Department of Mathematics and Computer Science
University of Southern Denmark, Odense

March 8, 2018
Marco Chiarandini

# DM559 – Linear and integer programming

## Sheet 4, Spring 2018 [pdf format]

---

**Solution:**
Included.

### Exercise 1*

Calculate the angles of the triangle with sides $\mathbf{a}, \mathbf{b}, \mathbf{c}$ and show it is an isosceles right-angled triangle, where

$$\mathbf{a} = \begin{bmatrix} 1 \\ 2 \\ 2 \end{bmatrix}, \qquad \mathbf{b} = \begin{bmatrix} -1 \\ 1 \\ 4 \end{bmatrix}, \qquad \mathbf{c} = \mathbf{b} - \mathbf{a}$$

**Solution:**

$$\mathbf{a} = \begin{pmatrix} 1 \\ 2 \\ 2 \end{pmatrix}, \qquad \mathbf{b} = \begin{pmatrix} -1 \\ 1 \\ 4 \end{pmatrix}, \qquad \mathbf{c} = \mathbf{b} - \mathbf{a} = \begin{pmatrix} -2 \\ -1 \\ 2 \end{pmatrix}.$$

The cosines of the three angles are given by

$$\frac{\langle \mathbf{a}, \mathbf{b} \rangle}{\|\mathbf{a}\| \|\mathbf{b}\|} = \frac{-1+2+8}{\sqrt{9}\sqrt{18}} = \frac{1}{\sqrt{2}}$$

$$\frac{\langle \mathbf{a}, \mathbf{c} \rangle}{\|\mathbf{a}\| \|\mathbf{c}\|} = \frac{-2-2+4}{\sqrt{9}\sqrt{9}} = 0;$$

$$\frac{\langle \mathbf{b}, \mathbf{c} \rangle}{\|\mathbf{b}\| \|\mathbf{c}\|} = \frac{2-1+8}{\sqrt{18}\sqrt{9}} = \frac{1}{\sqrt{2}}.$$

Thus, the triangle has a right-angle, and two angles of $\pi/4$.

Alternatively, as the vectors $\mathbf{a}$ and $\mathbf{c}$ are orthogonal, and have the same length, it follows immediately that the triangle is right-angled and isosceles.

### Exercise 2*

How are these two lines (intersecting, parallel, skew)?

$$L_1: \quad \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \\ 4 \end{bmatrix} + t \begin{bmatrix} 1 \\ 2 \\ -1 \end{bmatrix}$$

$$L_2: \quad \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 5 \\ 6 \\ -5 \end{bmatrix} + t \begin{bmatrix} -2 \\ 1 \\ 7 \end{bmatrix}$$

Determine a plane that contains the two lines. Show then that the following is also an equation of the plane:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \\ 4 \end{bmatrix} + t \begin{bmatrix} 1 \\ 2 \\ -1 \end{bmatrix} + s \begin{bmatrix} -3 \\ -1 \\ 8 \end{bmatrix}.$$

**Solution:**

The lines intersect if there is a point $(x, y, z)$ on both lines; that is, if there exist values of the parameters, $s, t$ such that

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 1 \\ 3 \\ 4 \end{pmatrix} + t \begin{pmatrix} 1 \\ 2 \\ -1 \end{pmatrix} = \begin{pmatrix} 5 \\ 6 \\ 1 \end{pmatrix} + s \begin{pmatrix} -2 \\ 1 \\ 7 \end{pmatrix}.$$

Equating components, we need to solve the three simultaneous equations in two unknowns,

$$\left.\begin{aligned} 1 + t &= 5 - 2s \\ 3 + 2t &= 6 + s \\ 4 - t &= 1 + 7s \end{aligned}\right\} \Rightarrow \begin{aligned} 2s + t &= 4 \\ -s + 2t &= 3 \\ 7s + t &= 3. \end{aligned}$$

We have already seen in Example 1.53 that the first two equations have the unique solution, $s = 1,\ t = 2$. Substituting these values into the third equation,

$$7s + t = 7(1) + 2 \neq 3,$$

we see that the system has no solution. Therefore, the lines do not intersect and must be skew.

