Linear Algebra Step by Step in SymPy

Dario Beraldi

July 22, 2015

Contents

1. Linear Equations and Matrices **2**
Solving linear systems . . . 2
Solve by row echelon form . . . 5
Vector arithmetic . . . 7
Arithmetic of Matrices . . . 9
Matrix Algebra . . . 13
Type of solutions . . . 14
The inverse matrix method . . . 16

2. Euclidean Space **19**
Properties of vectors . . . 19
Further Properties of Vectors . . . 22
Linear Independence . . . 26
Basis and Spanning Set . . . 28

3. General vector spaces **33**
3.3 Linear independence and basis . . . 33
3.6 Linear systems revisted . . . 39

5. Linear Transformations **48**
Introduction to linear transformations . . . 48
5.2 Kernel and range of a linear transformation . . . 53
5.3 Rank of a linear transformation . . . 54

6. Determinant and the Inverse Matrix **55**
Determinant of a Matrix . . . 56
Determinant of other matrices . . . 58
Properties of determinants . . . 61
LU factorization . . . 62

7. Eigenvalues and Eigenvectors **64**
Introduction to eigenvalues and eigenvectors 64
Properties of eigenvalues and eigenvectors 67
Diagonalization 69
Diagonalization of Symmetric Matrices 71
Singular Value Decomposition 73

Appendix **76**
Plotting with `matplotlib.pyplot` 76
Principal components 79
PageRank 79
Line of best fit (least squares) 84

Abstract

Excerpts from *Linear Algebra Step by Step* by K. Singh with `SymPy` implementation.

Start session with interactive `python` as:

```
ipython
from sympy import *
from matplotlib import pylab as plt

init_printing()
```

Starting `isympy` seems to interfere with `matplotlib` later.

1. Linear Equations and Matrices

Solving linear systems

Exercises 1.1 2f

$$\begin{aligned} ex - ey &= 2 \\ ex + ey &= 0 \end{aligned} \tag{1}$$

```
x, y= symbols('x y', real= true)
eq1= Eq(E*x - E*y, 2)
eq2= Eq(E*x + E*y, 0)
sols= solve([eq1, eq2])
```

Solved for:

$$\left\{x : e^{-1}, \quad y : -\tfrac{1}{e}\right\} \tag{2}$$

Check solutions by substituting them in the original equations 1

```
eq1.subs(sols)
True
eq2.subs(sols)
True
```

Solve the system 1 using matrix notation

```
coeffs= Matrix([[E, -E], [E, E]])
const= Matrix([2, 0])
[coeffs, const]
```

$$\left[\begin{bmatrix} e & -e \\ e & e \end{bmatrix}, \quad \begin{bmatrix} 2 \\ 0 \end{bmatrix}\right] \tag{3}$$

If everything is correct, solutions are consistent with 2:

```
coeffs.solve(const)
```

$$\begin{bmatrix} e^{-1} \\ -\frac{1}{e} \end{bmatrix} \tag{4}$$

Exercises 1.1 4a

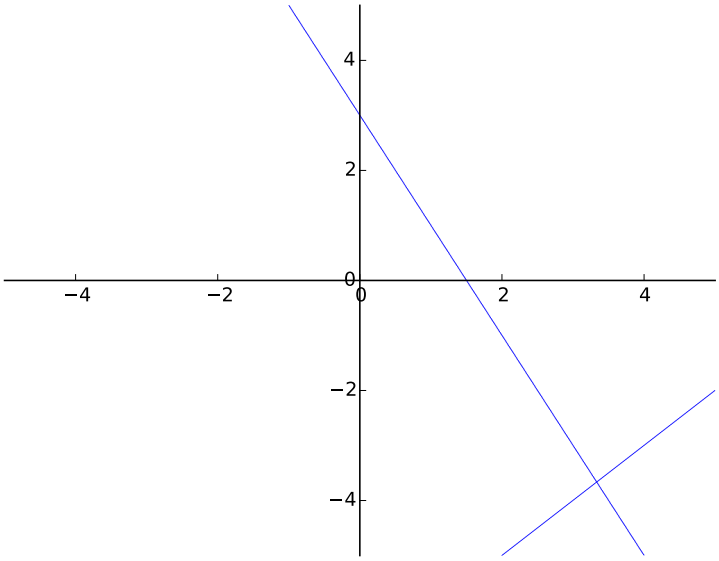
Plot the graphs of these linear equations

$$\begin{aligned} 2x + y &= 3 \\ x - y &= 7 \end{aligned} \tag{5}$$

```
eq1= Eq(2*x + y, 3)
eq2= Eq(x - y, 7)
```

Plot

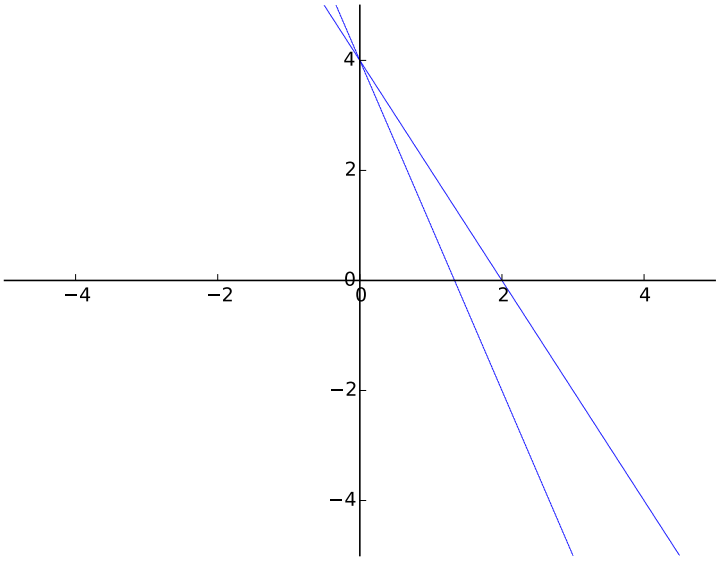
```
pp1= plot_implicit(eq1)
pp2= plot_implicit(eq2)
pp1.extend(pp2)
pp1.save('figs/ex4a.pdf')
```



Exercises 1.1 5b

```
eq1= Eq(12*x + 4*y, 16)
eq2= Eq(8*x + 4*y, 16)

pp1= plot_implicit(eq1)
pp2= plot_implicit(eq2)
pp1.extend(pp2)
pp1.save('figs/ex5b.pdf')
```



Solve by row echelon form

Given a linear system of equations, the solutions can be found by Gaussian elimination:

- Extract the coefficients and constants from the equations and put them in an augmented matrix.
- Transform the augmented matrix in reduced row echelon form (rref). This is the result of the Gaussian elimination process.
- The last columns of the rref matrix lists the solutions of the linear system.

Reminder: rref means that every leading coefficient is 1 and is the only nonzero entry in its column [1].

Exercises 1.2 2a

$$\begin{aligned} x + 2y + 3z &= 12 \\ 2x - y + 5z &= 3 \\ 3x + 3y + 6z &= 21 \end{aligned} \tag{6}$$

```
eq1= Eq(x + 2*y + 3*z, 12)
eq2= Eq(2*x - y + 5*z, 3)
eq3= Eq(3*x + 3*y + 6*z, 21)
```

[1] http://en.wikipedia.org/wiki/Row_echelon_form

Represent the linear systen as *augmented* matrix, where the last column holds the constants:

$$M = \begin{bmatrix} 1 & 2 & 3 & \mathit{12} \\ 2 & -1 & 5 & \mathit{3} \\ 3 & 3 & 6 & \mathit{21} \end{bmatrix} \tag{7}$$

In SymPy use the `Poly` class to conveniently extract the coefficients at the left hand side of the equations:

```
M= Matrix([
    Poly(eq1.lhs).coeffs(),
    Poly(eq2.lhs).coeffs(),
    Poly(eq3.lhs).coeffs()
])
const= Matrix([eq1.rhs, eq2.rhs, eq3.rhs])
M= const.col_insert(0, M)
```

In reduced row echelon form, with indexes of the pivot variables on the right:

$$RREF = \left(\begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 4 \\ 0 & 0 & 1 & 1 \end{bmatrix}, \quad \begin{bmatrix} 0, & 1, & 2 \end{bmatrix} \right) \tag{8}$$

Solutions can be read from last column of the RREF $\begin{bmatrix} x:1 \\ y:4 \\ z:1 \end{bmatrix}$. Verify the solutions solve the initial system of equations:

```
rref= M.rref()
sols= rref[0].col(-1)

eq1.subs({x:sols[0], y:sols[1], z:sols[2]}) # True
eq2.subs({x:sols[0], y:sols[1], z:sols[2]}) # True
eq3.subs({x:sols[0], y:sols[1], z:sols[2]}) # True

# Or the same returning dict {x: 1, y: 4, z: 1}
solve([eq1, eq2, eq3])
```

Exercises 1.2 3d

Linear system:

$$\begin{aligned} -2x + 3y - 2z &= 8 \\ -x + 2y - 10z &= 0 \\ 5x - 7y + 4z &= -20 \end{aligned} \tag{9}$$

Augmented matrix:

$$\begin{bmatrix} -2 & 3 & -2 & 8 \\ -1 & 2 & -10 & 0 \\ 5 & -7 & 4 & -20 \end{bmatrix} \tag{10}$$

Reduced row echelon form with solutions in the last column:

$$\left(\begin{bmatrix} 1 & 0 & 0 & -3 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & \frac{1}{2} \end{bmatrix}, \quad \begin{bmatrix} 0, & 1, & 2 \end{bmatrix} \right) \tag{11}$$

In SymPy :

```
eq1= Eq(-2*x + 3*y -2*z, 8)
eq2= Eq(-x + 2*y - 10*z, 0)
eq3= Eq(5*x - 7*y + 4*z, -20)

M= Matrix([
    Poly(eq1.lhs).coeffs(),
    Poly(eq2.lhs).coeffs(),
    Poly(eq3.lhs).coeffs()
])
const= Matrix([eq1.rhs, eq2.rhs, eq3.rhs])
M= const.col_insert(0, M)
rref= M.rref()
sols= rref[0].col(-1)

# Checked: {x: -3, y: 1, z: 1/2}
solve([eq1, eq2, eq3])
```

Vector arithmetic

Exercise 1.3.1

Given two vectors:

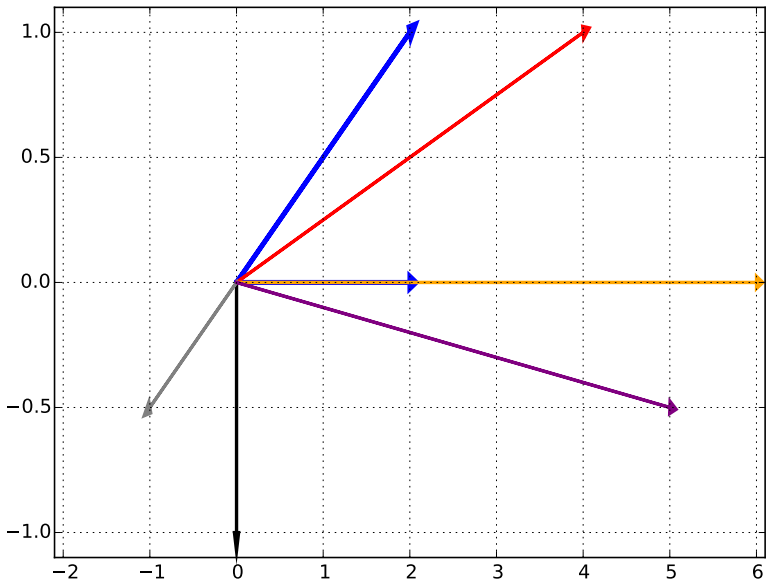
```
va= Matrix([2, 0])
vb= Matrix([2, 1])
```

plot the results of the operations

(a-e)

```
vA= va + vb          # [4 1]
vB= va - vb          # [0 -1]
vC= 3 * va           # [6 0]
vD= -1/2 * vb        # [-1 -0.5]
vE= 3*va - 1/2 * vb  # [5 -0.5]
```

And plot vectors




```
plt.arrow(0, 0, float(va[0]), float(va[1]), lw= 3, color= 'b', head_width=0.05)
plt.arrow(0, 0, float(vb[0]), float(vb[1]), lw= 3, color= 'b', head_width=0.05)
plt.arrow(0, 0, float(vA[0]), float(vA[1]), lw= 2, color= 'r', head_width=0.05)
plt.arrow(0, 0, float(vB[0]), float(vB[1]), lw= 2, color= 'black', head_width=0.05)
plt.arrow(0, 0, float(vC[0]), float(vC[1]), lw= 2, color= 'orange', head_width=0.05)
plt.arrow(0, 0, float(vD[0]), float(vD[1]), lw= 2, color= 'grey', head_width=0.05)
plt.arrow(0, 0, float(vE[0]), float(vE[1]), lw= 2, color= 'purple', head_width=0.05)
plt.xlim(-2.1, 6.1)
plt.ylim(-1.1, 1.1)
plt.grid()
plt.savefig('figs/ex1_3.pdf')
plt.close()
```

Exercise 1.3.8

Show that $x\mathbf{u} + y\mathbf{v} = \mathbf{w}$ where $\mathbf{u} = (1\ 0)^T$, $\mathbf{v} = (0\ 1)^T$, and $\mathbf{w} = (x\ y)^T$.

This is a consequence of $x(1\ 0) = (x\ 0)$ and $y(0\ 1) = (y\ 0)$. So that $(x\ 0)+(0\ y) = (x\ y)$

```
x, y= symbols('x y')
u= Matrix([1, 0])
v= Matrix([0, 1])
w= Matrix([x, y])
Eq(x*u + y*v, w) # True
```

Exercise 1.3.12

Find the real numbers x, y and z, if

$$\begin{bmatrix} x-2z \\ 2x+y \\ -y+6z \end{bmatrix} = \begin{bmatrix} 5 \\ 3 \\ 17 \end{bmatrix} \tag{12}$$

Which is solved for:

$$sols = \left[\left\{x:7, \quad y:-11, \quad z:1\right\}\right] \tag{13}$$

```
eq1= Eq(x * Matrix([1, 2, 0]) + y * Matrix([0, 1, -1]) + z * Matrix([-2, 0, 6]),
    Matrix([5, 3, 17]))
sols= solve(eq1)
```

Exercise 1.3.12

Show that vectors $\mathbf{u}$ and $\mathbf{v}$ in space $\mathbb{R}^n$ can be $\mathbf{u} \cdot \mathbf{v} = \mathbf{0}$ even if neither $\mathbf{u}$ or $\mathbf{v}$ are 0 vectors.

Set:

```
u= Matrix([-1, 1])
v= Matrix([1, 1])
u.dot(v) == 0 # True
```

Arithmetic of Matrices

Exercise 1.4.1

Given matrix $\mathbf{B}$, note that $3\mathbf{B} = \mathbf{B} + \mathbf{B} + \mathbf{B}$

$$\begin{bmatrix} 6 & -1 \\ 5 & 3 \end{bmatrix} \tag{14}$$

```
B= Matrix([[6, -1], [5, 3]])
3*B == (B + B + B)
```

Exercise 1.4.6

Note how matrx multiplication can result in zero matrix:

$$\begin{bmatrix} 5 & -1 & -2 \\ 10 & -2 & -4 \\ 15 & -3 & -6 \end{bmatrix} \begin{bmatrix} 1 & 1 & 3 \\ 1 & -1 & -1 \\ 2 & 3 & 8 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{15}$$

```
A= Matrix(3, 3, [5, -1, -2, 10, -2, -4, 15, -3, -6])
B= Matrix(3, 3, [1, 1, 3, 1, -1, -1, 2, 3, 8])
Z= A * B
```

Exercise 1.4.8

$\mathbf{x}_n = \mathbf{A}^n\mathbf{x}$ Describes a **discrete dynamical system**. Apply this formula to

$$A = \begin{bmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{bmatrix} \tag{16}$$

```
A= Matrix(2, 2, [1/2, 1/2, 1/2, 1/2])
```

For the matrix in 16 $A^n = A$:

```
A**2 == A # True
A**3 == A # True
A**10 == A # True
```

matrix in 16 is a Markov matrix since the colum sums equal 1.

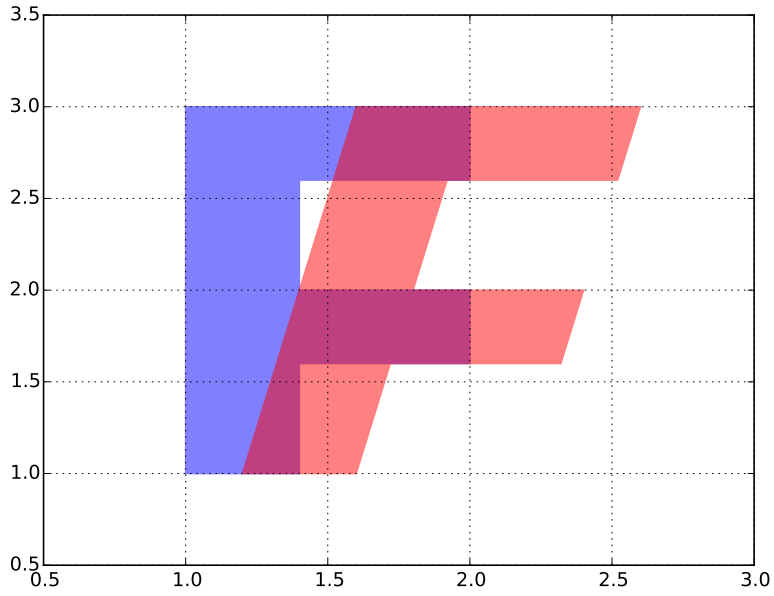
Exercise 1.4.10

Determine the image of the matrix **F** after transformation **AF**:

$$A = \begin{bmatrix} 1 & 0.2 \\ 0 & 1 \end{bmatrix} \tag{17}$$

$$F = \begin{bmatrix} 1 & 1 & 2 & 2 & 1.4 & 1.4 & 2 & 2 & 1.4 & 1.4 \\ 1 & 3 & 3 & 2.6 & 2.6 & 2 & 2 & 1.6 & 1.6 & 1 \end{bmatrix} \tag{18}$$

$$AF = \begin{bmatrix} 1.2 & 1.6 & 2.6 & 2.52 & 1.92 & 1.8 & 2.4 & 2.32 & 1.72 & 1.6 \\ 1 & 3 & 3 & 2.6 & 2.6 & 2 & 2 & 1.6 & 1.6 & 1 \end{bmatrix} \tag{19}$$



This is showing that the matrix operation $\mathbf{AF} = \mathbf{F_2}$ can be seen as $\mathbf{f}(\mathbf{F}) = \mathbf{F_2}$. That is, the matrix on the left-hand ($\mathbf{A}$) side acts like a function that transforms its argument ($\mathbf{F}$).

```
A= Matrix([[1, 0.2], [0, 1]])
F= Matrix([
    [1, 1, 2, 2, 1.4, 1.4, 2, 2, 1.4, 1.4],
    [1, 3, 3, 2.6, 2.6, 2, 2, 1.6, 1.6, 1]])

AF= A*F

verts_F= []
for i in range(F.cols):
    verts_F.append((F[0, i], F[1, i]))
verts_AF= []
for i in range(AF.cols):
    verts_AF.append((AF[0, i], AF[1, i]))

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
poly_F = patches.Polygon(verts_F, color= 'b', alpha= 0.5)
poly_AF = patches.Polygon(verts_AF, color= 'r', alpha= 0.5)
ax.set_xlim((0.5, 3))
ax.set_ylim((0.5, 3.5))
ax.add_patch(poly_F)
ax.add_patch(poly_AF)
plt.grid()
plt.savefig('figs/1_4_10.pdf')
```

Exercise 1.4.11

```
O= Matrix.zeros(2, 1)
X= Matrix([x, y])

A= Matrix([[1, 2], [3, 5]])
xy= A.solve(O) # x:0, y:0
# Or
xy= A.LUsolve(O)
# Or
solve(A*X)

A= Matrix([[2, 7], [3, 15]]) # x:0, y:0
xy= A.solve(O)

A= Matrix([[1, 4], [3, 12]]) # Linearly dependent!
xy= A.solve(O)
>>> ValueError: Matrix det == 0; not invertible.
```

Exercise 1.4.12

Determine whether vector $\mathbf{w}$ is a linear combination of vector $\mathbf{u}$ and $\mathbf{v}$.
Effectively this is asking if the equation $x\mathbf{u} + y\mathbf{v} = \mathbf{w}$ can be resolved.

