])

print( 'M, N' )
M,N
```

```
M, N
```

Out[14]:

$$\left(\begin{bmatrix}1 & -1 & 0\\2 & 3 & 4\\0 & 2 & 7\end{bmatrix}, \quad \begin{bmatrix}5 & 6 & 2\\8 & 7 & 7\\5 & 1 & 1\end{bmatrix}\right)$$

In [15]:
```
# Addition and substraction
print( 'M+N, M-N' )
M+N, M-N
```

```
M+N, M-N
```

Out[15]:

$$\left(\begin{bmatrix}6 & 5 & 2\\10 & 10 & 11\\5 & 3 & 8\end{bmatrix}, \quad \begin{bmatrix}-4 & -7 & -2\\-6 & -4 & -3\\-5 & 1 & 6\end{bmatrix}\right)$$

In [16]:
```
# Product
print( 'M*N' )
M*N
```

```
M*N
```

Out[16]:

$$\begin{bmatrix}-3 & -1 & -5\\54 & 37 & 29\\51 & 21 & 21\end{bmatrix}$$

In [17]:
```
# Division (product by the inverse)
M/N, M*N.inv()
```

Out[17]:

$$\left(\begin{bmatrix}1 & -1 & 0\\2 & 3 & 4\\0 & 2 & 7\end{bmatrix} \left(\begin{bmatrix}5 & 6 & 2\\8 & 7 & 7\\5 & 1 & 1\end{bmatrix}\right)^{-1}, \quad \begin{bmatrix}-\frac{1}{4} & \frac{1}{108} & \frac{47}{108}\\-\frac{1}{4} & \frac{77}{108} & -\frac{53}{108}\\-\frac{5}{4} & \frac{55}{36} & -\frac{43}{36}\end{bmatrix}\right)$$

In [18]:
```
# Power
M,M**2, M**3
```

Out[18]:

$$\left(\begin{bmatrix} 1 & -1 & 0 \\ 2 & 3 & 4 \\ 0 & 2 & 7 \end{bmatrix}, \quad \begin{bmatrix} -1 & -4 & -4 \\ 8 & 15 & 40 \\ 4 & 20 & 57 \end{bmatrix}, \quad \begin{bmatrix} -9 & -19 & -44 \\ 38 & 117 & 340 \\ 44 & 170 & 479 \end{bmatrix} \right)$$

Important: Matrix operate as mathematical matrices, thus it is not possible to add (or substract) scalars. It must be done using an auxiliary `ones()` Matrix:

```
In [19]: # These operations are not allowed
         # M + 2
         # 3 - M

         print( 'Adding or substracting a scalar: M + 2, 3 - M' )
         M, M + 2 * sp.ones(M.rows,M.cols), 3 * sp.ones(M.rows,M.cols) - M
```

```
Adding or substracting a scalar: M + 2, 3 - M
```

Out[19]:

$$\left(\begin{bmatrix} 1 & -1 & 0 \\ 2 & 3 & 4 \\ 0 & 2 & 7 \end{bmatrix}, \quad \begin{bmatrix} 3 & 1 & 2 \\ 4 & 5 & 6 \\ 2 & 4 & 9 \end{bmatrix}, \quad \begin{bmatrix} 2 & 4 & 3 \\ 1 & 0 & -1 \\ 3 & 1 & -4 \end{bmatrix} \right)$$

We have done all these examples with "numeric" matrices, but *sympy* operates in the same way with "symbolic" matrices.

```
In [20]: import sympy as sp
         sp.init_printing()

         # Define a matrix with Symbols
         sp.var("x")
         M = sp.Matrix([[x, 2*x, x], [x*x, 3, 4], [x-1, x, x**3]])

         print( "Matrix M" )
         M
```

```
Matrix M
```

Out[20]:

$$\begin{bmatrix} x & 2x & x \\ x^2 & 3 & 4 \\ x-1 & x & x^3 \end{bmatrix}$$

```
In [21]: M**2
```

Out[21]:

$$\begin{bmatrix} 2x^3 + x^2 + x(x-1) & 3x^2 + 6x & x^4 + x^2 + 8x \\ x^3 + 3x^2 + 4x - 4 & 2x^3 + 4x + 9 & 5x^3 + 12 \\ x^3(x-1) + x^3 + x(x-1) & x^4 + 2x(x-1) + 3x & x^6 + x(x-1) + 4x \end{bmatrix}$$

sympy does not attempt to simplify the resulting expression, it can be instructed to do so using the method `simplify()`:

```
In [22]: N = M**2
         N.simplify()
         N
```

Out[22]:

$$\left[\begin{matrix}x \left(2 x^{2} + 2 x - 1\right) & 3 x \left(x + 2\right) & x \left(x^{3} + x + 8\right)\\x^{3} + 3 x^{2} + 4 x - 4 & 2 x^{3} + 4 x + 9 & 5 x^{3} + 12\\x \left(x^{3} + x - 1\right) & x \left(x^{3} + 2 x + 1\right) & x \left(x^{5} + x + 3\right)\end{matrix}\right]$$

```
In [23]: M**-1
```

Out[23]:

$$\left[\begin{matrix}\frac{2 x}{- 2 x^{2} + 3} - \frac{1}{2 x^{5} - 4 x^{3} - x + 5} \left(1 - \frac{- 2 x^{2} + 8}{- 2 x^{2} + 3}\right) \left(2 x^{2} - 3\right) \left(\frac{x \left(- x + 2\right)}{- 2 x^{2} + 3} - \frac{1}{x} \left(x - 1\right)\right) + \frac{1}{x} & \frac{\left(1 - \frac{- 2 x^{2} + 8}{- 2 x^{2} + 3}\right) \left(- x + 2\right) \left(2 x^{2} - 3\right)}{\left(- 2 x^{2} + 3\right) \left(2 x^{5} - 4 x^{3} - x + 5\right)} - \frac{2}{- 2 x^{2} + 3} & \cdots\\- \frac{x}{- 2 x^{2} + 3} - \frac{\left(- x^{2} + 4\right) \left(2 x^{2} - 3\right) \left(\frac{x \left(- x + 2\right)}{- 2 x^{2} + 3} - \frac{1}{x} \left(x - 1\right)\right)}{\left(- 2 x^{2} + 3\right) \left(2 x^{5} - 4 x^{3} - x + 5\right)} & \frac{\left(- x + 2\right) \left(- x^{2} + 4\right) \left(2 x^{2} - 3\right)}{\left(- 2 x^{2} + 3\right)^{2} \left(2 x^{5} - 4 x^{3} - x + 5\right)} + \frac{1}{- 2 x^{2} + 3} & \cdots\\\frac{\left(2 x^{2} - 3\right) \left(\frac{x \left(- x + 2\right)}{- 2 x^{2} + 3} - \frac{1}{x} \left(x - 1\right)\right)}{2 x^{5} - 4 x^{3} - x + 5} & - \frac{\left(- x + 2\right) \left(2 x^{2} - 3\right)}{\left(- 2 x^{2} + 3\right) \left(2 x^{5} - 4 x^{3} - x + 5\right)} & \cdots\end{matrix}\right]$$

```
In [24]: N = M**-1
         N.simplify()
         N
```

Out[24]:

$$\left[\begin{matrix}\frac{- 3 x^{2} + 4}{2 x^{5} - 4 x^{3} - x + 5} & \frac{x \left(2 x^{2} - 1\right)}{2 x^{5} - 4 x^{3} - x + 5} & - \frac{5}{2 x^{5} - 4 x^{3} - x + 5}\\\frac{x^{5} - 4 x + 4}{x \left(2 x^{5} - 4 x^{3} - x + 5\right)} & \frac{- x^{3} + x - 1}{2 x^{5} - 4 x^{3} - x + 5} & \frac{- x^{2} + 4}{2 x^{5} - 4 x^{3} - x + 5}\\\frac{- x^{3} + 3 x - 3}{x \left(2 x^{5} - 4 x^{3} - x + 5\right)} & \frac{- x + 2}{2 x^{5} - 4 x^{3} - x + 5} & \frac{2 x^{2} - 3}{2 x^{5} - 4 x^{3} - x + 5}\end{matrix}\right]$$

`simplify()` is a transformation of the Matrix object. It returns `None` and, if possible modifies the inplace the given Expression.

1.4 Other advanced operations

Apart from basic Matrix algebra `sympy`also provide other advanced operations.

1.4.1 Transpose

The `T` data member of a Martrix object allows to access the transpose representation of a given matrix. Now for a symbolic matrix.