**Example 1.60** On the other hand, if we take a new line $L_3$, which is parallel to $L_2$ but which passes through the point $(5, 6, -5)$, then the lines

$$L_1 : \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 1 \\ 3 \\ 4 \end{pmatrix} + t \begin{pmatrix} 1 \\ 2 \\ -1 \end{pmatrix},$$

$$L_3 : \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 5 \\ 6 \\ -5 \end{pmatrix} + t \begin{pmatrix} -2 \\ 1 \\ 7 \end{pmatrix},\ t \in \mathbb{R}$$

do intersect in the unique point $(3, 7, 2)$.

**Activity 1.61** Check this. Find the point of intersection of the two lines $L_1$ and $L_3$.

**Example 1.63** You have shown that the lines $L_1$ and $L_3$ given in example 1.60 intersect in the point $(3, 7, 2)$. Two intersecting lines determine a plane. A vector equation of the plane containing the two lines is given by

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 3 \\ 7 \\ 2 \end{pmatrix} + s \begin{pmatrix} 1 \\ 2 \\ -1 \end{pmatrix} + t \begin{pmatrix} -2 \\ 1 \\ 7 \end{pmatrix}, \quad s, t \in \mathbb{R}.$$

Why? We know that $(3, 7, 2)$ is a point on the plane, and the directions of each of the lines must lie in the plane. As $s$ and $t$ run through all real numbers, this equation gives the position vector of all points on the plane. Since the point $(3, 7, 2)$ is on both lines, if $t = 0$ we have the equation of $L_1$, and if $s = 0$, we get $L_3$.

Any point which is on the plane can take the place of the vector $(3, 7, 2)^T$, and any non-parallel vectors which are linear combinations of $\mathbf{v}$ and $\mathbf{w}$ can replace these in the equation. So, for example,

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 1 \\ 3 \\ 4 \end{pmatrix} + t \begin{pmatrix} 1 \\ 2 \\ -1 \end{pmatrix} + s \begin{pmatrix} -3 \\ -1 \\ 8 \end{pmatrix}, \quad s, t \in \mathbb{R}$$

is also an equation of this plane.

## Exercise 3*

How many Cartesian equations would you need to describe a line in $\mathbb{R}^n$? And how many to describe an hyperplane?

**Solution:**

The vector equation of a line in $\mathbb{R}^n$ is:

$$\mathbf{x} = \mathbf{p} + t\mathbf{v} \qquad t \in \mathbb{R}$$

which requires one parameter, $t$. Component-wise, this leads to $n$ equations. Eliminating $t$ we are left with $n - 1$ Cartesian equations.

The vector eqaution of an hyperplane in $\mathbb{R}^n$ is:

$$\langle \mathbf{n}, \mathbf{x} \rangle = d$$

which gives a unique Cartesian Equation:

$$a_1 x_1 + a_2 x_2 + \ldots + a_n x_n = d$$

Alternatively, the vector equation of a hyperplane is

$$\mathbf{x} = \mathbf{p} + t_1\mathbf{v}_1 + t_2\mathbf{v}_2 + \ldots t_{n-1}\mathbf{v}_{n-1} \qquad t_1, t_2, \ldots, t_{n-1} \in \mathbb{R}$$

Hence, a vector parametric equation of a hyperplane in $\mathbb{R}^n$ would require $n - 1$ parameters (the $t_1, t_2, \ldots, t_{n-1}$ values.

## Exercise 4 Python: Geometric insight

Write a python function for each of the following specifications in $\mathbb{R}^3$:

a. takes as input the Cartesian equations of a line, $x = az + b$, $y = cz + d$ and returns the two arrays $\mathbf{p}$ and $\mathbf{q}$ of a vector equation $\mathbf{x} = \mathbf{p} + t\mathbf{q}$;

b. takes as input the vector equations of two lines and returns the intersecting point.

c. takes as input the vector equations of two lines and plots them in a 3D plot.

d. takes as input the vector equations of two lines and returns the Cartesian equation of the plane containing them.

e. takes as input the vector equations of two lines and plots the plane containing them in a 3D plot.