- Create an augmented matrix as [u, v, w].
- Transform the augmented matrix in row echelon form.
- Get coefficients x and y from last column of RREF. The three vectors are linearly independent and $\mathbf{w} \neq \mathbf{u} + \mathbf{v}$ ($\mathbf{w}$ is not a linear combination of $\mathbf{u}$ and $\mathbf{v}$).

```
w= Matrix(2, 1, [1, 0])
u= Matrix(2, 1, [5, 8])
v= Matrix(2, 1, [2, 4])
A=  w.col_insert(0, v).col_insert(0, u)
A.rref()
```

$$REF = \left(\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & -2 \end{bmatrix}, \quad \begin{bmatrix} 0, & 1 \end{bmatrix} \right) \tag{20}$$

$\mathbf{w}$ is a linear combination. In fact $1\mathbf{u} - 2\mathbf{v} = \mathbf{w}$

```
w= Matrix(2, 1, [0, 1])
u= Matrix(2, 1, [5, 8])
v= Matrix(2, 1, [2, 4])
A=  w.col_insert(0, v).col_insert(0, u)
A.rref()
```

$$\left(\begin{bmatrix} 1 & 0 & -\frac{1}{2} \\ 0 & 1 & \frac{5}{4} \end{bmatrix}, \quad \begin{bmatrix} 0, & 1 \end{bmatrix} \right) \tag{21}$$

```
w= Matrix(3, 1, [1, 2, 3])
u= Matrix(3, 1, [1, 0, 0])
v= Matrix(3, 1, [0, 1, 0])
A=  w.col_insert(0, v).col_insert(0, u)
A.rref()
```

$$REF = \left(\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \begin{bmatrix} 0, & 1, & 2 \end{bmatrix} \right) \tag{22}$$

$\mathbf{w}$ cannot be a linear combination. In fact $0\mathbf{u} + 0\mathbf{v} \neq [1\ 2\ 3]^T$

```
w= Matrix(3, 1, [1, 2, 3])
u= Matrix(3, 1, [4, 8, 0])
v= Matrix(3, 1, [1, 2, -3/7])
A=  w.col_insert(0, v).col_insert(0, u)
A.rref()
```

$$\left(\begin{bmatrix} 1 & 0 & 2.0 \\ 0 & 1.0 & -7.0 \\ 0 & 0 & 0 \end{bmatrix}, \quad \begin{bmatrix} 0, & 1 \end{bmatrix} \right) \tag{23}$$

Matrix Algebra

Exrcises 1.5.2

Given $A = \begin{bmatrix} 5 & -1 & -2 \\ 1 & -3 & 2 \end{bmatrix}$

A) Find $\mathbf{B}$ suc that $\mathbf{A} + \mathbf{B} = \mathbf{O}$ with $\mathbf{O}$ a 2 x 3 matrix.

Requires $\mathbf{A} = \mathbf{B} \Rightarrow \mathbf{B} = -\mathbf{A}$

```
A= Matrix([[5, -1, -2], [1, -3, 2]])
O= Matrix.zeros(2, 3)
```

Exrcises 1.5.7

Determine the scalar λ so that $\mathbf{Ax} = \lambda\mathbf{x}$ with $\mathbf{A} = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$ and $\mathbf{x} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$

$\lambda = 3$ since $\mathbf{Ax} = [3\ 3]^T$.

```
A= Matrix([[2, 1], [1, 2]])
x= Matrix(2, 1, [1, 1])
l= symbols('lambda')
solve(Eq(A*x, l*x)) # Lambda= 3
```

Exrcises 1.5.10

Given $\mathbf{T}$ a transition matrix from a Markov chain and $\mathbf{p}$ a probability vector. Note that the probility vector must sum to 1.

What is the probility that the chain is in a given state after k steps?

The answer is given by $\mathbf{p}_k = \mathbf{T}^k\mathbf{p}$.

For:

$$\begin{bmatrix} 0.6 & 0.7 \\ 0.4 & 0.3 \end{bmatrix} \tag{24}$$

$$\begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} \tag{25}$$

Determine $\mathbf{p}_k$ for k in 1, 2, 10, 100, 100000:

$p_{k=1} = \begin{bmatrix} 0.65 \\ 0.35 \end{bmatrix}$

$p_{k=2} = \begin{bmatrix} 0.635 \\ 0.365 \end{bmatrix}$

$p_{k=10} = \begin{bmatrix} 0.63636363635 \\ 0.36363636365 \end{bmatrix}$

$$p_{k=100} = \begin{bmatrix} 0.636363636363634 \\ 0.363636363636362 \end{bmatrix}$$

$$p_{k=100000} = \begin{bmatrix} 0.636363636361267 \\ 0.36363636363501 \end{bmatrix}$$

Note how the probability vector converges in the long period. The probability vector at the initial state $k = 0$ might represent the proportion of different species in the environment. Given the probility of change in $\mathbf{T}$, we can ask the question: What proportion of species we will see after k iterations (*e.g* generations)?

```
T= Matrix(2,2, [0.6, 0.7, 0.4, 0.3])
p= Matrix(2, 1, [0.5, 0.5])
ks= [1, 2, 10, 100, 100000]

for k in ks:
    p_k= T**k * p
    print latex(p_k)
```

Type of solutions

Exrcises 1.7.8

$$\mathbf{M} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 6 \\ 0 & 0 & 0 & 1 & 1 & 1 & 15 \\ 1 & 0 & 0 & 1 & 0 & 0 & 5 \\ 0 & 1 & 0 & 0 & 1 & 0 & 7 \\ 0 & 0 & 1 & 0 & 0 & 1 & 9 \end{bmatrix} \tag{26}$$

$$\mathbf{M_{rref}} = \left(\begin{bmatrix} 1 & 0 & 0 & 0 & -1 & -1 & -10 \\ 0 & 1 & 0 & 0 & 1 & 0 & 7 \\ 0 & 0 & 1 & 0 & 0 & 1 & 9 \\ 0 & 0 & 0 & 1 & 1 & 1 & 15 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0, & 1, & 2, & 3 \end{bmatrix} \right) \tag{27}$$

There are six variables with coefficents arranged as in matrix $\mathbf{M}$. In RREF it appears the first four variables, x_1 to x_4, have 1 as leading coefficient, as reported by `SymPy` in the list of pivot variables indexes on the right (0, 1, 2, 3). Variables x_5 and x_6 are *free* with coefficients given in the respective columns. The solutions are therefore:

$$\left\{ x_1 : x_5 + x_6 - 10, \quad x_2 : -x_5 + 7, \quad x_3 : -x_6 + 9, \quad x_4 : -x_5 - x_6 + 15 \right\} \tag{28}$$

```
M= Matrix([
[1, 1, 1, 0, 0, 0, 6],
[0, 0, 0, 1, 1, 1, 15],
[1, 0, 0, 1, 0, 0, 5],
[0, 1, 0, 0, 1, 0, 7],
[0, 0, 1, 0, 0, 1, 9],
])
M_rref= M.rref()
```

Solve system of equations:

```
x1, x2, x3, x4, x5, x6= symbols('x1:7')
r1= Eq(x1 + x2 + x3, 6)
r2= Eq(x4 + x5 + x6, 15)
r3= Eq(x1 + x4, 5)
c1= Eq(x2 + x5, 7)
c2= Eq(x3 + x6, 9)
sols= solve([r1, r2, r3, c1, c2])
```

Exrcises 1.7.11

```
x, y, z, t= symbols('x y z t', real= True)

eq1= Eq(2*x - 4*y + 4*z + 0.077*t, 3.86)
eq2= Eq(     -2*y + 2*z - 0.056*t, -3.47)
eq3= Eq(2*x - 2*y,                 0)
sols= solve([eq1, eq2, eq3], [x, y, z])
```

The solutions `sols` are in 32.

Solving via augmented matrix in reduced row echelon form:

```
M= Matrix([
    [2, -4, 4, 0.077, 3.86],
    [0, -2, 2, -0.056, -3.47],
    [2, -2, 0, 0, 0]
])
M_rref= M.rref()
```

$$\begin{aligned} 2x - 4y + 4z + 0.077t &= 3.86, \\ -2y + 2z - 0.056t &= -3.47, \\ 2x - 2y &= 0 \end{aligned} \tag{29}$$

$$M = \begin{bmatrix} 2 & -4 & 4 & 0.077 & 3.86 \\ 0 & -2 & 2 & -0.056 & -3.47 \\ 2 & -2 & 0 & 0 & 0 \end{bmatrix} \tag{30}$$

$$M_{rref} = \left(\begin{bmatrix} 1 & 0 & 0 & 0.0945 & 5.4 \\ 0 & 1 & 0 & 0.0945 & 5.4 \\ 0 & 0 & 1 & 0.0665 & 3.665 \end{bmatrix}, \quad \begin{bmatrix} 0, & 1, & 2 \end{bmatrix} \right) \tag{31}$$

Note that the 4th column holds the coefficients for the free variable t while the last column has the values for x, y, z. Compare to the solutions from `SymPy` `solve`:

$$\left\{ x : -0.0945t + 5.4, \quad y : -0.0945t + 5.4, \quad z : -0.0665t + 3.665 \right\} \tag{32}$$

The inverse matrix method

Exrcises 1.8.3

What effect has the matrix **E** on the matrix **A**. With **A** a generic 3x3 matrix:

```
x1, x2, x3, x4, x5, x6, x7, x8, x9= symbols('x1:10')
k= symbols('k', zero= False)
A= Matrix(3,3, [x1, x2, x3, x4, x5, x6, x7, x8, x9])
```

$$\mathbf{A} = \begin{bmatrix} x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 \\ x_7 & x_8 & x_9 \end{bmatrix} \tag{33}$$

$$\mathbf{E} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{34}$$

```
E_a= Matrix([
    [1, 0, 0],
    [0, -1, 0],
    [0, 0, 1]
])
print latex(E_a * A)
```

$$\mathbf{E_a A} = \begin{bmatrix} x_1 & x_2 & x_3 \\ -x_4 & -x_5 & -x_6 \\ x_7 & x_8 & x_9 \end{bmatrix} \tag{35}$$

```
E_b= Matrix([
    [0, 0, 1],
    [0, 1, 0],
    [1, 0, 0]
])
print latex(E_b * A)
```

$$\mathbf{E_b A} = \begin{bmatrix} x_7 & x_8 & x_9 \\ x_4 & x_5 & x_6 \\ x_1 & x_2 & x_3 \end{bmatrix} \tag{36}$$

```
E_c= Matrix([
    [k, 0, 0],
    [0, 1, 0],
    [0, 0, 1]
])
print latex(E_c * A)
```

$$\mathbf{E_c A} = \begin{bmatrix} kx_1 & kx_2 & kx_3 \\ x_4 & x_5 & x_6 \\ x_7 & x_8 & x_9 \end{bmatrix} \tag{37}$$

```
E_d= Matrix([
    [1, 0, 0],
    [0, 1, 0],
    [0, 0, -1/k]
])
print latex(E_d * A)
```

$$\mathbf{E_d A} = \begin{bmatrix} x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 \\ -\frac{x_7}{k} & -\frac{x_8}{k} & -\frac{x_9}{k} \end{bmatrix} \tag{38}$$

Exrcises 1.8.5

Solve the linear systems using inverse matrix method.

$$\begin{aligned} x + 2y &= 3, \\ -x + 4y &= 5 \end{aligned} \tag{39}$$

The coefficients on the LHS of the equations can be thought as a matrix $\mathbf{A}$. $\mathbf{A}$ transforms the vector of unknown coefficients $\mathbf{x}$ to produce the vector of constants $\mathbf{b}$ on the RHS:

$$\mathbf{Ax} = \mathbf{b} \tag{40}$$

To find the vector $\mathbf{x}$ we can rearrange 39 as:

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b} \tag{41}$$

In case of 39 we have $\mathbf{A} = \begin{bmatrix} 1 & 2 \\ -1 & 4 \end{bmatrix}$, $\mathbf{b} = \begin{bmatrix} 3 \\ 5 \end{bmatrix}$. So $\mathbf{A^{-1}} = \begin{bmatrix} \frac{2}{3} & -\frac{1}{3} \\ \frac{1}{6} & \frac{1}{6} \end{bmatrix}$ and $\mathbf{x} = \mathbf{A^{-1}b} = \begin{bmatrix} \frac{1}{3} \\ \frac{4}{3} \end{bmatrix}$

```
x, y= symbols('x y')
eq1= Eq(x + 2*y, 3)
eq2= Eq(-x + 4*y, 5)

A = Matrix(2, 2, [1, 2, -1, 4])
b = Matrix(2, 1, [3, 5])
Ai= A.inv()
x= Ai * b
```

In case of $\mathbf{Ax} = \mathbf{b}$ as:

$$\begin{bmatrix} 1 & 0 & 2 \\ 2 & 3 & 1 \\ 3 & 6 & 0 \end{bmatrix} \mathbf{x} = \begin{bmatrix} -1 \\ 1 \\ 9 \end{bmatrix} \tag{42}$$

Matrix $\mathbf{A}$ is not invertible and the system has no solutions.

```
A = Matrix([
[1, 0, 2],
[2, 3, 1],
[3, 6, 0]
])
b= Matrix(3, 1,  [-1 , 1, 9])

Ai= A.inv()
# ValueError: Matrix det == 0; not invertible.
```

Exrcises 1.8.7

Leontief input-output model. Represents the production $\mathbf{p}$ and demand $\mathbf{d}$ as a matrix. $\mathbf{p}$ and $\mathbf{d}$ are vectors where each entry represents a good. The production of goods is represented as a matrix where each row states how much units of the other goods (input) is required to produce one unit of that good (output). With this model we can ask how much input is needed to produce a given amount of output.

$$\begin{matrix} & O & E & S \\ O \\ E \\ S \end{matrix} \begin{pmatrix} 0.25 & 0.15 & 0.1 \\ 0.4 & 0.15 & 0.2 \\ 0.15 & 0.2 & 0.2 \end{pmatrix} = \mathbf{A}$$

Reading this matrix row by row, it tells how much Oil, Energy and Services it is needed to produce one unit of O, E, or S. For example 1 unit of O requires 0.25 units of O, 0.15 of E and 0.1 units of S.

From this matrix we can relate the production vector $\mathbf{p}$ to the demand vector $\mathbf{d}$ as:

$$\mathbf{p} = \mathbf{Ap} + \mathbf{d} \tag{43}$$

We want to know $\mathbf{p}$ given $\mathbf{A}$ and the demand vector $\mathbf{d}$. *I.e.* we want to solve 43 *w.r.t.* $\mathbf{p}$ by re-arranging to the form $\mathbf{Ax} = \mathbf{b}$.

$$\mathbf{p} - \mathbf{Ap} = \mathbf{d}$$

then use the identity matrix to obtain the solvable form $\mathbf{Ax} = \mathbf{b}$:

$$\mathbf{p}(\mathbf{I} - \mathbf{A}) = \mathbf{d}$$

Now solve by inverting **(I - A)**:

$$\mathbf{p} = (\mathbf{I} - \mathbf{A})^{-\mathbf{1}}\mathbf{d}$$

If the demand for O, E, and S is $\mathbf{d} = [100, 100, 100]^T$ we need to produce $\begin{bmatrix} O \\ E \\ S \end{bmatrix} = \begin{bmatrix} 220 \\ 276 \\ 235 \end{bmatrix}$

```
A = Matrix([
[0.25, 0.15, 0.1],
[0.4, 0.15, 0.2],
[0.15, 0.2, 0.2]
])

d= Matrix(3, 1, [100, 100, 100])
ii= eye(A.rows)
p= (ii - A).inv() * d
```

Exrcises 1.8.8

Show that if $\mathbf{A}$ is invertible then $A^T x = b$ has a unique solution.

```
x1, x2, x3, x4, x5, x6, x7, x8, x9= symbols('x1:10')
A= Matrix(3,3, [x1, x2, x3, x4, x5, x6, x7, x8, x9])

k1, k2, k3= symbols('k1:4')
b= Matrix(3, 1, [k1, k2, k3])

A_rref= A.transpose().solve(b).rref()
```

$$\mathbf{A_{rref}} = \left(\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad [0] \right) \tag{44}$$

Does it mean there is a unique solution?

2. Euclidean Space

Properties of vectors

Dot product, norm and distances. The dot product of two vectors $\mathbf{A}$ and $\mathbf{B}$ can be thought as the prodcut of their magnitude times the angle between them:

$$\mathbf{A} \cdot \mathbf{B} = \|\mathbf{A}\| \cdot \|\mathbf{B}\| cos(\theta)$$

Since $cos(\pi/2) = 0$, if the two vectors are orthogonal their dot product is 0. For example $\begin{bmatrix} 2 \\ 0 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 2 \end{bmatrix} = 0$

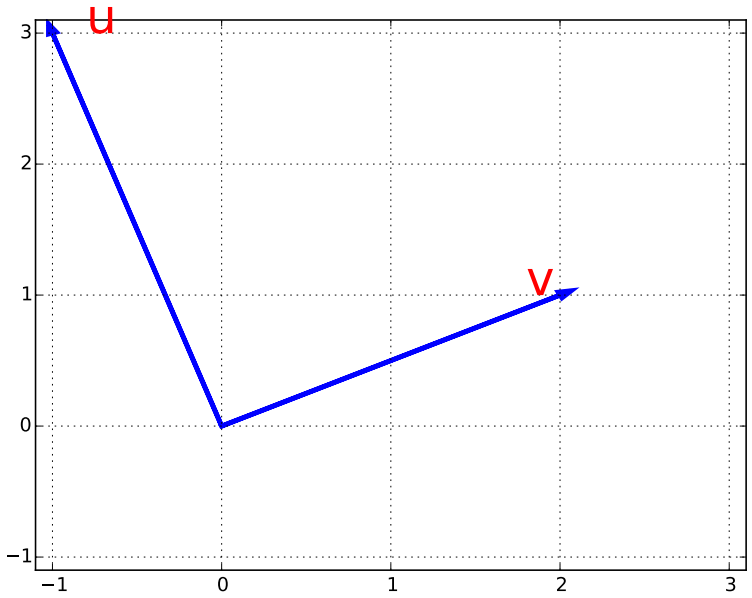
Exercise 2.1.1

Evaluate

```
import matplotlib.pyplot as plt

u= Matrix(2, 1, [-1, 3])
v= Matrix(2, 1, [2, 1])

plt.text(float(u[0])+0.2, float(u[1]), 'u', fontsize= 30, color= 'r')
plt.text(float(v[0])-0.2, float(v[1]), 'v', fontsize= 30, color= 'r')
plt.arrow(0, 0, float(u[0]), float(u[1]), lw= 3, color= 'b', head_width=0.05)
plt.arrow(0, 0, float(v[0]), float(v[1]), lw= 3, color= 'b', head_width=0.05)
plt.xlim((-1.1, 3.1))
plt.ylim((-1.1, 3.1))
plt.grid()
plt.savefig('figs/ex2_1_1.pdf')
plt.close()
```




```
u.dot(v) # 1
v.dot(u) # 1
u.dot(u) # 10
v.dot(v) # 5

u.norm()            # sqrt(10)
u.norm()**2         # 10
v.norm()            # sqrt(5)
```

```
v.norm()**2          # 5
(u + v).norm()**2    # 17
(u - v).norm()       # sqrt(13)
```

Exercise 2.1.3

Evaluate for $\mathbf{u} = \begin{bmatrix} -1 \\ 2 \\ 5 \\ -3 \end{bmatrix}$ and $\mathbf{v} = \begin{bmatrix} 2 \\ -3 \\ -1 \\ 5 \end{bmatrix}$

```
u= Matrix(4, 1, (-1, 2, 5, -3))
v= Matrix(4, 1, (2, -3, -1, 5))
```

Dot product:

```
v.dot(u)     # -28
u.dot(v)     # -28
v.dot(v)     # 39
u.dot(u)     # 39
```

Note that commutative property hold. NB `u * v` raises `ShapeError: Matrices size mismatch.`

Norm (*symbol*: $\|u\|$, $\|u+v\|$, $\|u\|^2$, etc)

```
u.norm()             # sqrt(39)
u.norm()**2          # 39
v.norm()             # sqrt(39)
v.norm()**2          # 39
(u + v).norm()**2    # 22
```

Distance $d(u, v) = \|u - v\|$

```
(u - v).norm() # sqrt(134)
(v - u).norm() # sqrt(134)
```

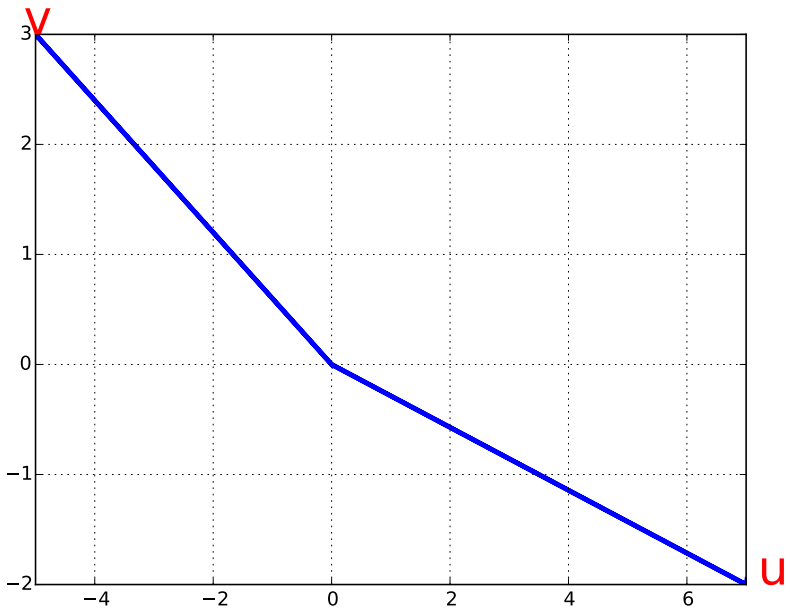
Exercise 2.1.6

Plot and compute for $\mathbf{u} = \begin{bmatrix} 7 \\ -2 \end{bmatrix}$ and $\mathbf{v} = \begin{bmatrix} -5 \\ 3 \end{bmatrix}$

$$\|\mathbf{u} + \mathbf{v}\| = \sqrt{5}$$

and

$$d(\mathbf{u}, \mathbf{v}) = \|\mathbf{u} - \mathbf{v}\| = 13$$




```
u= Matrix(2, 1, [7, -2])
v= Matrix(2, 1, [-5, 3])

plt.text(float(u[0])+0.2, float(u[1]), 'u', fontsize= 30, color= 'r')
plt.text(float(v[0])-0.2, float(v[1]), 'v', fontsize= 30, color= 'r')
plt.arrow(0, 0, float(u[0]), float(u[1]), lw= 3, color= 'b', head_width=0.05)
plt.arrow(0, 0, float(v[0]), float(v[1]), lw= 3, color= 'b', head_width=0.05)
plt.xlim((-5, 7))
plt.ylim((-2, 3))
plt.grid()
plt.savefig('figs/ex2_1_6.pdf')
plt.close()

(u + v).norm() # sqrt(5)
(u - v).norm() # 13
```

Further Properties of Vectors

Exercise 2.2.1

Find the angle between vectors. The angle is

$$\theta = \arccos\left(\frac{\mathbf{u}\cdot\mathbf{v}}{\|\mathbf{u}\|\cdot\|\mathbf{v}\|}\right) \tag{45}$$

In SymPy

```
def uv_angle(u, v):
    theta= acos(u.dot(v) / (u.norm() * v.norm()))
    return theta

u= Matrix([1, 1])
v= Matrix([0, 1])
uv_angle(u, v) # pi/4

# As degree:
uv_angle(u, v) * 180/pi # 45

u= Matrix([-1, 2, 3])
v= Matrix([sqrt(2), 1/sqrt(2), -1])
uv_angle(u, v) * 180/pi # ~115 deg
```

Exercise 2.2.4

Determine the vector of k so that the vectors are orthogonal to each other.