```
In [25]: import sympy as sp
         sp.init_printing()

         sp.var("x")
         M = sp.Matrix([[x, 2*x, 0], [2, x-1, 4], [x+1, 2, 7]])

         print( "Transpose of M" )
         M,M.T
```

```
Transpose of M
```

Out[25]:

$$\left(\left[\begin{matrix}x & 2 x & 0\\2 & x - 1 & 4\\x + 1 & 2 & 7\end{matrix}\right], \ \left[\begin{matrix}x & 2 & x + 1\\2 x & x - 1 & 2\\0 & 4 & 7\end{matrix}\right]\right)$$

1.4.2 Determinant

The determinat of a Matrix can be calculated using the method `det()`.

```
In [26]: import sympy as sp
         sp.init_printing()

         sp.var("x")
         M = sp.Matrix([[x, 2*x, 0], [2, x-1, 4], [x+1, 2, 7]])

         print( "Determinant of M" )
         M,M.det()
```

```
Determinant of M
```

Out[26]:

$$\left(\begin{bmatrix} x & 2x & 0 \\ 2 & x-1 & 4 \\ x+1 & 2 & 7 \end{bmatrix}, \quad 15x^2 - 35x \right)$$

1.5 Generator vectors of the kernel

The `nullspace()` method provides a base of the kernel, subspace that solves the equation:

$$M \cdot v = 0$$

```
In [27]: import sympy as sp
         sp.init_printing()

         # We define a matrix
         M = sp.Matrix([[1, 2, 3, 0, 0], [4, 10, 0, 0, 1]])

         M,M.nullspace()
```

Out[27]:

$$\left(\begin{bmatrix} 1 & 2 & 3 & 0 & 0 \\ 4 & 10 & 0 & 0 & 1 \end{bmatrix}, \quad \left[\begin{bmatrix} -15 \\ 6 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \quad \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \quad \begin{bmatrix} 1 \\ -\frac{1}{2} \\ 0 \\ 0 \\ 1 \end{bmatrix} \right] \right)$$

```
In [28]: for v in M.nullspace():
             print( M * v )
```

```
Matrix([[0], [0]])
Matrix([[0], [0]])
Matrix([[0], [0]])
```

1.5.1 Eigenvalues and Eigenvectors

The method `eignevals()` returns a dictionary with the eigenvalues (keys) and their multiplicities (values). The full information, in the form of tuples, is returned by `eigenvects()`. Each item of the tuple includes: eigenvalue (λ), multiplicity and eigenvectors (v). These methods provide the solution to the equation:

$$M \cdot v = \lambda \cdot v$$

```
In [29]: import sympy as sp
         sp.init_printing()

         # Define a matrix
         M = sp.Matrix([[3, -2, 4, -2], [5, 3, -3, -2], [5, -2, 2, -2], [5, -2, -3, 3]])

         M,M.eigenvals()
```

Out[29]:

$$\left(\begin{bmatrix}3 & -2 & 4 & -2\\5 & 3 & -3 & -2\\5 & -2 & 2 & -2\\5 & -2 & -3 & 3\end{bmatrix}, \quad \{-2:1, \quad 3:1, \quad 5:2\}\right)$$

```
In [30]: M.eigenvects()
```

Out[30]:

$$\left[\left(-2, \quad 1, \quad \left[\begin{bmatrix}0\\1\\1\\1\end{bmatrix}\right]\right), \quad \left(3, \quad 1, \quad \left[\begin{bmatrix}1\\1\\1\\1\end{bmatrix}\right]\right), \quad \left(5, \quad 2, \quad \left[\begin{bmatrix}1\\1\\1\\0\end{bmatrix}, \quad \begin{bmatrix}0\\-1\\0\\1\end{bmatrix}\right]\right)\right]$$

```
In [31]: for val,mult,vects in M.eigenvects():
             for vect in vects:
                 print( M * vect, ' = ', val * vect )

Matrix([[0], [-2], [-2], [-2]])  =  Matrix([[0], [-2], [-2], [-2]])
Matrix([[3], [3], [3], [3]])  =  Matrix([[3], [3], [3], [3]])
Matrix([[5], [5], [5], [0]])  =  Matrix([[5], [5], [5], [0]])
Matrix([[0], [-5], [0], [5]])  =  Matrix([[0], [-5], [0], [5]])
```

1.5.2 Diagonalization

The method `M.diagonalize()` returns a tuple (P, D) that solves the equation:

$$M = P \times D \times P^{-1}$$

where D is a diagonal matrix with the eigenvalues of M.