Assume the lines are actually intersecting and carry out the operations as much as possible in array form. If you encounter a system of linear equations to solve, $A\mathbf{x} = \mathbf{b}$, solve it by inverting the $A$ matrix, ie, $\mathbf{x} = A^{-1}\mathbf{b}$.
Try your program on

$$\mathbf{x} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} + t \begin{bmatrix} 2 \\ 1 \\ -1 \end{bmatrix}, \quad t \in \mathbb{R}$$

and the line expressed by the Cartesian equations,

$$x = 5, \quad y - 4 = \frac{z - 1}{2}$$

**Solution:**

http://www.imada.sdu.dk/~marco/DM559/Resources/Ipython/sh02-ex09.html

## Exercise 5

Write the *normal vector* of the line:

$$y = 3x - 4.$$

## Exercise 6

Show that the standard unit vectors in $\mathbb{R}^3$ denoted by:

$$\mathbf{i} = (1, 0, 0), \quad \mathbf{j} = (0, 1, 0), \quad \mathbf{k} = (0, 0, 1)$$

form an *orthogonal set*, which, since all vectors have unitary length is also called *othonormal set*.

## Exercise 7

Read the definition of the International Standard Book Number (ISBN) at http://mathworld.wolfram.com/ISBN.html.
Then, consider the following ISBN number: 0-13-512867-5. Does this number correspond to a possibly existing book or is it incorrect?

## Exercise 8

An image is an $M \times N \times 3$ array of RGB values, where $M$ and $N$ are the number of pixels in width and height. You can inspect this by yourself by loading your favourite image in jpeg or png format in Python with these lines of code:

```
from scipy import misc
l=misc.imread("/home/marco/Pictures/PIC710567.jpg")
import matplotlib.pyplot as plt
plt.imshow(l)
plt.show()
print type(l)
print l.shape
```

- You want to scale each color in the image by a different value. You try to multiply the image by a one-dimensional array with 3 values. What happens? Try to plot the result. Is a multiplication between a 3-D array (M,N,3) and a 1-D array (3) mathematically defined? How do you explain the result?

- Make a grey level version of the image. This can be achieved by averaging the color channels by `mean(l,axis=2)`. Explain what happens.

- Crop the part of the image outside a centered circle. Given the array representation use advanced indexing by creating a mask that will let you set to black only the pixels outside of the circle. (Try with a more simple shape to start with.)

**Solution:**

http://scipy-lectures.github.io/advanced/image_processing/

## Exercise 9 Vector Quantization Algorithm

Broadcasting happens in the vector quantization (VQ) algorithm used in information theory, classification, and other related areas. The basic operation in VQ finds the closest point in a set of points, called codes in VQ jargon, to a given point, called the observation.
In a very simple two-dimensional case, the values in observation describe the weight and height of an athlete to be classified. The codes represent different classes of athletes. Plot the codes and the observation in a scattered plot. Find then the closest point by calculating the distance between observation and each of the codes. The shortest distance provides the best match.

```
observation = np.array([111.0,188.0])
codes = np.array([[102.0, 203.0], # basketball player
              [132.0, 193.0], # football lineman
              [45.0, 155.0], # female gymnastic
              [57.0, 173.0]]) # marathon runner
```

However for higher dimensions and large datasets broadcasting becomes inefficient and it may result in lower performance.

**Solution:**

```
# here is the broadcast
diff = codes - observation
dist = sqrt(sum(diff**2,axis=-1))
nearest = argmin(dist)
```

## Exercise 10 Outer Product Applications

In class, we saw that the matrix multiplication between two vectors is possible if we transpose the first of the two vectors, such that we have a multiplication between a row vector and a column vector. Let $\mathbf{x} = [x_1, x_2, \ldots, x_n]^T$ and $\mathbf{y} = [y_1, y_2, \ldots, y_n]^T$ be column vectors, ie, $1 \times n$ matrices, then the matrix multiplication of:

$$\mathbf{x}^T\mathbf{y} = \begin{bmatrix} x_1 & x_2 & \ldots & x_n \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \ldots \\ y_n \end{bmatrix} = x_1y_1 + x_2y_2 + \cdots + x_ny_n$$

We have called this the *inner product* or *scalar product* or *dot product* of two vectors.
In class we saw also the matrix multiplication of a column vector times a row vector. It gives rise to what is called the *outer product* of vectors and it is represented with the symbol $\otimes$. Let $\mathbf{x}$ be an $(1 \times m)$