We want to set the angle between vector equal to 0, *i.e.* Eq. $45 = 0$.

```
k= symbols('k', real= True)
u= Matrix([-1, 5, k])
v= Matrix([-3, 2, 7])

eq= Eq(u.dot(v) / (u.norm() * v.norm()) , cos(pi/2))
sol= solve(eq)

# Check
u.subs(k, sol[0]).dot(v) == 0 # True
```

Exercise 2.2.6

A vector is normalized by dividing it by its norm. A normalized vector is called **unit vector**. In SymPy normlaize with:

```
def norm_u(u):
    return u / u.norm()
```

Determine the value of k so that $\hat{\mathbf{u}} = (1/\sqrt{2}\ 1/2\ k)^T$.

For $\mathbf{u}$ to be unit vector, its norm must equal 1. So $\|\mathbf{u}\| = \sqrt{|k|^2 + 0.75} = 1$. Solved for $k = \pm 1/2$.

```
u= Matrix([1/sqrt(2), 1/2, k])
eq= Eq(u.norm(), 1)
ksol= solve(eq) # [-0.5, 0.5]
```

Exercise 2.2.12: Support vector machine

Find the shortest distance between the vector $\mathbf{u}$ and the corresponding hyperplane. The hyperplane is defined as $\mathbf{v} \cdot \mathbf{x} + c = 0$ where $\mathbf{v}$ is a vector of coefficents for the variables in vector $\mathbf{x}$ and c is a constant term. The vector $\mathbf{u}$ might represent a data point in n-dimensions.

Remember that an equation in the usual form

$$ax + by + cz + k = 0$$

can be represented in matrix form as

$$ax + by + cz + k = \begin{bmatrix} a \\ b \\ c \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} + k = \mathbf{v} \cdot \mathbf{x} + k = 0$$

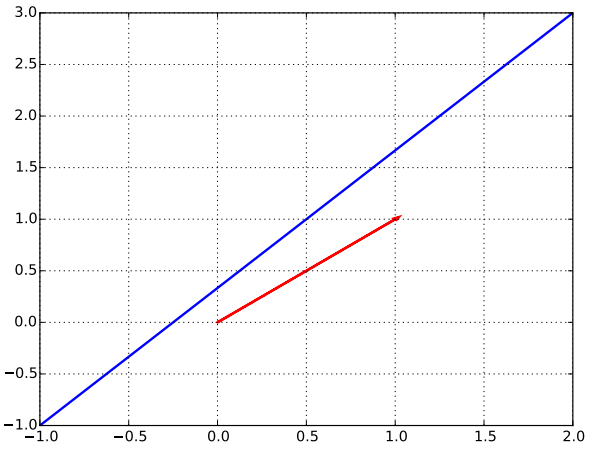
The shortest distance between $\mathbf{u}$ and hyerplane is given by

$$d_{shortest} = \frac{|\mathbf{u} \cdot \mathbf{v} + c|}{\|\mathbf{v}\|} \tag{46}$$

Given $\mathbf{u} = (1\ 1)^T$ and hyperplane $y = x + 1$, find the shortes distance. For this we need to extract the vector of coefficents $\mathbf{v}$ from $y = x + 1$, which for convenience can be re-arranged as $x - y + 1 = 0$. The coeffients $\mathbf{v}$ are $\begin{bmatrix} 1 & x \\ -1 & y \end{bmatrix}$.

Putting it all together:

$$d_{shortest} = \frac{|\begin{bmatrix} 1 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ -1 \end{bmatrix} + 1|}{\|\begin{bmatrix} 1 \\ -1 \end{bmatrix}\|} = \frac{\sqrt{2}}{2} \approx 0.707 \tag{47}$$



```
u= Matrix([1, 1])
hyp= Eq(x + 1, y)

# 1. Re-arrange hyp to have x + y + c= 0
hyp2= Eq(hyp.lhs - hyp.rhs)

# 2. Re-arrange in matrix form
cfdict= hyp2.lhs.as_expr().as_coefficients_dict()
v= Matrix([cfdict[x], cfdict[y]])
c= cfdict[numbers.One()]
d_shortest= abs(u.dot(v) + c) / v.norm()
```

Plotted with:

```
plt.plot([-1, 2], [hyp.lhs.subs(x, -2), hyp.lhs.subs(x, 2)], 'b-', lw= 2)
plt.arrow(0, 0, float(u[0]), float(u[1]), lw= 2, color= 'red')
plt.grid()
plt.xlim([-0.5, 1.5])
plt.ylim([-0.5, 1.5])
plt.savefig('figs/ex2_2_12_svm.pdf')
plt.close()
```

Exercise 2.2.12d

For $\mathbf{u} = \begin{bmatrix}1\\2\\3\\4\end{bmatrix}$ and $x + 2y + z + w = 10$. We have

$$d_{shortest} = \frac{|\mathbf{u}\cdot\mathbf{v}+c|}{\|\mathbf{v}\|} = \frac{|\begin{bmatrix}1\\2\\3\\4\end{bmatrix}\cdot\begin{bmatrix}1\\2\\1\\1\end{bmatrix} - 10|}{\|\begin{bmatrix}1\\2\\1\\1\end{bmatrix}\|} = \frac{2\sqrt{7}}{7} \approx 0.76$$

```
x,y,z,w = symbols('x y z w')
u = Matrix([1,2,3,4])
hyp= Eq(x + 2*y + z + w, 10)

hyp= hyp.lhs - hyp.rhs
v= [hyp.as_coefficients_dict()[t] for t in (x, y, z, w)]
v= Matrix(v)
c= hyp.as_coefficients_dict()[numbers.One()]

d_shortest= abs(u.dot(v) + c) / v.norm()
```

Linear Independence

Exercise 2.3.1

Determine which vectors in $\mathbb{R}^2$ are **linearly dependent**.
The linear combination of vectors $\mathbf{u}$ and $\mathbf{v}$ is

$$k\mathbf{u} + c\mathbf{v} = \mathbf{O} \tag{48}$$

Or more verbose:

$$k\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + c\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{49}$$

Where k and c are scalars (unknown coefficients). If 48 is satisfied for $k \neq 0$ or $c \neq 0$ then $\mathbf{u}$ and $\mathbf{v}$ are linearly dependent. Which means that $\mathbf{u}$ can be expressed as $\mathbf{v}$ times a scalar, or vice versa.
Expressed as a system of equations, 49 becomes:

$$\begin{aligned} kx_1 + cy_1 &= 0 \\ kx_2 + cy_2 &= 0 \end{aligned} \tag{50}$$

Now solve this system and check whether $k = 0$ and $c = 0$ are the only solutions.
For $\mathbf{u} = \begin{bmatrix} 6 \\ 10 \end{bmatrix}$ and $\mathbf{v} = \begin{bmatrix} -3 \\ -5 \end{bmatrix}$ we have:

$$k\begin{bmatrix} 6 \\ 10 \end{bmatrix} + c\begin{bmatrix} -3 \\ -5 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

re-arranged as a system

$$\begin{aligned} 6k - 3c &= 0 \\ 10k - 5c &= 0 \end{aligned}$$

With solutions $\left[\left\{k : \frac{c}{2}\right\}, \quad \{c : 2k\}\right]$ which hold for c and k different from 0.

```
u= Matrix([6, 10])
v= Matrix([-3, -5])
eq1= Eq(k * u[0] + c * v[0], 0)
eq2= Eq(k * u[1] + c * v[1], 0)

csol= solve([eq1, eq2], c)
ksol= solve([eq1, eq2], k)
```

Note that the conclusion of linear dependence can be reached by looking at the rank of the matrix $\mathbf{A} = \begin{bmatrix} \mathbf{u} & \mathbf{v} \end{bmatrix}$. In fact, $rank(\mathbf{A})$ is less than 2.

$$\mathbf{A} = \begin{bmatrix} \mathbf{u} & \mathbf{v} \end{bmatrix} = \begin{bmatrix} 6 & -3 \\ 10 & -5 \end{bmatrix}$$

with $rank(\mathbf{A}) = 1$. Note that trying to solve $\mathbf{A}$ results in SymPy error `ValueError: Matrix det == 0; not invertible.`

In addition, note that the RREF has only one leading row:

$$\mathbf{A_{rref}} = \left(\begin{bmatrix} 1 & -\frac{1}{2} & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad [0] \right)$$

```
A= v.col_insert(0, u)
A_rank= A.rank() # 1

try:
    A.solve(Matrix([0, 0])) # Matrix det == 0; not invertible.
except ValueError, e:
    print e

A_rref= A.col_insert(2, Matrix([0,0])).rref()
```

Unit vectors $\mathbf{e_1} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ and $\mathbf{e_2} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ are linearly independent. Note that $rank(\mathbf{A}) = N_{dimensions} = 2$ and the RREF is complete

$$\mathbf{A_{rref}} = \left(\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \quad [0, \quad 1] \right)$$

```
k, c= symbols('k c')
e1= Matrix([1, 0])
e2= Matrix([0, 1])

eq1= Eq(k * e1[0] + c * e1[1], 0)
eq2= Eq(k * e2[0] + c * e2[1], 0)

solve([eq1, eq2]) # {c: 0, k: 0}

# Also
A= e2.col_insert(0, e1)
A_rank= A.rank() # 2

A_rref= A.col_insert(2, Matrix([0,0])).rref()
```

Exercise 2.3.12 - *incomplete*

Determine the value of t in the **linearly independent** vectors

$$\mathbf{u} = \begin{bmatrix} t \\ 1 \\ 1 \end{bmatrix}, \mathbf{v} = \begin{bmatrix} -1 \\ t \\ 1 \end{bmatrix}, \mathbf{w} = \begin{bmatrix} 1 \\ 1 \\ t \end{bmatrix},$$

The RREF of the augmented matrix is:

$$\mathbf{A_{rref}} = \left(\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \quad \begin{bmatrix} 0, & 1, & 2 \end{bmatrix} \right)$$

which implies that either the coefficients $k_1 = k_2 = k_3 = 0$ or t

```
t, k1, k2, k3= symbols('t k1 k2 k3', real= True)

u= Matrix([ t, 1, 1]) # k1
v= Matrix([-1, t, 1]) # k2
w= Matrix([ 1, 1, t]) # k3

eq1= Eq(k1*t - k2   + k3,   0)
eq2= Eq(k1   + k2*t + k3,   0)
eq3= Eq(k1   + k2   + k3*t, 0)

A= w.col_insert(0, v).col_insert(0, u)
Au= Matrix([0,0,0]).col_insert(0, A)
A_rref= Au.rref()
```

Basis and Spanning Set

Example 2.17

Determine whether the vectors $\mathbf{u} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$ and $\mathbf{v} = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$ **span** $\mathbb{R}^2$. In other words, the question is whether with the axis $\mathbf{u}$ and $\mathbf{v}$ we can draw any vector in $\mathbb{R}^2$. In other words again, can any vector in $\mathbb{R}^2$ be defined as a ***linear combination*** of $\mathbf{u}$ and $\mathbf{v}$?

So let's take a generic vector in $\mathbb{R}^2$ $\mathbf{w} = \begin{bmatrix} a \\ b \end{bmatrix}$ and see if the equation

$$\mathbf{w} = k\mathbf{u} + c\mathbf{v} \tag{51}$$

can be solved to find the coefficients $\mathbf{k}$ and $\mathbf{c}$. If the answer is yes, than the generc vector $\mathbf{w}$ can be defined in terms of $\mathbf{u}$ and $\mathbf{v}$ given appropriate coefficients k and c respectively.

To solve 51 build the augmented matrix

$$\mathbf{A} = \begin{bmatrix} 1 & -1 & a \\ 2 & 1 & b \end{bmatrix}$$

,

put in RREF and see if the RREF is complete. In this case

$$\mathbf{A_{rref}} = \begin{bmatrix} 1 & 0 & \frac{a}{3} + \frac{b}{3} \\ 0 & 1 & -\frac{2a}{3} + \frac{b}{3} \end{bmatrix}$$

The coefficients $\mathbf{k}$ and $\mathbf{c}$ are given as the last column

$$k, c = \begin{bmatrix} \frac{a}{3} + \frac{b}{3} \\ -\frac{2a}{3} + \frac{b}{3} \end{bmatrix} \tag{52}$$

Here a and b can be thought as increments on the x and y axis.

For example, how do we express the vector $\mathbf{g} = \begin{bmatrix} 3 \\ 2 \end{bmatrix}$ in the **spanning set u** and **v**? By substituting in 52 3 for a and 2 for b we get the coefficients $k = 5/3$ and $c = -4/3$. Therefore

$$\mathbf{g} = \begin{bmatrix} 3 \\ 2 \end{bmatrix} = k\mathbf{u} + c\mathbf{v} \tag{53}$$

$$= \left[\tfrac{a}{3} + \tfrac{b}{3}\right] \mathbf{u} + \left[-\tfrac{2a}{3} + \tfrac{b}{3}\mathbf{v}\right] \tag{54}$$

$$= \left[\tfrac{3}{3} + \tfrac{2}{3}\right] \mathbf{u} + \left[-\tfrac{2 \cdot 3}{3} + \tfrac{2}{3}\mathbf{v}\right] \tag{55}$$

$$= 5/3\mathbf{u} + -4/3\mathbf{v} \tag{56}$$

$$= \begin{bmatrix} 3 \\ 2 \end{bmatrix} \tag{57}$$

```
a,b,c,k= symbols('a b c k')

u= Matrix([1, 2])
v= Matrix([-1, 1])
w= Matrix([a, b])

A= w.col_insert(0, v).col_insert(0, u)
A_rref= A.rref()

# Coefficients
k, c= A_rref[0].col(- 1)

# Or maybe easier using M.solve(const)
A= v.col_insert(0, u)
k, c= A.solve(w)

## Represent g in spanning set (u, v)
g= Matrix([3, 2])
k2= k.subs({a:g[0], b:g[1]})
c2= c.subs({a:g[0], b:g[1]})
g2= k2 * u + c2 * v
```

Exercise 2.4.1

Determine whether the following vectors span $\mathbb{R}^2$

```
# Generic vector
w= Matrix([a, b])

e1= Matrix([1, 0])
```

```
e2= Matrix([0, 1])

A= w.col_insert(0, e2).col_insert(0, e1)
```

For unit vectors $\mathbf{e_1}$ and $\mathbf{e_2}$ the answer is yes with coefficient a and b from RREF

$$\mathbf{A_{rref}} = \begin{bmatrix} 1 & 0 & a \\ 0 & 1 & b \end{bmatrix}$$

These vectors do not form a spanning set as a generic vector $\mathbf{w}$ cannot be formed by linear combination. The augmented matrix cannot be in RREF:

$$\mathbf{A_{rref}} = \begin{bmatrix} 1 & -\frac{1}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Trying to solve the augmented matrix `ValueError: Matrix det == 0; not invertible.`

$$\mathbf{A} = \begin{bmatrix} 2 & -1 & a \\ 2 & -1 & b \end{bmatrix}$$

Results in

```
u= Matrix([2, 2])
v= Matrix([-1, -1])
A= w.col_insert(0, v).col_insert(0, u)
A.rref() # Not in RREF

# Indeed:
A= v.col_insert(0, u)
A.solve(w) # ValueError: Matrix det == 0; not invertible.
```

Exercise 2.4.2

Determine whether the following vectors span $\mathbb{R}^3$

```
u= Matrix([1, 2, 1])
v= Matrix([2, 4, 0])
w= Matrix([-2, -2, 3])
```

So, let's define a generic vector in $\mathbb{R}^3$

```
a,b,c= symbols('a b c')
g= Matrix([a, b, c])
```

and define $\mathbf{g}$ as a linear combination of $\mathbf{u}$, $\mathbf{v}$, $\mathbf{w}$ *i.e.*

$$\mathbf{g} = k\mathbf{u} + j\mathbf{v} + l\mathbf{w}$$

Can we solve these linear system to find the unknow k, j, l?

To answer, prepare the augmented matrix (not really necessary to prepare it though). Note the use of `numpy.hstack` to column-bind vectors. Now try to solve it

```
A= Matrix(numpy.hstack((u, v, w, g)))
sol= A[:, :-1].solve(A[:, -1])
```

The syntax to slice the matrix comes from NumPy: `A[:, :-1]` means get all rows with ':' (1st index) and all columns excluding the last one with ':-1'.

Anyway, the solution exists and is

$$k, j, l = \begin{bmatrix} 3a - \frac{3b}{2} + c \\ -2a + \frac{5b}{4} - \frac{c}{2} \\ -a + \frac{b}{2} \end{bmatrix}$$

so **u, v, w** are spanning set of $\mathbb{R}^3$

Exercise 2.4.3

Determine whether the following vectors form a ***basis*** for $\mathbb{R}^2$

```
u= Matrix([1, 2])
v= Matrix([0, 1])
```

A ***basis*** is a spanning set whose vectors are also linearly independent. This means that a basis is the minimal set of vectors necessary to describe the $\mathbb{R}^n$ space. Note that if the vectors span $\mathbb{R}^n$ than they are linearly independent and *vice versa*. So we need to check only one of the two conditions.

In the case above we have linearl independence since:

$$A = \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 2 \end{bmatrix}$$

and a solution can be found:

$$a, b = \begin{bmatrix} -2a + b \\ a \end{bmatrix}$$

```
A= Matrix(numpy.array((u, v)))
sol= A.solve(Matrix([a, b]))
A.rank() == A.rows # True (2)
A.det() != 0 # True
A.rref()
```

Note also that $rank(A) = 2$, *i.e.* **A** is full rank and the RREF is complete $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

In this case, there is no linear independence and the vectors are not a basis:

```
u= Matrix([-2, -4])
v= Matrix([1, 2])
A= Matrix(numpy.array((u, v)))
sol= A.solve(Matrix([a, b]))  ## ValueError: Matrix det == 0; not invertible.
A.rank() == A.rows # False
A.det() == 0 # True
```

Exercise 2.4.4

Do these vector form a basis for $\mathbb{R}^4$?

```
u= Matrix([1, 1, 1, 1])
v= Matrix([1, 1, 0, -1])
w= Matrix([-1, -2, -3, 4])
x= Matrix([2, 2, 2, 2])
A= Matrix(numpy.hstack([u, v, w, x]))
```

No since they are not linearly independent. In fact u and x are multiple of each other.

Exercise 2.4.5

Is $\mathbf{b}$ in the space spanned by the columns of matrix $\mathbf{A}$?

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 0 \\ 2 & 5 & 0 \\ 3 & 4 & 9 \end{bmatrix}; \mathbf{b} = \begin{bmatrix} 1 \\ 3 \\ 4 \end{bmatrix} \tag{58}$$

To answer we want to know whether the columns of $\mathbf{A}$ can be linearly combined to produce $\mathbf{b}$. Can we found a vector $\mathbf{k}$ of coefficients for the columns in $\mathbf{A}$ that produce $\mathbf{b}$? So we want ot see if we can solve $\mathbf{Ak} = \mathbf{b}$ *w.r.t.* $\mathbf{k}$. *I.e.* let's try to solve $\mathbf{k} = \mathbf{A}^{-1}\mathbf{b}$. We can solve it for $\mathbf{k} = \begin{bmatrix} \frac{2}{3} \\ \frac{1}{3} \\ \frac{2}{27} \end{bmatrix}$.

We can also verify that $\mathbf{Ak} = \mathbf{b}$.

```
A= Matrix([[1, 1, 0], [2, 5, 0], [3, 4, 9]])
b= Matrix([1, 3, 4])

k= A.inv().dot(b)
# The same:
k= A.solve(b)

# Check back:
b == Matrix(A.dot(k))
```

What about this case:

```
A= Matrix([[1, 2, 3],
           [4, 5, 6],
           [7, 8, 9]])
b= Matrix([1, 3, 4])
A.inv() ## ValueError: Matrix det == 0; not invertible.
A.solve(b) ## The same
```

`ValueError: Matrix det == 0; not invertible`, in fact the columns of $\mathbf{A}$ are not linearly independent we cannot find a vector of coefficients $\mathbf{k}$ to represent $\mathbf{b}$.

Exercise 2.4.6

Are these vectors a basis for $\mathbb{R}^3$?

```
u= Matrix([5, 0, 0])
v= Matrix([0, 6, 0])
w= Matrix([0, 0, 7])
A= Matrix(numpy.hstack([u, v, w]))
A.solve(Matrix([a, b, c])) ## Solvable -> lin. indep. -> basis
```

What about:

```
alpha, beta, lamda= symbols('alpha beta lamda', real= True, nonzero= True)
u= Matrix([alpha, 0, 0])
v= Matrix([0, beta, 0])
w= Matrix([0, 0, lamda])
A= Matrix(numpy.hstack([u, v, w]))
A.solve(Matrix([a, b, c])) ## Solvable -> lin. indep. -> basis

# Also:
A.inv() # Ok
A.rank() == 3 # True
A.det() != 0 # True
```

3. General vector spaces

3.3 Linear independence and basis

Exercise 3.3.1

Are the **A** and **B** matrices linearly independent?

They are independent if there is no scalar that can multiply one matrix to return the other. So if there is a solution for x to $Ax = B$, **A** and **B** are linearly dependent.