```
In [32]: import sympy as sp
         sp.init_printing()

         sp.var("x")
         M = sp.Matrix([[2, x], [x, 3]])
         P_M, D_M = M.diagonalize()

         invP_M = P_M**-1
         invP_M.simplify()

         M, P_M, D_M, invP_M
```

Out[32]:

$$\left(\begin{bmatrix}2 & x\\x & 3\end{bmatrix}, \quad \begin{bmatrix}-\frac{2x}{\sqrt{4x^2+1}-1} & \frac{2x}{\sqrt{4x^2+1}+1}\\1 & 1\end{bmatrix}, \quad \begin{bmatrix}-\frac{1}{2}\sqrt{4x^2+1}+\frac{5}{2} & 0\\0 & \frac{1}{2}\sqrt{4x^2+1}+\frac{5}{2}\end{bmatrix}, \quad \begin{bmatrix}-\frac{x}{\sqrt{4x^2+1}} & \frac{1}{2}-\frac{1}{2\sqrt{4x^2+1}}\\\frac{x}{\sqrt{4x^2+1}} & \frac{1}{2}+\frac{1}{2\sqrt{4x^2+1}}\end{bmatrix}\right)$$

```
In [33]: P_M * D_M * invP_M
```

Out[33]:

$$\left[\begin{matrix}\frac{2x^2\left(-\frac{1}{2}\sqrt{4x^2+1}+\frac{5}{2}\right)}{\sqrt{4x^2+1}\left(\sqrt{4x^2+1}-1\right)}+\frac{2x^2\left(\frac{1}{2}\sqrt{4x^2+1}+\frac{5}{2}\right)}{\sqrt{4x^2+1}\left(\sqrt{4x^2+1}+1\right)} & -\frac{2x}{\sqrt{4x^2+1}-1}\left(\frac{1}{2}-\frac{1}{2\sqrt{4x^2+1}}\right)\left(-\frac{1}{2}\sqrt{4x^2+1}+\frac{5}{2}\right)+\frac{2x}{\sqrt{4x^2+1}+1}\left(\frac{1}{2}+\frac{1}{2\sqrt{4x^2+1}}\right)(\\ -\frac{x}{\sqrt{4x^2+1}}\left(-\frac{1}{2}\sqrt{4x^2+1}+\frac{5}{2}\right)+\frac{x\left(\frac{1}{2}\sqrt{4x^2+1}+\frac{5}{2}\right)}{\sqrt{4x^2+1}} & \left(\frac{1}{2}-\frac{1}{2\sqrt{4x^2+1}}\right)\left(-\frac{1}{2}\sqrt{4x^2+1}+\frac{5}{2}\right)+\left(\frac{1}{2}+\frac{1}{2\sqrt{4x^2+1}}\right)\left(\frac{1}{2}\sqrt{4x^2+1}\right.\end{matrix}\right.$$

```
In [34]: M_sol = P_M * D_M * invP_M
         M_sol.simplify()
         M_sol
```

Out[34]:

$$\begin{bmatrix}2 & x \\ x & 3\end{bmatrix}$$

1.5.3 Solving systems of equations

Given the system (same we used with *numpy*):

$$\begin{array}{ccccccc} x & - & y & + & z & = & 4 \\ 2x & + & y & - & 3z & = & 1 \\ 7x & - & y & - & 3z & = & 14 \end{array}$$

Three different ways to solve this system:

```
In [35]: import sympy as sp
         sp.init_printing()

         sp.var("x, y, z")
         expre1 = x - y + z - 4
         expre2 = 2*x + y - 3*z - 1
         expre3 = 7*x - y - 3*z - 14

         sp.solve( [expre1, expre2, expre3] )
```

Out[35]:

$$\left\{x : \frac{2z}{3}+\frac{5}{3}, \quad y : \frac{5z}{3}-\frac{7}{3}\right\}$$

```
In [36]: eq1 = sp.Eq( x - y + z, 4 )
         eq2 = sp.Eq( 2*x + y - 3*z, 1 )
         eq3 = sp.Eq( 7*x - y - 3*z, 14 )

         sp.solve( [eq1, eq2, eq3] )
```

Out[36]:

$$\left\{x : \frac{2z}{3}+\frac{5}{3}, \quad y : \frac{5z}{3}-\frac{7}{3}\right\}$$

```
In [37]: M = sp.Matrix( [[1, -1, 1], [2, 1, -3], [7, -1, -3]] )
         eq = sp.Eq( M * sp.Matrix([x,y,z]), sp.Matrix([4, 1, 14]) )
```

```
In [38]: sp.solve(eq )
```

Out[38]:

$$\left[\left\{x : \frac{2z}{3}+\frac{5}{3}, \quad y : \frac{5z}{3}-\frac{7}{3}\right\}\right]$$