For

$$\mathbf{A} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}; \mathbf{B} = \begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix}; \mathbf{A}x = \mathbf{B} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} x = \begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix}$$

$x = 2$ can solve the equation and transform **A** into **B** hence the two matrices are dependent.

```
A= Matrix(2, 2, [1,1,1,1])
B= Matrix(2, 2, [2,2,2,2])
solve(Eq(x*A, B), x) # {x: 2} dependent.
```

The equation $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} x = \begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix}$ has no solution for x, so **A** and **B** are independent.

```
A= Matrix(2, 2, [1, 2,
                 3, 4])
B= Matrix(2, 2, [2, 2,
                 2, 2])
Eq(A*x, B)
```

Example 3.15

Given three vectors in the space of continuous functions:

$$\mathbf{u} = cos(x) \quad \mathbf{v} = sin(x) \quad \mathbf{w} = 2$$

We show they are **linearly independent** by solving:

$$k_1\mathbf{u} + k_2\mathbf{v} + k_1\mathbf{w} = 0$$

For k_1, k_2, k_3. If the only solution is for $k_1 = k_2 = k_3 = 0$, the functions are linearly independent.

To find solutions for k_1, k_2, k_3 we need a system of three equations. In each equation x is replaced by a convenient value which makes the arithmetic easier. In this case we could use $x = 0, x = \pi, x = \pi/2$. Here we replace x with a generic constant c_1, c_2, c_3 since the dirty job of solving the system is left to SymPy .

$$\begin{bmatrix} k_1\mathbf{u} + k_2\mathbf{v} + k_3\mathbf{w} \\ k_1\mathbf{u} + k_2\mathbf{v} + k_3\mathbf{w} \\ k_1\mathbf{u} + k_2\mathbf{v} + k_3\mathbf{w} \end{bmatrix} = \begin{bmatrix} k_1\cos(c_1) + k_2\sin(c_1) + 2k_3 \\ k_1\cos(c_2) + k_2\sin(c_2) + 2k_3 \\ k_1\cos(c_3) + k_2\sin(c_3) + 2k_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

```
x,k1,k2,k3= symbols('x k1:4', real= True)
u= cos(x)
v= sin(x)
w= 2

eq= Eq(k1*u + k2*v + k3*w, 0)
c1, c2, c3= symbols('c1:4', real= True)
eq1= eq.subs(x, c1)
eq2= eq.subs(x, c2)
eq3= eq.subs(x, c3)
solve([eq1, eq2, eq3], [k1, k2, k3])
```

Given

$$\mathbf{f} = \cos^2(x), \quad \mathbf{g} = \sin^2(x), \quad \mathbf{h} = 5$$

The linear combination

$$k_1\mathbf{f} + k_2\mathbf{g} + k_3\mathbf{h} = k_1\cos^2(x) + k_2\sin^2(x) + 5k_3 = 0$$

is solved with the system

$$\begin{bmatrix} k_1\cos^2(c_1) + k_2\sin^2(c_1) + 5k_3 = 0 \\ k_1\cos^2(c_2) + k_2\sin^2(c_2) + 5k_3 = 0 \\ k_1\cos^2(c_3) + k_2\sin^2(c_3) + 5k_3 = 0 \end{bmatrix}$$

where $\{k_1 : -5k_3, \quad k_2 : -5k_3\}$. Since k_1, k_2, k_3 are not all equal to zero, the functions are linearly dependent.

```
f= cos(x)**2
g= sin(x)**2
h= 5

def testLinearIndep(f, g, h):
    ## Linear combination
    k1, k2, k3= symbols('k1:4')
    eq= Eq(k1*f + k2*g + k3*h, 0)

    ## Linear system:
    c1, c2, c3= symbols('c1:4')
    eq1= eq.subs(x, c1)
    eq2= eq.subs(x, c2)
    eq3= eq.subs(x, c3)

    ## Solutions to linear system
    M= Matrix(3, 1, [eq1, eq2, eq3])
    sols= solve(M, [k1, k2, k3])
    return sols
```

It can be attcked using Wronskian determinant $\mathbf{W}$. If $\mathbf{W} \neq 0$ than the functions are linearly independent. However $\mathbf{W} = 0$ does not necessarily mean dependence.

```
k1, k2, k3= symbols('k1:4')
f= cos(x)**2
g= sin(x)**2
h= 5

def wronskian(f, g, h):
    W= Matrix([[f, g, h],
            [diff(f, x), diff(g, x), diff(h, x)],
            [diff(f, x, 2), diff(g, x, 2), diff(h, x, 2)]])
    return W.det()

wronskian(f, g, h) # == 0 => *Suggests* lin. dep.
```

Exercise 3.3.3c

```
f= 1
g= x
h= x**2
```

```
testLinearIndep(f, g, h) #  {k1: 0, k2: 0, k3: 0} => Lin. Indep.
wronskian(f, g, h) # == 2 -> Lin. Indep.
```

Exercise 3.3.3d

In this case, the Wronskian appears to be non-zero:

$$\mathbf{W} = -3\sin^2(x)\sin(2x)\cos(x) - 6\sin(x)\cos^2(x)\cos(2x) + 3\sin(2x)\cos^3(x)$$

Although it seems to be very close to 0 for any value of x.

```
f= sin(2*x)
g= sin(x) * cos(x)
h= cos(x)
testLinearIndep(f, g, h) # {k1:-k2/2, k3: 0} => Lin dep.
w= wronskian(f, g, h) # \neq 0. Why?

## Wronskian close to but not zero:
max([float(w.subs(x, c/100)) for c in range(-200*pi, 200*pi)]) # 6.66e-16
```

Exercise 3.3.3e

```
f= exp(x) * sin(2*x)
g= exp(x) * sin(x)* cos(x)
h= exp(x) * cos(x)
testLinearIndep(f, g, h) # {k1:-k2/2, k3: 0} => Lin dep.
w= wronskian(f, g, h)
max([float(w.subs(x, c/100)) for c in range(-200*pi, 200*pi)]) # 3.818e-08
```

Exercise 3.3.3f

```
f= 1
g= exp(x)
h= exp(-x)
testLinearIndep(f, g, h) # k1 = k2 = k3 = 0 # Lin. Indep.
wronskian(f, g, h)       # wronskian= 2
```

Exercise 3.3.3g

```
f= exp(x)
g= exp(2*x)
h= exp(3*x)
testLinearIndep(f, g, h) # k1 = k2 = k3 = 0 # Lin. Indep.
wronskian(f, g, h)       # wronskian == 2*exp(6*x) != 0
```

Exercise 3.3.4

Same as exercise

```
f= sin(x)
g= sin(3*x)
h= sin(5*x)
testLinearIndep(f, g, h) # k1 = k2 = k3 = 0 # Lin. Indep.
w= wronskian(f, g, h)  # !=0
```

Exercise 3.3.4

Show that the following matrices are a **basis** for M_{22}, the vector space of 2 by 2 matrices.

$$\mathbf{A} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, \quad \mathbf{C} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}, \quad \mathbf{D} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$$

The following conditions must satisfied:

1. **Span**: Can any vector in M_{22} be generated with **A, B, C, D**?
2. **Linear independence**: Are **A, B, C, D** linearly independent?

To see if **A, B, C, D span** M_{22} we generate a generic matrix $\mathbf{M} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ and then we see if a linear combination of **A, B, C, D** can produce **M**:

$$k_1\mathbf{A} + k_2\mathbf{B} + k_3\mathbf{C} + k_4\mathbf{D} = \mathbf{M}$$

Yes, we can combine **A, B, C, D** and produce **M** by setting $\{a : k_1, \quad b : k_2, \quad c : k_3, \quad d : k_4\}$
Are **A, B, C, D** also linearly independent? In other words is the linear combination

$$k_1\mathbf{A} + k_2\mathbf{B} + k_3\mathbf{C} + k_4\mathbf{D} = \mathbf{O}$$

satisifed for values of ks other than $k_1 = k_2 = k_3 = k_4 = 0$? No, as show in the code.

```
A= Matrix(2, 2, [1, 0, 0, 0])
B= Matrix(2, 2, [0, 1, 0, 0])
C= Matrix(2, 2, [0, 0, 1, 0])
D= Matrix(2, 2, [0, 0, 0, 1])

## Span
a,b,c,d= symbols('a b c d')
k1, k2, k3, k4= symbols('k1:5')
M= Matrix(2, 2, [a, b, c, d])
solve(Eq(k1*A + k2*B + k3*C + k4*D, M)) # {a: k1, b: k2, c: k3, d: k4}

## Linear independence
eq= Eq(k1*A + k2*B + k3*C + k4*D, zeros(2,2))
solve(eq) # {k1: 0, k2: 0, k3: 0, k4: 0}
```

Exercise 3.3.6

Check whether the three polynomials p, q, r form a basis for the vector space of the polynomials of degree ≤ 2.

1. **Span**: Take a general polynomial y of degree 2 and see if the linear combination of the p, q, r can produce y
2. **Linear independence**: Check whether p, q, r are linearly independent.

p, q, r do span P^2 by setting $\{k_1 : a + b + c, \quad k_2 : 2a + b, \quad k_3 : a\}$

p, q, r are linearly independent since the only solution to $k_1 + k_2 (t-1) + k_3 (t-1)^2 = 0$ is for $\{k_1 : 0, \quad k_2 : 0, \quad k_3 : 0\}$.

```
t,a,b,c= symbols('t a b c')
k1, k2, k3= symbols('k1:4')
p= 1
q= t - 1
r= (t - 1)**2

## Span
y= a*t**2 + b*t + c
eq= Eq(k1*p + k2*q + k3*r, y)
solve(eq, [k1, k2, k3]) # {k1: a + b + c, k2: 2a + b, k3: a}

## Linear independence
eq= Eq(k1*p + k2*q + k3*r, 0)
solve(eq, [k1, k2, k3]) # {k1: 0, k2: 0, k3: 0}
```

With $\mathbf{p}, \mathbf{q}, \mathbf{r}$ we can define any vector in P^2. For example, how can we combine $\mathbf{p}, \mathbf{q}, \mathbf{r}$ to obtain $\mathbf{p_1} = t^2 + 1$? *I.e.*, what coefficients k_1, k_2, k_3 do we need to assign to $\mathbf{p}, \mathbf{q}, \mathbf{r}$? We need to solve the equation

$$k_1\mathbf{p} + k_2\mathbf{q} + k_3\mathbf{r} = t^2 + 1$$

for k_1, k_2, k_3. It is solved with $\{k_1 : 2, \quad k_2 : 2, \quad k_3 : 1\}$.

We can check this by substituting k_1, k_2, k_3 in $k_1\mathbf{p} + k_2\mathbf{q} + 5\mathbf{r}$ and see that we obtain $t^2 + 1$.

```
p1= t**2 + 1
eq= Eq(k1*p + k2*q + k3*r, p1)
sols= solve(eq, [k1, k2, k3]) ## {k1: 2, k2: 2, k3: 1}

## Check sols return p1
pp= k1*p + k2*q + k3*r
pp.subs(sols).simplify()      ## => t**2 + 1
pp.subs(sols).simplify() - p1 ## => 0
```

Exercise 3.3.7

$\mathbf{p} = 1, \quad \mathbf{q} = t^2 - 2t, \quad \mathbf{r} = 5\,(t-1)^2$ do not form a basis for P^2 since the three polynomials are not linearly independent. In fact, $k_1\mathbf{p} + k_2\mathbf{q} + 5\mathbf{r} = 0$ is solved for $\{k_1 : -5k_3, \quad k_2 : -5k_3\}$

```
p= 1
q= t**2 - 2*t
r= 5*(t-1)**2

eq= Eq(k1*p + k2*q + k3*r, 0)
sols= solve(eq, [k1, k2, k3])
```

3.6 Linear systems revisted

It is useful to keep in mind that all these statements are equivalent. If one is true all the others are true. Given a n by n (square) matrix $\mathbf{A}$ we have:

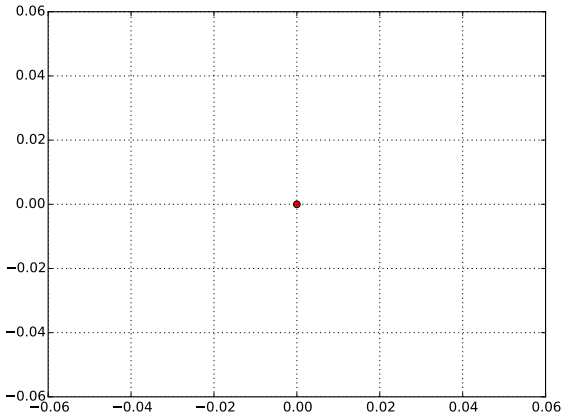
- $\mathbf{A}$ is invertible.
- $\mathbf{A}$ is full rank, *i.e.* $rank(\mathbf{A}) = nrows(\mathbf{A}) = n$.
- Rows in $\mathbf{A}$ are linearly independent.
- Columns in $\mathbf{A}$ are linearly independent.
- $nullity(\mathbf{A}) = 0$.
- The *nullspace* of $\mathbf{A}$ contains only the zero vector.

Given the system $\mathbf{Ax} = \mathbf{O}$, the ***nullspace*** is the set of vectors that satisfies the system. The *dimension* of the nullspace is called ***nullity***. Consequently for a n by n matrix we have:

$$rank(\mathbf{A}) + nullspace(\mathbf{A}) = n$$

Exercise 3.6.1

Nullspace of 2 by 2 unit matrix $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ is the zero vector since this is the only solution to $\mathbf{Av} = \mathbf{O}$.

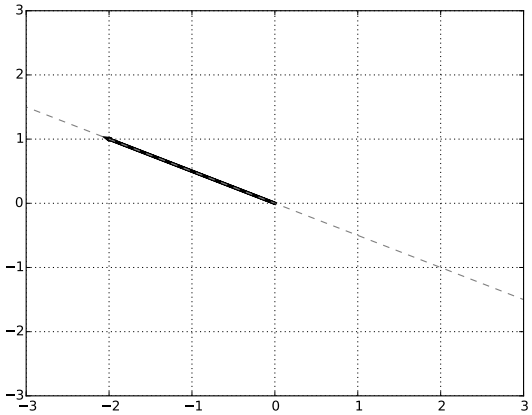



```
x,y= symbols('x,y')
v= Matrix([x, y])
A= eye(2)

sols= solve(Eq(A*v, Matrix([0, 0])))[0] ## [{x: 0, y: 0}]
A.nullspace() ## Empty.

plt.plot(sols[x], sols[y], 'ro')
plt.grid()
plt.savefig('figs/ex3_6_1.pdf')
```

For $\mathbf{A} = \begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix}$ we need to solve $\begin{bmatrix} x+2y \\ 2x+4y \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$. Solution is $\{x : -2y\}$ or $\begin{bmatrix} -2 \\ 1 \end{bmatrix}$. Graphically, the nullspace is the vector passing by $\begin{bmatrix} -2 \\ 1 \end{bmatrix}$ as shown here below. In fact any point sitting on this line will make $\mathbf{Av} = \mathbf{O}$. *E.g.* with $\mathbf{v} = [2 \ -1]^T$ we have $\begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix} \begin{bmatrix} 2 \\ -1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$



```
A= Matrix(2, 2, [1, 2, 2, 4])

sols= solve(Eq(A*v, zeros(A.rows, 1)))[0]
## Or simply
nsp= A.nullspace()[0]

plt.plot((0, nsp[0]*3), (0, nsp[1]*3))

plt.plot((-nsp[0]*3, nsp[0]*3), (-nsp[1]*3, nsp[1]*3), color= 'grey',
    linestyle= '--')
plt.arrow(0, 0, float(nsp[0]), float(nsp[1]), 'g', lw= 3)
plt.xlim(-3, 3)
plt.ylim(-3, 3)
plt.grid()
plt.savefig('figs/ex3_6_1b.pdf')
```

For $\begin{bmatrix}1 & 0\\ 1 & 2\\ 6 & 10\end{bmatrix}$ we need to solve $\begin{bmatrix}x\\ x+2y\\ 6x+10y\end{bmatrix} = \begin{bmatrix}0\\ 0\\ 0\end{bmatrix}$. This is for $\{x : 0, \quad y : 0\}$.

```
A= Matrix(3, 2, [1, 0, 1, 2, 6, 10])

sols= solve(Eq(A*v, zeros(A.rows, 1)))[0]
## Or
A.nullspace() # -> []
```

For $\begin{bmatrix}1 & 2\\ 3 & 4\\ 5 & 6\end{bmatrix}$ we have $\begin{bmatrix}x+2y\\ 3x+4y\\ 5x+6y\end{bmatrix} = \begin{bmatrix}0\\ 0\\ 0\end{bmatrix}$ solved only for $x = 0; y = 0$

```
A= Matrix(3, 2, [1, 2, 3, 4, 5, 6])
sols= solve(Eq(A*v, zeros(A.rows, 1)))[0]
```

Exercise 3.6.3

Determine nullspace and nullity for $\mathbf{A} = \begin{bmatrix}1 & -2 & -3\\ 4 & -5 & -6\\ 7 & -8 & -9\end{bmatrix}$.

The solution to $\begin{bmatrix}x-2y-3z\\ 4x-5y-6z\\ 7x-8y-9z\end{bmatrix} = \begin{bmatrix}0\\ 0\\ 0\end{bmatrix}$.

is $\{x : -z, \quad y : -2z\}$. Rearranged in vector form we have $\begin{bmatrix}x\\ y\\ z\end{bmatrix} = \begin{bmatrix}-z\\ -2z\\ z\end{bmatrix} = z(\begin{bmatrix}-1\\ -2\\ 1\end{bmatrix})$ for z being any real. Note therefore that the nullspace is the solution to the linear system.

The nullity is the dimension of the nullspace. In this case only one vector makes the nullspace, hence $nullity(\mathbf{A}) = 1$

```
A= Matrix([[1, -2, -3],
           [4, -5, -6],
           [7, -8, -9]])
x,y,z= symbols('x,y,z')
v= Matrix([x, y, z])
sols= solve(Eq(A*v, zeros(A.rows, 1)), x)[0]
v.subs(sols)
# Also
nsp= A.nullspace() # -> [[-1, -2, 1]]
nlty= len(nsp)     # -> 1
```

For $\mathbf{A} = \begin{bmatrix} 2 & -2 & -2 \\ 4 & -4 & -4 \\ 8 & -8 & -8 \end{bmatrix}$ the linear systen to solve is $\begin{bmatrix} 2x-2y-2z \\ 4x-4y-4z \\ 8x-8y-8z \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$.
Solved for $\{x : y+z\}$. Substituing the results in $\mathbf{v}$ we have the nullspace:

$$\mathbf{v_{sol}} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} y+z \\ y \\ z \end{bmatrix} = y(\begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}) + z(\begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix})$$

The nullspace is made of two vectors $y(\begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}) + z(\begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix})$ hence the nullity is 2.
Note that $rank(\mathbf{A}) = 1$ consistent with $nullity(\mathbf{A}) + rank(\mathbf{A}) = n$

```
A= Matrix([[2, -2, -2],
           [4, -4, -4],
           [8, -8, -8]])
sols= solve(Eq(A*v, zeros(A.rows, 1)), x)
v.subs(sols)
# Equivalent to
y*Matrix([1,1,0]) + z*Matrix([1, 0, 1])
# Same as
nsp= A.nullspace()
nlty= len(nsp)
```

This system $\begin{bmatrix} 2x+9y-3z \\ 5x+6y-z \\ 9x+8y-9z \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$ has solution only for $[\{x : 0, \quad y : 0, \quad z : 0\}]$
hence the nullspace is the zero vector and the nullity is 0. In fact, the dimension of the zero vector is zero.

```
A= Matrix([[2, 9, -3],
           [5, 6, -1],
           [9, 8, -9]])
sols= solve(Eq(A*v, zeros(A.rows, 1)))
nsp= A.nullspace() # -> []
nlty= len(nsp)     # -> 0
```

Same as above for $\mathbf{A} = \begin{bmatrix} -3 & 1 & -1 \\ 2 & 5 & -7 \\ 4 & 8 & -4 \end{bmatrix}$

```
A= Matrix([[-3, 1, -1],
           [2, 5, -7],
           [4, 8, -4]])
sols= solve(Eq(A*v, zeros(A.rows, 1)))
```

Exercise 3.6.4

Determine the nullspace and nullity of $\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 & 12 & 13 & 14 \\ 15 & 16 & 17 & 18 & 19 & 20 & 21 \\ 22 & 23 & 24 & 25 & 26 & 27 & 28 \\ 29 & 30 & 31 & 32 & 33 & 34 & 35 \end{bmatrix}$

Nullspace is $\left[\begin{bmatrix} 1 \\ -2 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 2 \\ -3 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 3 \\ -4 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 4 \\ -5 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 5 \\ -6 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}\right]$ contaning five vectors, five is the nullity of $\mathbf{A}$.

```
x1,x2,x3,x4,x5,x6,x7= symbols('x1:8')
A= Matrix(5, 7, range(1, 36))
sols= solve(Eq(A*v, zeros(A.rows, 1)))
v.subs(sols) # Nullspace. You could read out the nullspace vectors from here
nsp= A.nullspace()
nly= len(A) # 5
```

Exercise 3.6.5

Determine bases for row, column and null space for $\mathbf{A} = \begin{bmatrix} 1 & -4 & -9 \\ 2 & 5 & -7 \end{bmatrix}$.

In other words, we need to find a set of **linearly independent** vectors such that every vector in the given vector space (e.g. row space) is **linearly dependent** of these vectors. Bases or coordinates are the basic building blocks to represent a vector space.

To find a basis for the rows of a matrix (i.e. for this vector space) we need to put the matrix un RREF. The rows of the RREF matrix are linearly independent.

For the rows in $\mathbf{A}$, $\mathbf{A_{rref}} = \begin{bmatrix} 1 & 0 & -\frac{73}{13} \\ 0 & 1 & \frac{11}{13} \end{bmatrix} = \begin{bmatrix} 13 & 0 & -73 \\ 0 & 13 & 11 \end{bmatrix}$,

the rows of $\mathbf{A_{rref}}$ are the bases for the row space of $\mathbf{A}$.

The basis for the column vector can be obtained by looking at the leading 1s in A_{rref}, these are $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$. *The same conclusion is reached by transposing* $\boldsymbol{A}$ *and repeating the reasoning for the row space. Like* `A.transpose().rref()`

The basis for the nullspace is given by a nullspace itself `A.nullspace()` which is $\begin{bmatrix} \frac{73}{13} \\ -\frac{11}{13} \\ 1 \end{bmatrix}$. In this case the nullity is 1.

```
A= Matrix(2, 3, [1, -4, -9, 2, 5, -7])
B= Matrix(3, 2, [1, 3, 2, 5, -14, -37])
C= Matrix([[1, 3, -9, 5],
           [2, 6, 7, 1],
           [1, 3, -8, 1]])
rbasis= rref= A.rref()
cbasis= rref.A.transpose().rref()
nspbasis= A.nullspace()
```

Exercise 3.6.7

Determine whether the following system have *infinite*, *unique*, or *no* solutions. Without actually solving the system we can get the type of solution by looking at the rank of coefficient matrix and augmented matrix:

- **Infinite solutions** $rank(\mathbf{A}) = rank(\mathbf{A}_{\mathbf{augmented}}) < nrow(\mathbf{A})$
- **Unique solution** $rank(\mathbf{A}) = rank(\mathbf{A}_{\mathbf{augmented}}) = nrow(\mathbf{A})$
- **No solution** $rank(\mathbf{A}) \neq rank(\mathbf{A}_{\mathbf{augmented}})$

```
def numSolutions(A, b):
    rankA= A.rank()
    rankA_aug= Matrix(numpy.hstack([A, b])).rank()
    n= A.rows
    print 'nrow(A)= %s' %(n)
    print 'rank(A)= %s' %(rankA)
    print 'rank(A_aug)= %s' %(rankA_aug)
    if rankA == rankA_aug == n:
        return 'unique solution'
    elif rankA == rankA_aug < n:
        return 'oo solutions'
    elif rankA != rankA_aug:
        return 'no solutions'
    else:
        return None

A= Matrix([[1,2,3],
           [4,5,6],
           [7,8,9]])
b= Matrix([1,2,4])
numSolutions(A, b) # No solutions
```

Consider the system

$$A = \begin{bmatrix} 2x+ & 5y & -3z & -7w \\ 1x+ & 1y & -4z & -8w \\ 3x+ & 4y & +0z & +1w \\ 5x+ & 21y & -1z & +3w \end{bmatrix} \quad b = \begin{bmatrix} 0 \\ 9 \\ 6 \\ 2 \end{bmatrix}$$

It has a unique solution has showed here using ranks and verified by actually solving the system. Solution are

$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} \frac{332}{117} \\ -\frac{545}{351} \\ -\frac{1091}{117} \\ \frac{1298}{351} \end{bmatrix}$$

```
A= Matrix([[2, 5, -3, -7],
           [1, 1, -4, -8],
           [3, 4, 0, 1],
           [5, 21, -1, 3]])
b= Matrix([0, 9, 6, 2])
numSolutions(A, b) ## Unique solution
nrow(A)= 4
rank(A)= 4
rank(A_aug)= 4

## Solve
xyzw_sols= A.solve(b)
```

If we add a redundant row, the system is overdetermined and has infinite solutions:

$$\mathbf{A} = \begin{bmatrix} 2 & 5 & -3 & -7 \\ 1 & 1 & -4 & -8 \\ 3 & 4 & 0 & 1 \\ 5 & 21 & -1 & 3 \\ 5 & 21 & -1 & 3 \end{bmatrix} \mathbf{b} = \begin{bmatrix} 0 \\ 9 \\ 6 \\ 2 \\ 2 \end{bmatrix}$$

Note also the exception thrown by `Matrix.solve` `ValueError: For over-determined system, M, having more rows than columns, try M.solve_least_squares(rhs).`. Using least squares we get the unique solution.

```
A= Matrix([[2, 5, -3, -7],
           [1, 1, -4, -8],
           [3, 4, 0, 1],
           [5, 21, -1, 3],
           [5, 21, -1, 3]])
b= Matrix([0, 9, 6, 2, 2])
numSolutions(A, b) ## 'oo solutions'
nrow(A)= 5
rank(A)= 4
rank(A_aug)= 4

A.solve(b) ## ValueError: For over-determined system

## Use linear least squares:
def linlsq(A, b):
    betas= (A.transpose() * A).inv() * A.transpose() * b
```

```
    return betas

## Same as sympy function:
linlsq(A, b)             ## [332/117], [-545/351], [-1091/117], [1298/351]
A.solve_least_squares(b) ##
```

Exercise 3.6.8a

Test if the vectors **u** are in *nullspace* of the corresponding matrices:

- Get the nullspace of **A** by solving $\mathbf{Av} = \mathbf{O}$
- Test if **u** is a linear combination of the nullspace

To find the nullspace either use SymPy `Matrix.nullspace()` or solve the system $\mathbf{Av} = \mathbf{O}$ where **v** is a general vector of unknowns $[x_1\ x_2\ ...\ x_n]$. Then, to extract the nullspace from the solution(s), substitute the solutions in **v** and extract the coeffcients. For matrix

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & -1 \\ 2 & -1 & 0 \\ 5 & 2 & -3 \end{bmatrix}$$

solve:

$$\begin{bmatrix} x_1 + x_2 - x_3 \\ 2x_1 - x_2 \\ 5x_1 + 2x_2 - 3x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

The solution are substituted in **v** and the coefficients extracted:

$$sols = \left\{ x_1 : \frac{x_3}{3}, \quad x_2 : \frac{2x_3}{3} \right\} = \begin{bmatrix} \frac{x_3}{3} \\ \frac{2x_3}{3} \\ x_3 \end{bmatrix} = x_3(\begin{bmatrix} \frac{1}{3} \\ \frac{2}{3} \\ 1 \end{bmatrix})$$

So the nullspace is $x(\begin{bmatrix} \frac{1}{3} \\ \frac{2}{3} \\ 1 \end{bmatrix})$ for any real x.

```
u= Matrix([1, 2, 3])
A= Matrix([[1,1,-1], [2, -1, 0], [5, 2, -3]])

## Nullspace
x1,x2,x3= symbols('x1:4')
v= Matrix([x1, x2, x3])
sols= solve(Eq(A*v, zeros(A.rows, 1)))[0]
nsp= v.subs(sols) / x3

## Or simply
nsp= A.nullspace()
```

Now, is the vector **u** part of the nullspace? See LinearlyDependentVectors on Wolfram for different ways to test linear dependence between vectors.

```
## Test for linear dependence between u and nullspace vector:
detm= Matrix(2,2, [nsp.dot(nsp), nsp.dot(u),
            u.dot(nsp),   u.dot(u)]).det() ## == 0 -> u is in the nullspace

## Or test if 2xn matrix has rank < 2
Matrix(numpy.hstack([nsp, u])).transpose().rank() < 2 # rank<2. Vectors are lin.
                                                      # dep.
```

Actually, the simple way is just to test:

$$\mathbf{Au} = \mathbf{O} = \begin{bmatrix} 1 & 1 & -1 \\ 2 & -1 & 0 \\ 5 & 2 & -3 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Since the nullspace is the set of vectors that transformed by **A** results in the zero vector.

```
Eq(A*u, zeros(A.rows, 1)) ## True; u is in the nullspace of A
```

Exercise 3.6.8b

```
u= ones(3, 1)
B= Matrix(2, 3, [1, 4, -5,
                 -7, 5,  2])
Eq(B*u, zeros(B.rows, 1)) # True: u is in the nullspace

## Or: extract nullspace from B and check if u is linearly dependent
nsp= B.nullspace()[0]
Matrix(numpy.hstack([nsp, u])).rank() < 2 # true: nullspace and u are lin dep
```

Exercise 3.6.8c

In case $\mathbf{u} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}; \mathbf{C} = \begin{bmatrix} 1 & 3 & -9 & 5 \\ 2 & 6 & 7 & 1 \\ 1 & 3 & -8 & 1 \end{bmatrix}$ dimensions are incompatible. Trying to compute `C*u` gives `ShapeError: Matrices size mismatch.` Consequently **u** cannot be in the nullspace of **C**.

```
u= Matrix([2,1,5])
C= Matrix(2, 4, [1, 2, 3, 4,
                 5, 6, 7, 8])

Eq(C*u, zeros(C.rows, 1)) # ShapeError: Matrices size mismatch.
```

Exercise 3.6.8d

For $\mathbf{u} = \begin{bmatrix} 1 \\ 3 \\ -4 \\ 7 \end{bmatrix}; \mathbf{D} = \begin{bmatrix} 1 & -2 & 6 & 1 \\ 3 & -6 & 7 & 8 \\ 5 & 2 & 1 & 7 \\ 1 & 6 & 3 & 2 \end{bmatrix}; \mathbf{Du} \neq \mathbf{O}$ *I.e.* $\mathbf{u}$ is not in the nullspace.
Note also that $rank(\mathbf{D}) = nrow(\mathbf{D})$ and, consequently, $nullspace(\mathbf{D}) = null$. Hence no vector is in the nullspace of $\mathbf{D}$.

```
u= Matrix([1, 3, -4, 7])
D= Matrix([[1, -2, 6, 1],
           [3, -6, 7, 8],
           [5, 2, 1, 7],
           [1, 6, 3, 2]])

Eq(D*u, zeros(D.rows, 1)) # False, u not in nullspace

# Not also:
nsp= D.nullspace() # empty []
D.rank() == D.rows # Matrix is full rank
```

5. Linear Transformations

Introduction to linear transformations

A *transformation* or *mapping* converts every element in the starting set (**domain**) to a unique element in the arrival sets (**codomain**). A transformation T between spaces V and W is linear if and only if

A) $T(\mathbf{u} + \mathbf{v}) = T(\mathbf{u}) + T(\mathbf{v})$ and

B) $T(k\mathbf{u}) = kT(\mathbf{u})$

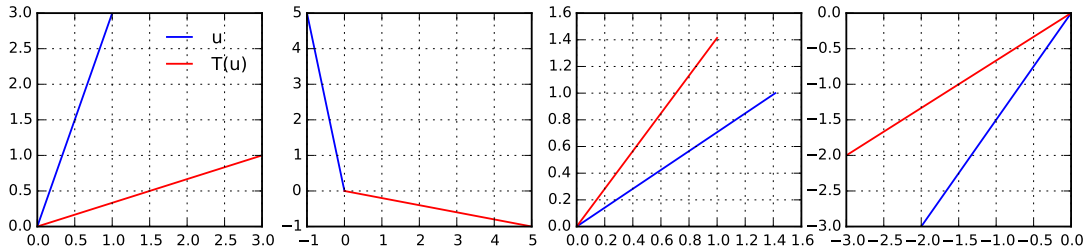
Exercise 5.1.1

Consider the transformation $T(\mathbf{u}) = \mathbf{Au}$. Determine $T(\mathbf{u})$ for the given $\mathbf{u}$.

```
A= Matrix([[0, 1], [1, 0]])
u_a= Matrix([1, 3])
u_b= Matrix([-1, 5])
u_c= Matrix([sqrt(2), 1])
u_d= Matrix([-2, -3])

plt.rc('font', size= 8)
fig, ax= plt.subplots(1, 4)
for i, u in enumerate([u_a, u_b, u_c, u_d]):
    Tu= A.dot(u)
    ax[i].plot( (0, u[0]), (0, u[1]), color= 'blue', label= 'u' )
    ax[i].plot( (0, Tu[0]), (0, Tu[1]), color= 'red', label= 'T(u)' )
    ax[i].grid()
```

```
ax[0].legend(frameon= False)
fig.set_size_inches(10, 2)
plt.savefig('figs/ex5_1_1a.pdf', bbox_inches='tight')
```



Exercise 5.1.5

Given the transformation (mapping) 59 in $\mathbb{R}^3$:

$$T\left(\begin{bmatrix} x \\ y \\ z \end{bmatrix}\right) = \begin{bmatrix} x+y \\ y+z \\ z+x \end{bmatrix} \tag{59}$$

determine whether the transformation 59 is linear.

In other words, we need to check whether the conditions **A** and **B** above are satisfied.

First define a function with the transformation T to be tested. Then define two generic vectors **u** and **v** and see if applying T satisfies the conditions for linearity.

```
x, y, z, s, t, l, k= symbols('x y z s t l k', real= True)

def T(u):
    Tu= Matrix([u[0] + u[1],
                u[1] + u[2],
                u[2] + u[0]])
    return(Tu)
```

Get two vectors and test conditions for linearity:

```
u= Matrix([x, y, z])
v= Matrix([s, t, l])

## Cond A:
assert (T(u + v)) - (T(u) + T(v)) == zeros(3, 1)

## Cond B:
assert (T(k * u) - k * T(u)).expand() == zeros(3, 1)
```

Assertions return `True` hence the transformation `T` is linear.

Pay attention how the conditions for equality are tested in SymPy . We subtract one term from the other and check the result is zero, in this case the zero vector in $\mathbb{R}^3$ $[0\ 0\ 0]^T$. Using the double equal sign `==` would return `False` in the second assertion since `==` tests for identity of form not for **symbolic identity**. See also Why does SymPy say that two equal expressions are unequal?

The trasformation T:

$$T(\begin{bmatrix} x \\ y \\ z \end{bmatrix}) = \begin{bmatrix} \sqrt{x} \\ \sqrt{y} \\ \sqrt{z} \end{bmatrix}$$

Is not linear since $\sqrt{a+b} = \sqrt{a} + \sqrt{b}$ is not satisfied.

```
def T(u):
    Tu= Matrix([sqrt(u[0]), sqrt(u[1]), sqrt(u[2])])
    return(Tu)

u= Matrix([x, y, z])
v= Matrix([s, t, l])

## Cond A
(T(u + v)).expand() - (T(u) + T(v)).expand() == zeros(3, 1) ## False

## Cond B
(T(k * u)).expand() - (k * T(u)).expand() == zeros(3, 1) ## False
```

Exercise 5.1.6 a

This is an helper function which might be useful elsewhere [2]. Here we use it to implement the tranformation 60 where we need to swap the coefficients of a polynomial. By returning a dictionary of terms and coefficients, `getCoeffDict` makes the manipulation of coefficients easier.

$$T(c_2x^2 + c_1x + c_0) = c_0x^2 + c_1x + c_2 \tag{60}$$

```
def getCoeffDict(exprs, x):
    """Return a dict of coefficients for each power of the variable ‘x‘ in
    expression ‘exprs‘. Examples:

        >>> x, a, b, c= symbols('x a b c')
        >>> getCoeffDict(a*x**2 + b*x + c, x)
        {x**2: a, x: b, 1: c}
        >>> getCoeffDict(a*x**2, x)
        {1: 0, x: 0, x**2: a}
    """
    exprs= exprs.expand()
```

[2] See also on StackOverflow How to get zero for absent constant term.

```
    n= Poly(exprs).degree(x)
    cdict= {}
    for i in range(0, n+1):
        cdict[x**i]= exprs.coeff(x, i)
    return(cdict)
```

Let's proceede testing whether 60 is a linear tranformation. Note that the transformation 60 works in polynomial space, not in Euclidean space. However, the approach remains the same.

1. Define the transformation of interest

   ```
   def T(p):
       k= getCoeffDict(p, x)
       Tp= k[1]*x**2 + k[x]*x + k[x**2]
       return(Tp)
   ```

2. Get two generic vectors:

   ```
   k= symbols('k', nonzero= True)
   c0, c1, c2= symbols('c0 c1 c2')
   b0, b1, b2= symbols('b0 b1 b2')

   p= c2*x**2 + c1*x + c0
   q= b2*x**2 + b1*x + b0
   ```

3. Test condition $T(\mathbf{u}+\mathbf{v}) = T(\mathbf{u}) + T(\mathbf{v})$

   ```
   ( (T(p+q)) - (T(p)+T(q)) ).expand() == 0 ## True
   ```

4. Test condition $T(k\mathbf{u}) = kT(\mathbf{u})$

   ```
   (T(k * p)) - (k * T(p)).expand() == 0 ## True
   ```

Exercise 5.1.6 b

Similar to *a* above. For

$$T(c_2x^2 + c_1x + c_0) = c_2^2x^2 + c_1^2x + c_0^2$$

```
def T(p):
    k= getCoeffDict(p, x)
    Tp= k[x**2]**2 * x**2 + k[x]**2 * x + k[1]**2
    return(Tp)

p= c2*x**2 + c1*x + c0
q= b2*x**2 + b1*x + b0

# Is `kT(u) = T(ku)`?
((k*T(p)).expand() - (T(k*p)).expand()) == 0 # False
```

Condition $T(k\mathbf{u}) = kT(\mathbf{u})$ can't be satisfied hence the transformation is not linear.

$$k\left(c_0^2 + c_1^2 x + c_2^2 x^2\right) \neq c_0^2 k^2 + c_1^2 k^2 x + c_2^2 k^2 x^2$$

Exercise 5.1.7 a

Check whether matrix transposition is a linear transformation. See `?Matrix.equals` for testing matrix equality.

```
def T(A):
    return A.transpose()

a, b, c, d, e, f, g, h, i= symbols('a:i')
A= Matrix(2,2, [a,b,c,d])
B= Matrix(2,2, [f,g,h,i])

T(A+B).equals(T(A) + T(B)) ## True
(k*T(A)).equals(T(k*A))    ## True
```

Matrix transposition *is* linear.

Again, we are not working in Euclidean space but the approach is the same. This is *matrix space of size* n *by* n.

Exercise 5.1.9

Test whether *integration* is linear. Note that integration maps from space $C0, 1$ to $\mathbb{R}$.

```
f= Function('f')(x)
def T(f):
    return integrate(f, (x, 0, 1))

x= symbols('x', real= True, nonnegative= True, nonzero= False)
f= Function('f')(x)
g= Function('g')(x)

(T(f+g)).equals(T(f) + T(g)) ## Returns None
(k*T(f)).equals(T(k*f))  ## True
```

First condition is satisified since:

$$\int_0^1 f(x) + g(x)\,dx = \int_0^1 f(x)\,dx + \int_0^1 g(x)\,dx$$

SymPy returns `None` and this might be ok since [*`Function.equals` returns*] *None (instead of T/F) for an expression that *is* zero but won't simplify to zero.* (See SymPy discussion group)

5.2 Kernel and range of a linear transformation

The **kernel** of a linear transformation is the set of domain (starting) values which map to zero in the codomain (arrival). *I.e.* the kernel of T is:

$$T(\mathbf{v}) = \mathbf{O})$$

The **range** or **image** of a linear transformation is the subset of values in the codomain W where a linear transformation T can arrive. Formally:

$$range(T) = \{T(\mathbf{v}) | \mathbf{v} \ in \ V\}$$

The *range* answers the question: I have this linear transformation T from domain V to codomain W. Which is the set of arrival values in W that I can produce with T?

Exercise 5.2.1

Find the kernel of the transformation in $\mathbb{R}^2 \to \mathbb{R}^2$ given by $T(\mathbf{v}) = \mathbf{A}\mathbf{v}$ with $\mathbf{A} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$.

The task is finding the solution to the equation $T(v) = 0$. In this case:

$$ker(T) = \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

```
def T(v):
    A= eye(2)
    return A * v

v= Matrix([x, y])
z= Matrix([0, 0])
ker= solve( Eq(T(v), z) )
```

Find the kernel of the transformation T in $\mathbb{R}^3 \to \mathbb{R}^2$

$$T(\begin{bmatrix} x \\ y \\ z \end{bmatrix}) = \begin{bmatrix} y - z \\ x - z \end{bmatrix}$$

We need to solve the equation $T(v) = \mathbf{O}$. Solutions are $\{x : z, \quad y : z\}$, that is any value of $zin\mathbb{R}$ makes $T(v) = \mathbf{O}$. Therefore the kernel of T is

$$ker(T) = \begin{bmatrix} z \\ z \\ z \end{bmatrix} = r \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \mid r \in \mathbb{R}$$

Note that the kernel is defined in the domain space, $\mathbb{R}^3$ here, not in the codomain.

```
x,y,z= symbols('x y z')
def T(v):
    return Matrix([v[1] - v[2],
                   v[0] - v[2]])

v= Matrix([x, y, z])

ker= solve( Eq(T(v), Matrix([0, 0])) )
```

Exercise 5.2.3

What is the kernel of the differentiation? *I.e.* the linear transformation is $T(\mathbf{p}) = p\prime(x)$. What polynomials produce zero after differentiation?

The answer is the constant polynomial c (the first derivative of a constant is zero). To show this in P^2 get a generic polynomial of order 2 $p = ax^2 + bx + c$, obtain the first derivative, $p\prime = 2ax + b$, and see which set of values for a and b make $p\prime = 0$. The answer is $a = 0;\ b = 0$. By substituting a and b in the original polynomial p we obtain the kernel of p:

$$ker(p) = 0x^2 + 0x + c = c$$

```
def T(p):
    return p.diff(x)

a,b,c,x= symbols('a b c x')
p= a*x**2 + b*x + c

eq= Eq(T(p), 0)
sols= solve(eq, [a, b])

ker= p.subs({a: sols[a], b: sols[b]}) # ker= c
```

In polynomial space of order 3 $P^3 \to P^1$ it is the same:

```
a,b,c,d,x= symbols('a b c d x')
p= a*x**3 + b*x**2 + c*x + d
eq= Eq(T(p), 0)
sols= solve(eq, [a, b, c])
ker= p.subs({a: sols[a], b: sols[b], c: sols[c]}) # ker= d
```

What is the *range* of $T(\mathbf{p}) = p\prime(x)$? The transformation differentiation produces the polynomial $2ax + b$, so the range of the differenatiation is the generic is set of polynomials of order 1 (for domain polynomial of order 2).

5.3 Rank of a linear transformation

The rank of a matrix is the number of linearly independent rows and it can be thought of as a measure of redundancy of a matrix. If the rank is much lower than the number of rows, many rows are redundant.

The rank of a linear transformation is the dimension of transformation's range. In practice, it tells how much information has been lost.

Exercise 5.3.1

For the linear transformations $T(\mathbf{v}) = \mathbf{A}\mathbf{v}$ with $\mathbf{A} = \begin{bmatrix} 1 & 2 \\ 1 & 2 \end{bmatrix}$ determine

- *kernel*(T)
  The solution to $T(\mathbf{v}) = \mathbf{O}$ is $\{x : -2y\}$. Substituting the $x = -2y$ in the generic vector we get $\mathbf{v} = \begin{bmatrix} -2y \\ y \end{bmatrix}$ or in general terms $kernel(T) = \begin{bmatrix} -2r \\ r \end{bmatrix}$ or $kernel(T) = r \begin{bmatrix} -2 \\ 1 \end{bmatrix}$. This means that the vector $[-2\ 1]^T$ multplied for any real r returns the kernel of T.
- *nullity*(T) The nullity of T is the dimension of $kernel(T)$. From above $kernel(T)$ contains **one** vector (actually one vector in a basis of $kernel(T)$) then the nullity is 1.
- *range*(T) The range is the output of T, which is $range(T) = \begin{bmatrix} x + 2y \\ x + 2y \end{bmatrix}$. The output is defined for any real number for both dimensions.
- *rank*(T) The rank is the dimension of the range. There is one vector in the range, so the rank is 1.

```
def T(v):
    A= Matrix(2, 2, [1, 2, 1, 2])
    return A*v

x, y, r= symbols('x y r')
v= Matrix([x, y])

## Kernel
xsol= solve( Eq(T(v), zeros(v.rows, 1)), x )
ker= v.subs(xsol).subs(y, r) / r

## Nullity

## Range: Generic output of T(v):
print T(v)

## Rank(T)
```

6. Determinant and the Inverse Matrix

Remember that division is not a valid operation in matrix algebra, *e.g.* $1/\mathbf{A}$ or $\mathbf{A}/\mathbf{B}$ are not meanigful. Instead of *dividing by* a matrix we *multiply by the inverse* of that matrix. For example, to find the vector of $\mathbf{x}$ in $\mathbf{A}\mathbf{x} = \mathbf{b}$ we can't do $\mathbf{x} = \mathbf{b}/\mathbf{A}$ but we can do $\mathbf{A^{-1}}\mathbf{A}\mathbf{x} = \mathbf{A^{-1}}\mathbf{b}$; $\mathbf{x} = \mathbf{A^{-1}}\mathbf{b}$. That's why matrix inversion is so important.

To compute the inverse matrix we need to pass by its determinant. In particular we need to compute $1/det(\mathbf{A})$. This means that if $det(\mathbf{A}) = 0$ matrix inversion is not possible and the matrix is said to be *singular*. Therefore, the determinant can tell whether a linear system as a unique solution. I.e. if $det(\mathbf{A}) \neq 0$ then you have a unique solution. This is why determinants are important.

Calculating the determinant is a lengthy process that increases very quickly with the size of the matrix, like $O(n!)$ or $O(n^3)$ with n the size of the matrix [3]. This means that shortcuts had to be invented to make the process feasible for large matrices, for example by splitting the original matrix in smaller and easier matrices. The various matrix decompositions also known as factorizations, like LU decomposition are aimed at reducing such complexity.

Determinant of a Matrix

The **determinant** of a *square* matrix is a unique value the tells whether the matrix is invertible. In turn this tells whether the system $\mathbf{Ax} = \mathbf{b}$ has a unique solution.

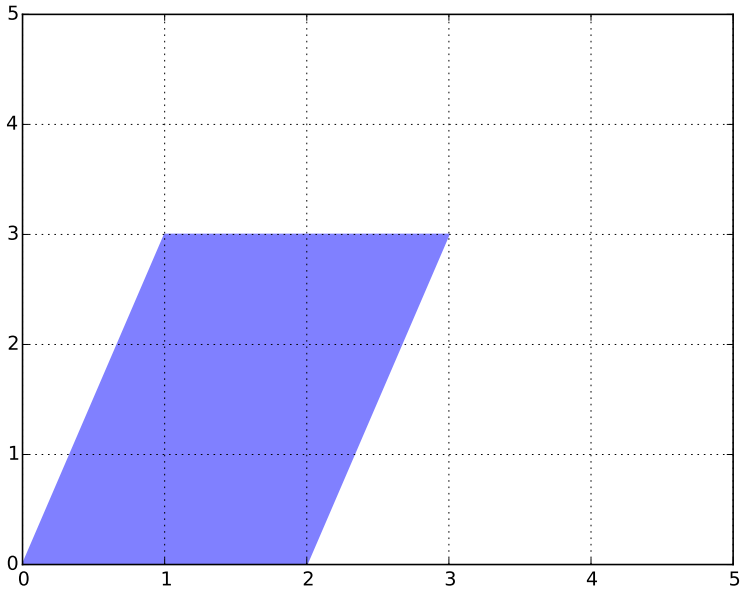
Exercise 6.1.4

Geometric interpretation of the $det(\mathbf{A})$ The determinant of a 2x2 matrix is the area of the parallelogram defined by the coordinates of the column vectors.

For $\mathbf{A} = \begin{bmatrix} 2 & 1 \\ 0 & 3 \end{bmatrix}$ show that $det(\mathbf{A}) = 6$ is the area of the parallelogram defined by the points (column vectors) $\begin{bmatrix} 2 & 0 \end{bmatrix}^T$ and $\begin{bmatrix} 1 & 3 \end{bmatrix}^T$

Note that the vertices in varibale `vert` are $\begin{bmatrix} 0 \\ 0 \end{bmatrix}$, $\begin{bmatrix} 2 \\ 0 \end{bmatrix}$, $\begin{bmatrix} 3 \\ 3 \end{bmatrix}$, $\begin{bmatrix} 1 \\ 3 \end{bmatrix}$ where the coordinates $(3, 3)$ are the row sums.

[3] http://en.wikipedia.org/wiki/Computational_complexity_of_mathematical_operations#Matrix_algebra




```
from matplotlib import patches

A= Matrix(2, 2, [2, 1, 0, 3])
A.det() # 6

verts= [(0, 0),
        A.col(0),
        (sum(A.row(0)), sum(A.row(1))),
        A.col(1)]

parall= patches.Polygon(verts, color= 'b', alpha= 0.5)
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.add_patch(parall)
ax.set_xlim((0, 5))
ax.set_ylim((0, 5))
plt.grid()
plt.draw()
fig.savefig('figs/ex6_1_4.pdf')
plt.close()
```

Note that a negative determinant means that area is rotated.

```
A1= Matrix(2,2, [1, 2, 3, 0])
A1.det() # -6
```

Similarly, the determinant of a 3x3 matrix is the volume of the parallelepid described by the matrix coordinates.

Exercise 6.1.5

Determine whether the following system has a unique solution

$$[2x + 3y = -2, \quad 5x - 2y = 14]$$

The system can be re-written in matrix form as

$$\mathbf{Ax} = \mathbf{b} \quad \begin{bmatrix} 2x + 3y \\ 5x - 2y \end{bmatrix} = \begin{bmatrix} -2 \\ 14 \end{bmatrix}$$

The determinant of $\mathbf{A}$ is $det(\mathbf{A}) = -19 \neq 0$, so the system has a unique solution for $x = 2, y = -2$

```
eq1= Eq(2*x + 3*y, -2)
eq2= Eq(5*x - 2*y, 14)
sols= solve([eq1, eq2])

# Can be re-written as
A= Matrix(2, 2, [2, +3, 5, -2])
det(A) != 0 # True -> Unique solution
b= Matrix([-2, 14])
X= Matrix([x, y])
axb= Eq(A*X, b)
solve(axb) # Same as above
```

Note that a unique solution is found for any values of the vector $\mathbf{b}$, the right hand side of the equations or responses. What decides whether the system has a unique solution are the coefficients in the matrix $\mathbf{A}$, *i.e.* the left hand side of the equation or predictors. For example:

```
X= Matrix([x, y])
A= Matrix(2, 2, [2, +3, 5, -2])
b1= Matrix([-2, 14])
b2= Matrix([7, 5])
b3= Matrix([pi, pi])

for b in [b1, b2, b3]:
    print solve(Eq(A*X, b))
# {x: 2, y: -2}
# {x: 29/19, y: 25/19}
# {x: 5*pi/19, y: 3*pi/19}
```

Determinant of other matrices

Exercise 6.2.2

Show that $det(\begin{bmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ 7 & 3 & -2 \\ 4 & 2 & 7 \end{bmatrix}) = 25\mathbf{i} - 57\mathbf{j} + 2\mathbf{k}$

```
i,j,k= symbols('i j k')
M= Matrix(3, 3, [i, j, k, 7, 3, -2, 4, 2, 7])
M.det() #
```

Exercise 6.2.3

find x so that

$$det(\begin{bmatrix} 1 & 0 & -3 \\ 5 & x & -7 \\ 3 & 9 & x-1 \end{bmatrix}) = 0$$

We have $det(\mathbf{M}) = x^2 + 8x - 72$, so to have $x^2 + 8x - 72 = 0$ we need $x = -4 + 2\sqrt{22}, \quad x = -2\sqrt{22} - 4$. As shown by substituting these solutions in $\mathbf{M}$ and calculating the determinant.

```
M= Matrix([[1, 0, -3],
           [5, x, -7],
           [3, 9, x-1]])

detm= det(M)
eq= Eq(detm, 0)
sols= solve(eq)
det(M.subs(x, sols[0])) == 0 # True
det(M.subs(x, sols[1])) == 0 # True
```

Exercise 6.2.4

Find the **cofactor matrix C** of $\mathbf{A} = \begin{bmatrix} 1 & 0 & 5 \\ -2 & 3 & 7 \\ 6 & -1 & 0 \end{bmatrix}$

Cofactor matrix:

$$\mathbf{C} = \begin{bmatrix} 7 & 42 & -16 \\ -5 & -30 & 1 \\ -15 & -17 & 3 \end{bmatrix}$$

The cofactor matrix is used to calculate the **inverse** of matrix.

```
A= Matrix([[1, 0, 5],
           [-2, 3, 7],
           [6, -1, 0]])
C= A.cofactorMatrix()
```

Maybe useful to note:

$$\mathbf{A} = \begin{bmatrix} x_1 & x_2 \\ x_3 & x_4 \end{bmatrix} \quad \mathbf{C_A} = \begin{bmatrix} x_4 & -x_3 \\ -x_2 & x_1 \end{bmatrix}$$

and

$$\mathbf{A} = \begin{bmatrix} x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 \\ x_7 & x_8 & x_9 \end{bmatrix} \mathbf{C_A} = \begin{bmatrix} x_5x_9 - x_6x_8 & -x_4x_9 + x_6x_7 & x_4x_8 - x_5x_7 \\ -x_2x_9 + x_3x_8 & x_1x_9 - x_3x_7 & -x_1x_8 + x_2x_7 \\ x_2x_6 - x_3x_5 & -x_1x_6 + x_3x_4 & x_1x_5 - x_2x_4 \end{bmatrix}$$

```
x1,x2,x3,x4,x5,x6,x7,x8,x9= symbols('x1:10')

A= Matrix(2,2,[x1, x2, x3, x4])
A.cofactorMatrix()

A= Matrix(3,3,[x1,x2x,x3,x4,x5,x6,x7,x8,x9])
A.cofactorMatrix()
```

Exercise 6.2.5

The inverse of $\mathbf{A}$ is:

$$\mathbf{A}^{-1} = \frac{1}{det(\mathbf{A})} adj(\mathbf{A})$$

where the *adjoint* of $\mathbf{A}$ is the transpose of the cofactor matrix of $\mathbf{A}$: $adj(\mathbf{A}) = \mathbf{C}^T$

```
def inv(A):
    C= A.cofactorMatrix()
    adj= C.transpose()
    inva= (1 / det(A)) * adj
    return inva
```

Find the inverse of the following matrices.

```
A= Matrix(2, 2, [9, 2, 13, 3])
inv(A) # checked against A.inv()

D= Matrix(3, 3, [3, -5, 3,
                 2, 1, -7,
                 -10, 4, 5])
inv(D)
```

Exercise 6.2.12

Find the values of k for which the matrix $\mathbf{A} = \begin{bmatrix} k & 1 & 2 \\ 0 & k & 2 \\ 5 & -5 & k \end{bmatrix}$ is invertible. Since inversion requires $det(\mathbf{A}) \neq 0$, we need to get the determinant of $\mathbf{A}$ and solve it for k. Values of k that are not solutions of $det(\mathbf{A}) = 0$ make $\mathbf{A}$ invertible.
$det(\mathbf{A}) = k^3 + 10$ solved for k in $\left[-\sqrt[3]{10}, \quad \frac{\sqrt[3]{10}}{2} - \frac{\sqrt{3}i}{2}\sqrt[3]{10}, \quad \frac{\sqrt[3]{10}}{2} + \frac{\sqrt{3}i}{2}\sqrt[3]{10}\right]$

```
A= Matrix(3, 3, [k, 1, 2,
                 0, k, 2,
                 5, -5, k])

sols= solve(Eq(A.det(), 0))

## Check A is not invertible for k in sols:
A.subs(k, sols[0]).inv()  ## ValueError: Matrix det == 0; not invertible.
A.subs(k, sols[1]).inv()  ## ValueError: Matrix det == 0; not invertible.
A.subs(k, sols[2]).inv()  ## ValueError: Matrix det == 0; not invertible.

## Check it is invertible for other values of k
A.subs(k, 0).inv()
A.subs(k, sols[0]+1).inv()
A.subs(k, sols[2]+1).inv()
```

Properties of determinants

As noted above, calculating the determinant of large matrices can take ages. However, the determinant of diagonal and triangular matrices are very easy to compute: Just multiply the entries along the leading diagonal. This means that if a matrix can be rearranged in one or more triangular matrices, by means of row operations and decomposition, than the calculation of the determiant is much simpler.

Exercise 6.3.1

Find the determinant of $\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -10 & 0 \\ 0 & 0 & 1 \end{bmatrix}$.

The matrix is diagonal: Multiply the entries along the leading diagonal $det(\mathbf{A}) = (1)(-10)(1) = -10$

```
def diagprod(A):
    dp= 1
    for i in range(A.rows):
        dp= dp*A[i, i]
    return dp

A= Matrix(3, 3, [1, 0, 0,
                 0, -10, 0,
                 0, 0, 1])
diagprod(A) # -10
# Cheack against
diagprod(A) - A.det() == 0 # True
```

For $\mathbf{B} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$

The matrix is not diagonal or triangular as such but it can be made diagonal by swapping rows.

```
B= Matrix(3, 3, [1, 0, 0, 0, 0, 1, 0, 1, 0])
```

LU factorization

Lower-Upper (LU) factorization decomposes a square matrix in an upper and lower triangular matrix. After this decomposition the computation of the determinant is much faster. LU decomposition can also be used as a fast way to solve linear systems. Computer systems use LU to solve linear systems.
The general procedure to obtain the **L** and **U** matrices for a matrix **M** is

1. Apply row operations $1, 2, \ldots, n$ to **M** until you obtain the **U** matrix.
2. Now start from the identity matrix **I**. Apply to **I** the same row operations in *reverse* order $n, \ldots, 2, 1$ and you get the **L** matrix.

To solve the system $\mathbf{Ax} = \mathbf{b}$ consider the equivalence $\mathbf{Ax} = (\mathbf{LU})\mathbf{x} = \mathbf{L}(\mathbf{Ux}) = \mathbf{b}$. Set $\mathbf{Ux} = \mathbf{y}$ and compute $\mathbf{y}$. Now substitute $\mathbf{y}$ to $\mathbf{Ux}$ to obtain $\mathbf{Ly} = \mathbf{b}$, solve it and you have the solution to the original system $\mathbf{Ax} = \mathbf{b}$. The **LU** route is much faster than solving $\mathbf{Ax} = \mathbf{b}$ directly since we deal with trangular matrices.
Home made python code for solving $\mathbf{Ax} = \mathbf{b}$ via **LU** decomposition:

```
def LUsolve(A, b):
    x= symbols('x1:%s' %(b.rows+1))
    x= Matrix(b.rows, 1, x)
    Ax_b= Eq(A*x, b)                # Original system Ax = b
    L, U= A.LUdecomposition()[0:2] # Get U & L triang matrices from A
    LUx_b= Eq(L*U*x, b)             # Decompose Ax = b -> (LU)x = b
    assert(LUx_b - Ax_b == 0)       # Check transformation works
    y= U*x                          # Compute Ux = y and use y in place of Ux
    Ly_b= Eq(L*y, b)                #
    assert(Ly_b - Ax_b == 0)        # Check transformation works
    sols= solve(Ly_b, x)            # Solve Ly =b
    return sols
```

In addition, **LU** decomposition is useful to find the inverse of a matrix:

$$\mathbf{A^{-1}} = (\mathbf{LU})^{-1} = \mathbf{U^{-1}L^{-1}}$$

Exercise 6.4.1

Solve the following linear systems using **LU** factorization

$$\mathbf{Ax} = \mathbf{b}; \quad \mathbf{A} = \begin{bmatrix} 1 & 2 & 2 \\ 3 & -3 & -2 \\ 4 & -1 & -5 \end{bmatrix}; \quad \mathbf{b} = \begin{bmatrix} 5 \\ 0 \\ -10 \end{bmatrix}$$

The solution is for $\{x_1 : 1, \quad x_2 : -1, \quad x_3 : 3\}$ and it is (obvioulsy) the same whether we use the home made LU solver or SymPy built in. In fact SymPy probably uses LU decomposition.

```
x1, x2, x3= symbols('x1:4')
A= Matrix(3, 3, [1, 2, 2, 3, -3, -2, 4, -1, -5])
b= Matrix(3, 1, [5, 0, -10])
x= Matrix(3, 1, [x1, x2, x3])

sols= LUsolve(A, b) # {x1: 1, x2: -1, x3: 3}

# Check against sympy
axb= Eq(A*x, b)
ssym= solve(axb, x)
for _x in x:
    assert(sols[_x] == ssym[_x])
```

Exercise 6.4.2

Same as above solve the linear system $\mathbf{Ax} = \mathbf{b}$ for $\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 17 & 22 & 27 & 8 \\ 77 & 44 & 47 & -494 \\ -10 & 1 & 7 & 63 \end{bmatrix}$; $\mathbf{b} = \begin{bmatrix} -10 \\ 22 \\ 2106 \\ -243 \end{bmatrix}$

Solved for $\{x_1 : 1, \quad x_2 : -2, \quad x_3 : 3, \quad x_4 : -4\}$.

```
A= Matrix([[1, 2, 3, 4],
           [17, 22, 27, 8],
           [77, 44, 47, -494],
           [-10, 1, 7, 63]])
b= Matrix(4, 1, [-10, 22, 2106, -243])

sols= LUsolve(A, b)
```

Exercise 6.4.5

Find the inverse of $\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \\ 3 & 7 & 14 \\ 4 & 13 & 38 \end{bmatrix}$

```
A= Matrix(3, 3, [1,2,3, 3,7,14, 4,13,38])
L, U, _= A.LUdecomposition()
Ainv= U.inv() * L.inv()
assert(Ainv == A.inv())

## Note that U^-1 * L^-1 != L^-1 * U^-1
L.inv() * U.inv()
```

7. Eigenvalues and Eigenvectors

Introduction to eigenvalues and eigenvectors

Eigenvector $\mathbf{u}$ and eigenvalue λ of a square matrix $\mathbf{A}$ is a of vector and matched scalar value so that

$$\mathbf{Au} = \lambda\mathbf{u}$$

If you imagine the matrix $\mathbf{A}$ as a transformation, $\mathbf{u}$ is that vector that transformed by $\mathbf{A}$ is only shrank or extended in length, but it is not rotated or transposed. The magnitude of extension/shrinkage that $\mathbf{A}$ applies to $\mathbf{u}$ is the scalar value λ, the eigenvalue.

The eigenvalues are found by applying the **characteristic equation**

$$det(\mathbf{A} - \mathbf{I}\lambda) = \mathbf{0}$$

Where does the characteristic equation come from? Consider the following rearragments:

$\mathbf{Au} = \lambda\mathbf{u} \rightarrow$

$\mathbf{Au} = \lambda\mathbf{Iu}$ *multiply by* $\boldsymbol{I}$ *doesn't change anything*

$\mathbf{Au} - \lambda\mathbf{Iu} = \mathbf{0}$

$(\mathbf{A} - \lambda\mathbf{I})\mathbf{u} = \mathbf{0}$

Consider the matrix $\mathbf{A} - \lambda\mathbf{I}$, what value of λ makes this matrix equal to $\mathbf{0}$? Answer, solve $det(\mathbf{A} - \lambda\mathbf{I}) = \mathbf{0}$ for λ.

This is a useful informal description of eigenvalues/vectors:

If we consider matrix as a transformation then in simple terms eigenvalue is the strength of that transformation in a particular direction known as eigenvector.

Exercise 7.1.1.

Find the eigenvalues and eigenvectors of the matrix $\mathbf{A} = \begin{bmatrix} 7 & 3 \\ 0 & -4 \end{bmatrix}$.

First find the eigenvalue(s) by setting

$$det(\mathbf{A} - \mathbf{I}\lambda) = det(\begin{bmatrix} 7 & 3 \\ 0 & -4 \end{bmatrix} - \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix}) = det(\begin{bmatrix} -\lambda + 7 & 3 \\ 0 & -\lambda - 4 \end{bmatrix}) = \lambda^2 - 3\lambda - 28 = 0$$

Solving $\lambda^2 - 3\lambda - 28 = 0$ we find $\lambda = -4; \lambda = 7$.

To find the correspondong eigenvectors, substitute each value of λ in $\mathbf{Au} = \lambda\mathbf{u}$ and solve for $\mathbf{u}$. For $\lambda = -4$ we have

$$\mathbf{Au} = \lambda\mathbf{u}; \quad \begin{bmatrix} 7 & 3 \\ 0 & -4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = -4 \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

solved for x_1

$$\left\{ x_1 : -\frac{3x_2}{11} \right\}$$

Setting x_2 to, say, 1 we obtain the eigenvector $\mathbf{u}_{\lambda=-4} = [-3/11 \quad 1]^T$. **NB** we could set x_2 to any value, for example $x_2 = 11$ and obtain $[-3 \quad 11]^T$. In fact, there is an infinite number of equivalent eigenvectors for each eigenvalue since any multiple of $\mathbf{u}$, *i.e.* $k\mathbf{u}$ for $k \neq 0$, is valid.

The point is that matrix $\mathbf{A}$ multiplies the vector $\mathbf{u}$ by the scalar λ, any vector that multiplied by $\mathbf{A}$ returns $\lambda\mathbf{u}$ is an eigenvector of that λ.

Same for $\lambda = 7$: $\begin{bmatrix} 7 & 3 \\ 0 & -4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = -7 \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ the solution for x_2 is $\{x_2 : 0\}$ while a value for x_1 could be 1, so that $\mathbf{u}_{\lambda=7} = [1 \quad 0]^T$.

```
lamda= symbols('lamda')
A= Matrix(2, 2, [7, 3, 0, -4])

# Find eigenvalue(s)
deta= det(A - eye(A.rows) * lamda)
eigval= solve(deta, lamda)

# Find eigenvectors by sub lambda in A*u=lambda*u
u= Matrix(A.rows, 1, symbols('x1:%s' %(A.rows+1)))
eigvec= Eq(A*u, eigval[0]*u)
s1= solve(eqvec, x1)
eigvec= Eq(A*u, eigval[1]*u)
s2= solve(eqvec, x1)

eigvec= Eq(A*u, eigval[0]*u)
solve(eigveceq, u[0])
```

Exercise 7.1.2

Find the general eigenvectors for $\mathbf{A} = \begin{bmatrix} 1 & 0 & 4 \\ 0 & 4 & 0 \\ 3 & 5 & -3 \end{bmatrix}$ given the eigenvalue $\lambda_1 = 4$ and $\lambda_2 = -5$.

For each given λ we need to solve $\mathbf{Au} = \lambda\mathbf{u}$ for a generic vector $[x \; y \; z]^T$. For $\lambda = 4$ we have solutions $\left\{x : \frac{4z}{3}, \quad y : \frac{3z}{5}\right\}$. Setting $z = 1$ we obtain $\mathbf{u}_4 = [\frac{4}{3} \; \frac{3}{5} \; 1]^T$ and in general $\mathbf{u}_4 = k[\frac{4}{3} \; \frac{3}{5} \; 1]^T$. We could use a more convenient value of z, like $z = 15$ so that we obtain $\mathbf{u}_4 = k[20 \; 9 \; 15]^T$

```
x, y, z= symbols('x y z')
A= Matrix([[1, 0, 4], [0, 4, 0], [3, 5, -3]])
u= Matrix(A.rows, 1, [x, y, z])

lamda= 4
evec= Eq(A*u, lamda*u)
sols= solve(evec, [x, y, z])
z1= 1
sols= solve(evec.subs(z, z1), [x, y])
u4= Matrix(A.rows, 1, [sols[x], sols[y], z1])
# Or
```

```
z1= 15
sols= solve(evec.subs(z, z1), [x, y])
u4= Matrix(A.rows, 1, [sols[x], sols[y], z1])
```

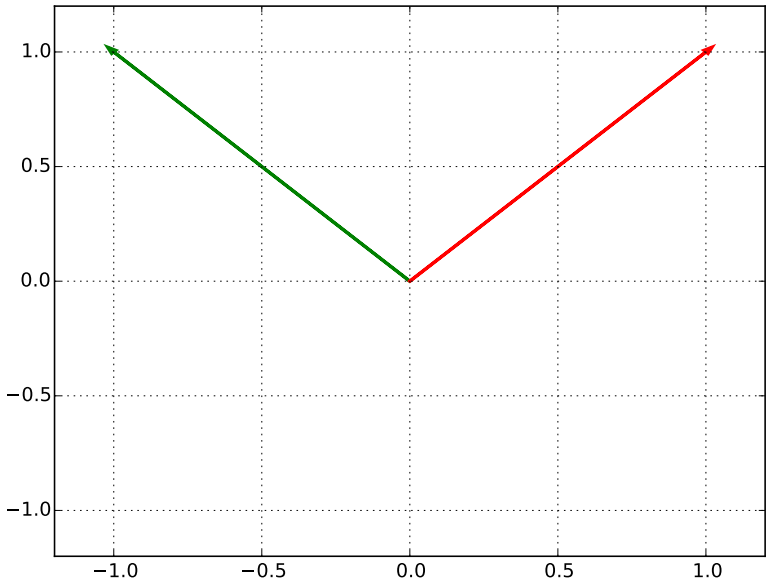
Exercise 7.1.3

Plot the eigenspaces E_λ of $\mathbf{A} = \begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix}$.

This time we use SymPy to find the eigenvectors, which are

$$\left[\left(2,\quad 1,\quad \left[\begin{bmatrix} -1 \\ 1 \end{bmatrix}\right]\right),\quad \left(4,\quad 1,\quad \left[\begin{bmatrix} 1 \\ 1 \end{bmatrix}\right]\right)\right]$$

The output of SymPy `Matrix.eigenvects()` is a list of tuples, one tuple for eigenvalue. Each tuple contains: Eigenvalue, multitlicity, eigenvector.




```
A= Matrix(2, 2, [3, 1, 1, 3])
evecs= A.eigenvects()

plt.subplot()
plt.xlim(-1.2, 1.2)
plt.ylim(-1.2, 1.2)
plt.arrow(0, 0, float(evecs[0][2][0][0]), float(evecs[0][2][0][1]),
    linewidth= 2, color= 'g')
plt.arrow(0, 0, float(evecs[1][2][0][0]), float(evecs[1][2][0][1]),
    linewidth= 2, color= 'r')
plt.grid()
plt.savefig('figs/ex7_1_3.pdf')
plt.close()
```

Exercise 7.1.4

What is the effect of multiplying matrix $\mathbf{A} = \begin{bmatrix} 5 & -2 \\ 7 & -4 \end{bmatrix}$ by its eigenvector. Plot the eigenspace and compute a basis vector for each eigenspace.

Eigenvalues, multiplicity and eigenvectors:

$$\left[\left(-2, \quad 1, \quad \left[\begin{bmatrix} \frac{2}{7} \\ 1 \end{bmatrix}\right]\right), \quad \left(3, \quad 1, \quad \left[\begin{bmatrix} 1 \\ 1 \end{bmatrix}\right]\right)\right]$$

Multiplying (transforming) the eigenvector by the matrix has the same effect as extending the eigenvector by a factor λ, there is no change in direction of the eigenvector. *I.e.* $\mathbf{Au} = \lambda\mathbf{u}$.

The basis for the eigenspace are the eigenvectors $\mathbf{u} = \begin{bmatrix} \frac{2}{7} \\ 1 \end{bmatrix}, \quad \mathbf{v} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$. Which means $\mathbf{u}$ is the basis vector for $\lambda = -2$ and $\mathbf{v}$ is the basis vector for $\lambda = 3$. They are a basis because every vector in their space can be created by linear combination of these vector. In fact, every segment sitting on the line $[2/7 \quad 1]^T$ can be obtained by multiplying $[2/7 \quad 1]^T$ by an appropriate scalar (and the same for any segment sitting on $\mathbf{u}$).

```
A= Matrix(2,2, [5, -2, 7, -4])
evecs= A.eigenvects()
ev= [x[2][0] for x in evecs]

# Au = lambda*u
u= evecs[0][2][0]
lamu= evecs[0][0]
A* u == lamu * u ## True

v= evecs[1][2][0]
lamv= evecs[1][0]
A* v == lamv * v ## True
```

Properties of eigenvalues and eigenvectors

Exercise 7.2.3

The characteristic equation is the characteristic polynomial equated to zero.

The characteristic polynomial ($p(\lambda)$) for a 2x2 matrix $\mathbf{A}$ is $p(\lambda) = det(\mathbf{A} - \lambda\mathbf{I})$ and it is equivalent to $p(\lambda) = \lambda^2 - tr(\mathbf{A})\lambda + det(\mathbf{A})$, as shown below.

```
a,b,c,d= symbols('a:d', real= True)
lamda= symbols('lambda', real= True)
A= Matrix(2, 2, [a, b, c, d])

charp= lamda**2 - A.trace() * lamda + A.det()
chareq= det(A - lamda * eye(2))
Eq(chareq.expand() - charp.expand()) # True
```

Note that equality between polynomials is tested after `expand`ing them to canonical form. Simply testing `charp - chareq` will not return 0 as it should. See also SymPy FAQ

Exercise 7.2.4

Given $\mathbf{A}$ a 2x2 matrix with $tr(\mathbf{A}) = 2a$ and $det(\mathbf{A}) = a^2$, obtain the eigenvalue λ.

Start from the characteristic polynomial $p(\lambda) = \lambda^2 - tr(\mathbf{A})\lambda + det(\mathbf{A})$ and substitute the values $tr(\mathbf{A}) = 2a$ and $det(\mathbf{A}) = a^2$. The resulting equation $p(\lambda) = ad - bc + \lambda^2 - \lambda\,(a + d)$ is solved for $\lambda = a$. Therefore given the conditions above an eigenvalue of $\mathbf{A}$ is a. Since a is the only solution and the matrix is 2x2, than a has multiplicity 2, *i.e.* it occurs twice.

```
a,b,c,d= symbols('a:d', real= True)
lamda= symbols('lambda', real= True)
A= Matrix(2, 2, [a, b, c, d])

charp= lamda**2 - A.trace() * lamda + A.det()
charp2= charp.subs(A.trace(), 2*a).subs(A.det(), a**2)
ev= solve(charp2, lamda) # [a]
```

Exercise 7.2.5

For $\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 \\ -3 & 1 & 0 \\ 7 & 9 & 1 \end{bmatrix}$ obtain eigenvalues and eigenvectors. Write down a set of basis vectors.

Since this is a triangular matrix, the eigenvalues can be read from the main diagonal. In this case $\lambda = 1$ with multiplicity 3. Consequently there is only one eigenvector $\mathbf{v} = [0\ 0\ 1]^T$.

The basis is $\mathbf{x}\ \mathbf{y}\ \mathbf{z} = [0\ 0\ 1]^T$ which means that the eigenspace for $\lambda = 1$ is the vertical axis z in 3D space.

```
A= Matrix([[1, 0, 0], [-3, 1, 0], [7, 9, 1]])
ev= A.eigenvects()
```

Note that in triangular matrices only the leading diagonal determines the eigenvalues/vectors. The generic triangular matrix $\begin{bmatrix} 1 & 0 & 0 \\ a & 1 & 0 \\ b & c & 1 \end{bmatrix}$ has the same values as the matrix $\mathbf{A}$ above:

```
A1= Matrix([[1, 0, 0], [a, 1, 0], [b, c, 1]])
ev1= A.eigenvects() ## Same as above
ev == ev1 # True
```

Note that if n eigenvalues are distinct then the corresponding eigenvectors are linearly independent.

Diagonalization

Diagonalization is the process of converting an n x n matrix in a diagonal one. The values on the diagonal matrix are the eigenvalues of the original matrix. Diagonal matrices are much easier to work with then full matrices.

The power of matrices, useful in Markov models, is much simplified by means of diagonalization since $\mathbf{D}^n$ is easy to compute if $\mathbf{D}$ is a diagonal matrix.

Diagonalize $\mathbf{A}$ means that:

$$\mathbf{P^{-1}AP = D} \quad \textit{and} \quad \mathbf{A = PDP^{-1}}$$

That is, there exists a matrix $\mathbf{P}$ so that the multiplication above transforms $\mathbf{A}$ into a diagonal matrix $\mathbf{D}$. $\mathbf{P}$ is a matrix of eigenvectors of $\mathbf{A}$:

$$\mathbf{A^m = PD^mP^{-1}}$$

(that's why diagonalization appears in the chapter about eigenvalues/vectors).

Exercise 7.3.1

```
A= Matrix(2, 2, [1, 0, 0, 2])
evecs= [x[2][0] for x in A.eigenvects()]
P= Matrix(np.column_stack(evecs))
D= P.inv() * A * P
Eq(A, P * D * P.inv())

A= Matrix(2, 2, [2, 2, 1, 3])
evecs= [x[2][0] for x in A.eigenvects()]
P= Matrix(np.column_stack(evecs))
D= P.inv() * A * P
Eq(A, P * D * P.inv())
```

Exercise 7.3.2

Find $\mathbf{A}^5$ for $\mathbf{A} = \begin{bmatrix} 2 & 2 \\ 1 & 3 \end{bmatrix}$.

1. Find the eigenvectors of $\mathbf{A}$. Put them in matrix $\mathbf{P}$ by column binding.
2. Get the eigenvalues of $\mathbf{A}$. Put them along the diagonal matrix $\mathbf{D}$.
3. Calculate power of $\mathbf{A}$ as $\mathbf{A}^m = \mathbf{PD^mP^{-1}}$ for $m = 5$.

We use SymPy `Matrix.eigenvects` to obtain eigenvectors/values. For $\mathbf{A}$ we have the list of tuples $\left[\left(1, \quad 1, \quad \left[\begin{bmatrix} -2 \\ 1 \end{bmatrix}\right]\right), \quad \left(4, \quad 1, \quad \left[\begin{bmatrix} 1 \\ 1 \end{bmatrix}\right]\right)\right]$. Each tuple is (eigenvalue, multiplicity, eigenvector).

The matrix of eigenvectors is $\mathbf{P} = \begin{bmatrix} -2 & 1 \\ 1 & 1 \end{bmatrix}$ and the diagonal matrix of eigenvalues is $\mathbf{D} = \begin{bmatrix} 1 & 0 \\ 0 & 4 \end{bmatrix}$

Now $\mathbf{A}^5$ is easy to compute as $\mathbf{A}^5 = \mathbf{PD^5P^{-1}} = \begin{bmatrix} -2 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1^5 & 0 \\ 0 & 4^5 \end{bmatrix} \begin{bmatrix} -\frac{1}{3} & \frac{1}{3} \\ \frac{1}{3} & \frac{2}{3} \end{bmatrix} = \begin{bmatrix} 342 & 682 \\ 341 & 683 \end{bmatrix}$

```
A= Matrix(2, 2, [2, 2, 1, 3])

def powerMat(A, m):
    """Return the square matrix A to the power of m"""
    # Eigenvectors/values
    ev= A.eigenvects()
    P= Matrix(np.array([x[2][0] for x in ev])).transpose()
    D= diag(*[x[0] for x in ev])
    Am= P * D**Rational(m) * P.inv()
    return Am

A5= powerMat(A, 5)

## Check against SymPy
A**5 == A5 ## True
```

For $\mathbf{A} = \begin{bmatrix} 3 & 0 \\ 4 & 4 \end{bmatrix}^{-1/2} = \begin{bmatrix} \frac{\sqrt{3}}{3} & 0 \\ -\frac{4\sqrt{3}}{3} + 2 & \frac{1}{2} \end{bmatrix}$
Note the use of `Rational(-1/2)` required by SymPy to deal with rational numbers.

```
A= Matrix(2, 2, [3, 0, 4, 4])
Am= powerMat(A, -1/2)
A**Rational(-1/2) == Am ## True
```

Exercise 7.3.4

Determine whether the following matrices are diagonalizable.
To answer the question consider that:

- If an n x n matrix has n distinct eigenvalues, than the corresponding eigenvectors are *linearly independent.*
- If an n x n matrix has n distinct eigenvalues then the matrix is diagonalizable.

Note that in some cases a matrix can be diagonalizable even if doesn't have n distinct eigenvalues (*I.e.* the above is not an *if and only if* condition).
For the identity matrix $\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ we have one eigenvalue $\lambda = 1$ with multiplicity 3 and eigenvectors $\left[\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \right]$.

The matrix of eigenvectors therefore is $\mathbf{P} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$, and the matrix of eigenvalues $\mathbf{P} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$.
The identity matrix is therefore diagonalizable even if the distinct eigenvalues is less than the matrix dimension since $\mathbf{I} = \mathbf{A} = \mathbf{PDP^{-1}}$.

```
A= eye(3)
ev= A.eigenvects()
P= Matrix(np.column_stack(ev[0][2])).transpose()
D= diag(*[ev[0][0]]*A.rows)

A == P * D * P.inv() # True
```

For $\mathbf{A} = \begin{bmatrix} -1 & 2 & 3 \\ 0 & 2 & 5 \\ 0 & 0 & 8 \end{bmatrix}$ the matrix is triangular and therefore the eigenvalues are on the leading diagonal. The eigenvalues are distinct and therefore the matrix is diagonalizable.

```
A= Matrix([[-1, 2, 3], [0, 2, 5], [0, 0, 8]])
ev= A.eigenvects()
P= Matrix(np.array([x[2][0] for x in ev])).transpose()
D= diag(*[x[0] for x in ev])
A == P * D * P.inv()
```

Diagonalization of Symmetric Matrices

Some useful definitions and properties:

- A matrix is symmetric if $\mathbf{A} = \mathbf{A^T}$.
- Symmtric matrices can be diagonalized by an orthogonal matrix.
- Orthogonality: Two vectors are orthogonal, *i.e.* their angle is 90°, if their inner (dot) product is zero:

  $$\mathbf{u} \cdot \mathbf{v} = 0$$

  Note that orthogonality makes sense for vectors of any dimension even if geometrically vectors of length > 3 cannot be visualized
- The distinct eigenvalues of a symmetric matrices have corresponding eigenvectors which are othogonal. *N.B.* the covariance matrix is symmetric.

For example, matrix $\mathbf{M} = \begin{bmatrix} -5 & 4 & 2 \\ 4 & -5 & 2 \\ 2 & 2 & -8 \end{bmatrix}$ is symmetric and has the following eigenvalues, multiplicity and eigenvectors

$$\left[\left(-9, \quad 2, \quad \left[\begin{bmatrix} -1 \\ 1 \\ 0 \end{bmatrix}, \quad \begin{bmatrix} -\frac{1}{2} \\ 0 \\ 1 \end{bmatrix} \right] \right), \quad \left(0, \quad 1, \quad \left[\begin{bmatrix} 2 \\ 2 \\ 1 \end{bmatrix} \right] \right) \right]$$

Eigenvalue -9 has multiplicity 2 (two eigenvectors), and these are *not* orthogonal. However, these two eigenvectors are orthogonal to the eigenvector with eigenvalue 0: $\begin{bmatrix} -1 \\ 1 \\ 0 \end{bmatrix} \cdot \begin{bmatrix} -\frac{1}{2} \\ 0 \\ 1 \end{bmatrix} = 1/2$ and $\begin{bmatrix} 2 \\ 2 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} -1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} -\frac{1}{2} \\ 0 \\ 1 \end{bmatrix} = 0$

```
M= Matrix([[-5, 4, 2],
           [4, -5, 2],
           [2, 2, -8]])
ev= M.eigenvects()
u= ev[0][2][0]
v= ev[0][2][1]
z= ev[1][2][0]

u.dot(v) == 0 # False
z.dot(u) == 0 # True
z.dot(v) == 0 # True
```

- A matrix $\mathbf{Q}$ is **orthogonal** if its column vectors are orthogonal to each other and of norm 1, *i.e.* they orthonormal.
- **Orthogonal diagonalization** of matrix $\mathbf{A}$ means the following decomposition

$$\mathbf{A} = \mathbf{Q^{-1}DQ} = \mathbf{Q^{T}DQ}$$

  As before, $\mathbf{Q}$ column-binds the normalized eigenvectors of $\mathbf{A}$. $\mathbf{D}$ is a diagonal matrix with the eigenvalues of $\mathbf{A}$ along the leading diagonal.
- Because of the diagonalization $\mathbf{A} = \mathbf{Q^{-1}DQ} = \mathbf{Q^{T}DQ}$ symmetric matrices are easy to diagonalize since we don't need to invert $\mathbf{Q}$, an expensive operation, we just need to transpose $\mathbf{Q}$.

Exercise 7.4.1

Find and orthogonal matrix $\mathbf{Q}$ which diagonalizes the following matrices.

- Find eigenvectors of $\mathbf{A}$, column-bind them

```
def normalize(v):
    """Normalize vector v
    """
    vsq= sqrt(sum([x**2 for x in v]))
    return 1 / vsq * v

def getQD(M):
    """Return a tuple of
    * Matrix of normalized eigenvectors of M column binded
    * Diagonal matrix of eigenvalues of A
    """
    assert M == M.transpose() # Check M is symmetric.
    ev= M.eigenvects()
    eigval_lst= []
    eigvec_lst= []
    for vmv in ev:
        eigvec= vmv[2]
        for v in eigvec:
            # For each eigenvector in this eigenvalue.
            eigval_lst.append(vmv[0])
            vnorm= normalize(v)
            eigvec_lst.append(list(vnorm))
    Q= Matrix(eigvec_lst).transpose()
    assert Q * Q.transpose() == eye(A.rows) # Test Q is orthogonal
    D= diag(*eigval_lst)
    return (Q, D)

A= Matrix(2, 2, [1, 0, 0, 2])
Q,D= getQD(A)
Q.transpose() * A * Q == D # True

# ----

A= Matrix(2, 2, [5, 12, 12, -5])
Q,D= getQD(A)
Q.transpose() * A * Q == D # True

# ----

A= Matrix([[5, sqrt(12)], [sqrt(12), 1]])
Q,D= getQD(A)
Q.transpose() * A * Q == D # True
```

Singular Value Decomposition

Useful concepts and properties:

- Eigenvalues/vectors can be obtained only for square matrices, so if **A** is not square we can't get eigenvalues/vectors.

- Good news is $\mathbf{A^T A}$ is square *and* symmetric! This means that $\mathbf{A^T A}$ can be easily diagonalized.
- The eigenvalues of $\mathbf{A^T A}$ are either positive or zero.

Exercise 7.5.1

Decompose $\mathbf{A}$ so that

$$\mathbf{A = UDV^T}$$

- Singular values of $\mathbf{A}$: The square root of the eigenvalues of $\mathbf{A^T A}$, denoted $\sigma_1, \sigma_2, \ldots, \sigma_n$ with n the number of columns in $\mathbf{A}$. The eigenvectors corresponding to σ_i, $\mathbf{v_i}$ give the equation

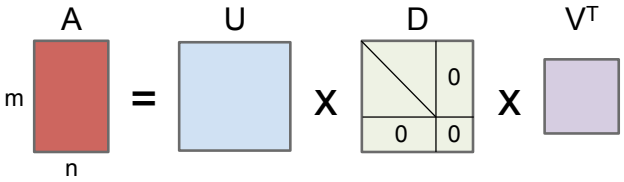
  $$\sigma_\mathbf{i} \mathbf{u_i} = \mathbf{A v_i}$$

  σs and vectors $\mathbf{v}$ should be in order largest to smallest.
- $\mathbf{U}$ Matrix of vectors normalized $\mathbf{u_i}$ from above.
- $\mathbf{D}$: Diagonal matrix of singular values of $\mathbf{A}$.

  $$\mathbf{D} = \begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & \sigma_n \end{bmatrix}$$

  Take care of the size of $\mathbf{D}$. If $\mathbf{A}$ is rectangular with more rows than columns, than the diagonal is filled up with zeros.
- $\mathbf{V}$: Matrix of eigenvectors of $\mathbf{A^T A}$ of size n:

  $$\mathbf{V} = (\mathbf{v_1}\ \mathbf{v_2} \ldots \mathbf{v_n})$$



For matrix $\mathbf{A}$

$$\mathbf{A} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 2 \end{bmatrix}$$

We have:

$$\mathbf{A^T A} = \begin{bmatrix} 2 & 2 \\ 2 & 5 \end{bmatrix}$$

σs are $\begin{bmatrix} \sqrt{6}, & 1 \end{bmatrix}$ which go in $\mathbf{D}$ matrix

$$\mathbf{D} = \begin{bmatrix} \sqrt{6} & 0 \\ 0 & 1 \end{bmatrix}$$

Matrix $\mathbf{U}$ has the $\mathbf{u}$ vectors:

$$\mathbf{U} = \begin{bmatrix} \frac{\sqrt{30}}{30} & -\frac{2\sqrt{5}}{5} \\ \frac{\sqrt{30}}{15} & \frac{\sqrt{5}}{5} \\ \frac{\sqrt{30}}{6} & 0 \end{bmatrix}$$

Finally, $\mathbf{V}$:

$$\mathbf{V} = \begin{bmatrix} \frac{\sqrt{5}}{5} & -\frac{2\sqrt{5}}{5} \\ \frac{2\sqrt{5}}{5} & \frac{\sqrt{5}}{5} \end{bmatrix}$$

Check $\mathbf{A} = \mathbf{UDV^T}$.

This code is not always correct!

```
A= Matrix([[1, 0], [0, 1], [1, 2]])

# Singular values and Matrix D
AtA= A.T * A
svv= sorted(AtA.eigenvects(), key= lambda x: x[0], reverse= True)
sigmas= [sqrt(x[0]) for x in svv]
atavecs= [x[2][0] for x in svv]
atavecs= [normalize(x) for x in atavecs]

U= []
D= []
V= []
for v, s in zip(atavecs, sigmas):
    if s == 0:
        continue
    u_i= A*v / s    # u_1 = A * v_i
    U.append(u_i)
    D.append(s)
    V.append(v)
U= Matrix(np.array(U)).T
D= diag(*D)
V= Matrix(np.array(V)).T

# Check decomposition
A == U * D * V.T
```

Interpreting the decompostion

Since a matrix $\mathbf{A}$ can be thought of a transformation, we can ask the question of what operations lead to such a transformation.

The SVD decomposes $\mathbf{A}$ into three simple transformations [4]:

[4] From Wikipedia Singular value decomposition

- An initial rotation $\mathbf{V^T}$. $\mathbf{V}$ is a symmetric matrix, so its action is to rotate.
- A scaling $\mathbf{D}$ along the coordinate axes. $\mathbf{D}$ is diagonal, so its action is to stretch or shrink, without rotating or shearing.
- A final rotation $\mathbf{U}$. The lengths σ_1 and σ_2 of the semi-axes are the singular values of M.

See also the applet at Transformation Matrix for some effects of matrix transformation

Appendix

Plotting with `matplotlib.pyplot`

The documentation of `matplotlib` and its modules is quite extensive but it doesn't give a simple overview of how graphics are organized. Therefore this section outlines the idea behind `pyplot` graphics and its components.

`pyplot` is a layer on `matplotlib` to provide graphic facilities similar to Matlab. `pyplot` appears to be the preferred way to plot graphics in `python/matplotlib`. See also matplotlib usage and the tutorial Getting Started With Matplotlib.

So, first import the `pyplot` module:

```
import matplotlib.pyplot as plt
```

and get some dummy data to play with:

```
import numpy as np

x= np.array([1, 2, 3, 4])
y= x**2
```

Here `x` and `y` are numpy arrays, although `pyplot` accepts any iterable.

Figure

```
fig= plt.figure()
```

`matplotlib.figure.Figure()` is the top level of a graphics where everything starts and it is therefore the equivalent of a blank sheet of paper where you draw the plot on. Use `figure` to set among other things the **size** of the figure (in inches) and the **dpi** resolution, if applicable. More or less the call `fig= plt.figure()` is equivalent to `R`'s `g<- ggplot()`. `figure()` can be called with only default parameters; in fact, a call to `Figure` can be skipped altogheter (see below).

Axis

```
axes= fig.add_subplot(1, 3, 2)
```

To draw on a `Figure` object you need to put a set of `Axes` where you actually draw stuff.

`Axes` can be put with `.add_subplot` or `.add_axes`. `add_subplot` is similar to `R par(mfrow= c(n, m))` in that it divides the figure in n rows and m columns. The third argument in `add_subplot` states which subplot should be drawn upon. *E.g.* `fig.add_subplot(1, 3, 2)` sets one row, three columns, and draws in the middle subplot (number 2).

Alternatively `.add_axes` can be used to specify the characteristics of the plotting box. It's roughly equivalent to `R par(mar= c(a, b, c, d))` in that you set the position of the axes relative to the overall figure. It can also be used to add plot on top of each other, like figure insets.

The `Axes` object can be used to control graphic parameters like x- and y-limits, the axis labels and the plot title. It controls more or less what you can control with `R`'s `par()`.

As for `Figure`, you can add a subplot with only default arguments or skip the call altogether.

Plotting

```
axes.plot(x, y)
```

Drawing is realized by adding stuff to the `Axes` object, typically by means of `plot()` function. Similar to `R plot()`, with `plot` you can set the plotting style (line, point), colour, etc. See pyplot.plot for documentation of setting line styles and plotting features.

Rendering

```
fig.savefig('filename.pdf')
fig.show()
plt.close()
```

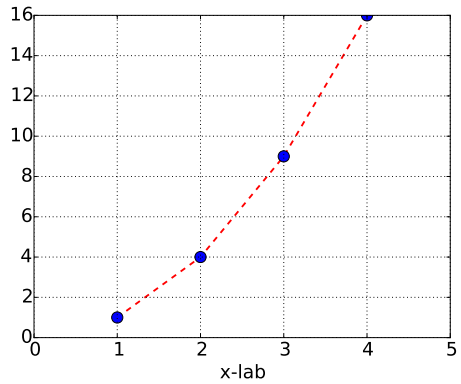
The actual plot is visualized with the `show()` method and saved to file with `savefig()` method. By default, file format is deduced from file extension. Finally, call `close` to close the grahic device(s).

In practice...

In practice you can take shortcuts to create plots. It is not strictly necessary to explicitly set up a `Figure` and `Axes` object.

```
plt.plot(x, y, 'r--', lw= 2)
plt.plot(x, y, 'bo', markersize=12)
plt.xlabel('x-lab')
```

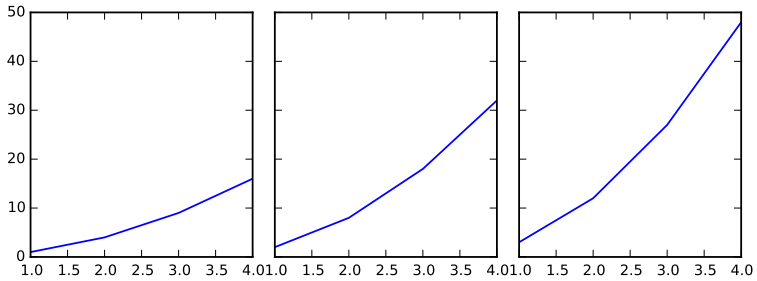
```
plt.xlim([0, 5])
plt.rc('font', size= 20)
plt.grid()
plt.savefig('figs/app_simple.pdf', bbox_inches= 'tight')
plt.show()
```



For simple plots, you can just call `plt.plot()` with the data to plot and optional graphic paramaters. The bare minimum to see an xy-plot is just `plt.plot(x, y); plt.show()`

Note that you keep adding features using `plt.<feature>`, then when you call `savefig` or `show` everything comes together. In contrast to `R`, the order with which you add these features doesn't matter. However, if you want to save to file *and* show on screen, remember to call `savefig` first and `show` then.

```
plt.rc('font', size= 8)
fig, axlst = plt.subplots(1, 3, sharex=True, sharey= True)
axlst[0].plot(x, y)
axlst[1].plot(x, y*2)
axlst[2].plot(x, y*3)
fig.subplots_adjust(wspace= 0.1)
fig.set_size_inches(18/2.54, 6/2.54)
fig.savefig('figs/app_subplot.pdf')
fig.show()
```



For multiple plots it might be best to set up the `Figure` object and the list (array actually) of subplots in one call to `plt.subplots()`. Each element of the array contains an `Axes` object that can be individually populated. The `Axes`'s

can be juxtaposed by playing with the method `Figure.subplots_adjust`. For example to set the spacing.

Example

```
fig= plt.figure()
ax1= fig.add_subplot(1, 3, 1)
ax1.plot(x, y, 'r--')
ax2= fig.add_subplot(1, 3, 2)
ax2.plot(x, y, 'pg')
ax3= fig.add_subplot(1, 3, 3)
ax3.plot(x, y, 'p')
for i, txt in enumerate(x):
    ax3.annotate(txt, (x[i], y[i]))
fig.show()
```

Principal components

...

For interpretation of eigenvalues and eigenvectors of a covariance matrix see http://www.visiondummy.com/2014/04/geometric-interpretation-covariance-matrix/

PageRank

Google ranks pages predominantly according to two criteria: 1) Number of links a page receives, *i.e.* many other pages rerfers to it, or in other words it is 'cited' very often; 2) Incoming links are not all equal, incoming links from important pages are more important, *i.e.* being 'cited' by a very important page counts more.

See also The Mathematics of Google Search

Stochastic matrix, *i.e.* where column sums equal 1, describing how node weights are distibuted.

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 1.0 & 0.5 \\ 0.3 & 0 & 0 & 0 \\ 0.3 & 0.5 & 0 & 0.5 \\ 0.3 & 0.5 & 0 & 0 \end{bmatrix} \tag{61}$$

- Each **column** describes how each node distributes its weight.

  For example, 1^{st} column, $[0\ \frac{1}{3}\ \frac{1}{3}\ \frac{1}{3}]^T$, says that node x_1 gives 0 of its weight to itself (of course), 1/3 to x_2, 1/3 to x_3, and 1/3 to x_4; 4^{th} column describes node x_4 and it says that 1/2 weight is given to x_1, 0 to x_2, 1/2 to x_3, and 0 to itself.

- Each **row** describes the weight or score of each node, since it is the SUM of all the node weights.

For example, $1st$ row is the score for node x_1. x_1 receives 0 from itself (of course), 0 of the weight of x_2, 1 (*i.e. all*) of the weight from x_3, and 1/2 of the weight from x_4.

So matrix $\mathbf{A}$ tells the *proportion* of weight to and from nodes. The question is: *How much each node weighs?* If all the nodes had the same weight the row sums of $\mathbf{A}$ would give the page rank already. However, we want to give more weight to pages that have many incoming links and/or incoming links from high ranking pages. For example, node x_1 receives all of the weight from x_3 (1) and half of the weight from x_4 (0.5). But how much do x_3 and x_4 weigh?

The assign weights to nodes we need to solve the linear system where the LHS is matrix $\mathbf{A}$ and RHS is the vector of weights (ranks) we want to find out:

$$\begin{bmatrix} 0x_1 + 0x_2 + 1x_3 + 0.5x_4 \\ 0.3x_1 + 0x_2 + 0x_3 + 0x_4 \\ 0.3x_1 + 0.5x_2 + 0x_3 + 0.5x_4 \\ 0.3x_1 + 0.5x_2 + 0x_3 + 0x_4 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \tag{62}$$

This is equivalent to finding the eigenvector of $\mathbf{A}$ for the eigenvalue 1. In fact see that if we set $\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \mathbf{u}$, we have in matrix notation:

$$\mathbf{Au} = \lambda\mathbf{u}; \quad and\ for\ \lambda = 1: \quad \mathbf{Au} = \mathbf{u}; \tag{63}$$

which is the definition of eigenvectors/values. Note that since $\mathbf{A}$ is stochastic the largest eigenvalue is always 1 [5].

In this example the eigenvector for $\lambda = 1$ is $\mathbf{u} = \begin{bmatrix} 1 \\ 0.33 \\ 0.75 \\ 0.5 \end{bmatrix}$. We can normalize the scores to sum to 1 and obtain the pageRank $PR = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 0.39 \\ 0.13 \\ 0.29 \\ 0.19 \end{bmatrix}$

Represent the web as a grpah and implement it as a dictionary of dictionaries. The outer dictionary has pages (nodes) as keys. The value of each key (page) is a dictionary of outgoing links. This inner dictionary has key: The arraival page, value: The weight transferred to that page.

```
graph= {
    1: {2: 0, 3: 0, 4: 0},
    2: {3: 0, 4: 0},
    3: {1: 0},
    4: {1: 0, 3: 0}
}
```

[5] see also proof that the largest eigenvalue of a stochastic matrix on Math Stack Exchange.

```
# Assign weights to pages. Outgoing links:
for p in graph:
    pout= graph[p]
    for w in pout:
        pout[w]= 1.0 / len(pout)

# Matrix representation
pages= sorted(graph.keys())
A= np.array(zeros(len(pages), len(pages)))
for ci in range(len(pages)):
    page= graph[pages[ci]]
    for ri in range(len(pages)):
        outk= pages[ri]
        if outk in page:
            A[ri][ci]= page[outk]
A= Matrix(A)

# Solve A * u = u in order to get the ranks u:
u= Matrix(symbols('x1:%s' %(A.cols+1)))
Au= Eq(A*u, u)
sols= solve(Au.subs(x1, 1), u[1:])
sols[x1]= 1

# Or get all eigenvects, but slower since it calculates all of them:
A.eigenvects()[0]

# Normalize ranks to sum to 1
s= sum(sols.values())
pageRank= {}
for x in sols:
    pageRank[x]= sols[x]/s

PR= Matrix([pageRank[x] for x in u])

# Test we did it right
Eq(A * PR, PR) # True
```

What if a page has no outgoing links? For example [6], in this graph page 4 has no outgoing links:

```
graph= {
    1: {2:0},
    2: {3: 0},
    3: {1: 0, 2: 0, 4: 0},
    4: {}
}
```

[6] From J. Hefferon - Linear Algebra, Topic: Page Ranking

In this case we can imaging that a surfer reaching page 4 will move at random to any other page. *I.e.* we distribute the weight of page 4 uniformaly across all the other pages:

```
for p in graph:
    pout= graph[p]
    if pout == {}:
        # If a page has no outgoing links distribute its weight across all web
        for x in graph:
            graph[p][x]= 1.0 / len(graph)
    else:
        for w in pout:
            pout[w]= 1.0 / len(pout)
        graph[p]= pout
```

This web looks like:

```
graph=
{1: {2: 1.0},
 2: {3: 1.0},
 3: {1: 0.33, 2: 0.33, 4: 0.33},
 4: {1: 0.25, 2: 0.25, 3: 0.25, 4: 0.25}}
```

As above, transform the graph in a stochastic matrix:

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0.33 & 0.25 \\ 1.0 & 0 & 0.33 & 0.25 \\ 0 & 1.0 & 0 & 0.25 \\ 0 & 0 & 0.33 & 0.25 \end{bmatrix}$$

```
pages= sorted(graph.keys())
A= np.array(zeros(len(pages), len(pages)))
for ci in range(len(pages)):
    page= graph[pages[ci]]
    for ri in range(len(pages)):
        outk= pages[ri]
        if outk in page:
            A[ri][ci]= page[outk]
A= Matrix(A)
```

Obtain eigenvector for $\lambda = 1$ and normalize to have ranks to sum to 1

$$PR = \begin{bmatrix} 0.16 \\ 0.32 \\ 0.36 \\ 0.16 \end{bmatrix}$$

Remember that there is an infinite number of eigenvectors belonging to an eigenvalue. In fact we talk about *eigenspace*. For convenience we choose the eigenvector whose sum is 1.

```
def rankMatrix(A):
    """Return page ranks for matrix of weights A"""
    u= Matrix(symbols('x1:%s' %(A.cols+1)))
    Au= Eq(A*u, u)
    Au= Au.subs(x1, 1) # This sub is arbitrary, any real will do.
    sols= solve(Au, u[1:])
    sols[x1]= 1

    s= sum(sols.values())
    pageRank= {}
    for x in sols:
        pageRank[x]= sols[x]/s
    PR= Matrix([pageRank[x] for x in u])
    return PR
```

Google edits the page rank matrix $\mathbf{A}$ to add some randomess in the behaviour of the surfer. That is, a surfer every now and then might jump to a page not linked in to the current one. The probability of jumping to an unlinked page is α, typically between 0.85 and 0.99. To model this behaviour we edit the weight matrix $\mathbf{A}$ as follow:

- Edit the weights in weight matrix $\mathbf{A}$ by multiplying by the correction factor α, $\mathbf{A_{rnd}} = \alpha\mathbf{A}$.
- Add to the corrected matrix $\alpha\mathbf{A}$ weights so that it returns to be stochastic, *i.e.* add $(1-\alpha)\mathbf{R}$ where $\mathbf{R}$ is a matrix of the same dim as $\mathbf{A}$ and with column sums 1.

Putting it all together, we obtain the *Google matrix* $\mathbf{G}$ with the *linear combination*:

$$\mathbf{G} = \alpha\mathbf{A} + (1-\alpha)\mathbf{R}$$

For this example:

$$\mathbf{G} = \alpha \begin{bmatrix} 0 & 0 & 0.33 & 0.25 \\ 1.0 & 0 & 0.33 & 0.25 \\ 0 & 1.0 & 0 & 0.25 \\ 0 & 0 & 0.33 & 0.25 \end{bmatrix} + (1-\alpha) \begin{bmatrix} 0.25 & 0.25 & 0.25 & 0.25 \\ 0.25 & 0.25 & 0.25 & 0.25 \\ 0.25 & 0.25 & 0.25 & 0.25 \\ 0.25 & 0.25 & 0.25 & 0.25 \end{bmatrix} =$$

$$\begin{bmatrix} 0 & 0 & 0.28 & 0.2125 \\ 0.85 & 0 & 0.28 & 0.2125 \\ 0 & 0.85 & 0 & 0.2125 \\ 0 & 0 & 0.28 & 0.2125 \end{bmatrix} + \begin{bmatrix} 0.0375 & 0.0375 & 0.0375 & 0.0375 \\ 0.0375 & 0.0375 & 0.0375 & 0.0375 \\ 0.0375 & 0.0375 & 0.0375 & 0.0375 \\ 0.0375 & 0.0375 & 0.0375 & 0.0375 \end{bmatrix} = \begin{bmatrix} 0.0375 & 0.0375 & 0.32 & 0.25 \\ 0.8875 & 0.0375 & 0.32 & 0.25 \\ 0.0375 & 0.8875 & 0.0375 & 0.25 \\ 0.0375 & 0.0375 & 0.32 & 0.25 \end{bmatrix}$$

The page ranks now become $PR_{\alpha=0.85} = \begin{bmatrix} 0.17 \\ 0.36 \\ 0.34 \\ 0.17 \end{bmatrix}$.

Note that if $\alpha = 1$ than there is no randomess at all. If $\alpha = 0$ instead the surfing is complete random and the page ranks become $PR_{\alpha=1} = \begin{bmatrix} 0.25 \\ 0.25 \\ 0.25 \\ 0.25 \end{bmatrix}$

```
def googleRank(A, alpha= 0.85):
    """Ranks pages in weight matrix A after having add some random surfing
    behaviour alpha"""
    R= Matrix(A.rows, A.cols, [1/A.cols] * A.rows * A.cols)
    G = alpha * A + (1 - alpha) * R
    PR= rankMatrix(G)
    return PR

googleRank(A, 0.85)
```

Line of best fit (least squares)

...