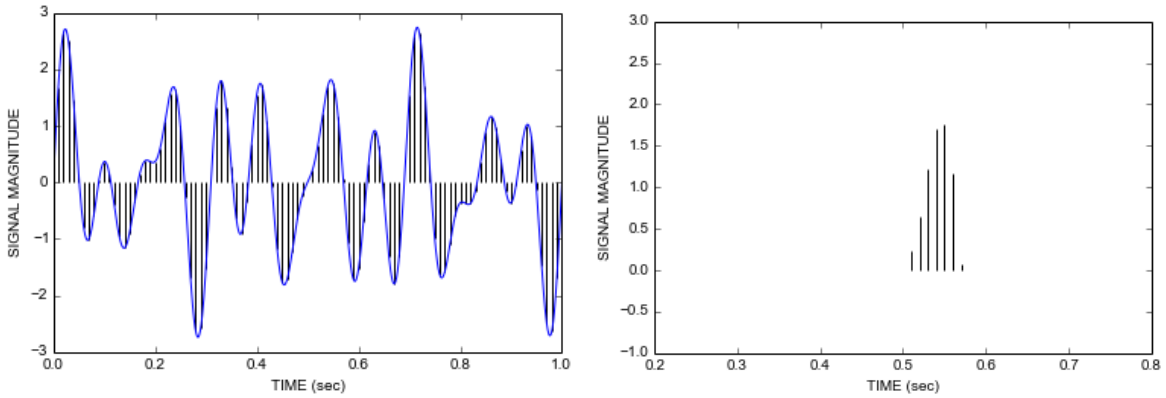


Figure 1:

column matrix and $\mathbf{y}$ a $(1 \times n)$ column matrix (ie, two column vectors).

$$\mathbf{x} \otimes \mathbf{y}^T = \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_m \end{bmatrix} \begin{bmatrix} y_1 & y_2 & \dots & y_n \end{bmatrix} = \begin{bmatrix} x_1y_1 & x_1y_2 & \cdots & x_1y_n \\ x_2y_1 & x_2y_2 & \cdots & x_2y_n \\ \vdots & \vdots & \ddots & \vdots \\ x_my_1 & x_ny_2 & \cdots & x_my_n \end{bmatrix}$$

Thus every row of the new matrix is a multiple of the vector $[y_1, y_2, \dots, y_n]$ and every column a multiple of the vector $[x_1, x_2, \dots, x_m]$.

- Consider a website for film reviews. In a simplified version, users can vote +1 if they like a movie, -1 if they do not like it and 0 if they are indifferent. For a group of users that watched the same set of movies, how could you use the dot product to understand which users had the most similar evaluations? Try on the following example that generates a matrix with 5 random column vectors, one per user, for 30 movies. Which users have the highest concordance in their evaluations? As always try to do this in one line code. (You should obtain that users 0 and 1 are those most concordant)

```
import numpy as np
np.random.seed(4)
np.random.choice([-1,0,1],(30,5))
```

**Solution:**

```
{(i,j):np.dot(X[:,i],X[:,j]) for (i,j) in itertools.product(range(5),range(5)) if i>j
    }
```

- In signal processing or other applications like mass spectrometry there is sometimes the need to align a sub-pattern with the full spectrum. For example, the sound wave depicted in Figure 1, left, is sampled at regular intervals. We want then to find out from which time interval the sub-pattern depicted in the right figure is most likely to have been sampled. How could you do this task with the dot product? Would it be efficient? Try your method on the data from the picture.

```
import numpy as np
import matplotlib.pyplot as plt

# the signal
t = np.linspace(0, 1, 1001)
y = np.sin(2*np.pi*t*6) + np.sin(2*np.pi*t*10) + np.sin(2*np.pi*t*13)

# the sampled signal
```

```
s=np.arange(1000,step=10)
t_sampled = t[s]
y_sampled = y[s]

# the figure
fig=plt.figure()
plt.plot(t, y, 'b-')
plt.xlabel("TIME (sec)")
plt.ylabel("SIGNAL MAGNITUDE")
plt.vlines(t_sampled,[0],y_sampled)
plt.show()

# the subpattern
#pattern = np.array([ 0.22694118, 0.63757689, 1.21845958, 1.71114521, 1.76007351,
    1.16501524, 0.06652395])
s1=np.arange(100)[(t_sampled>0.45) & (t_sampled<0.6) & (y_sampled>=0)]
pattern = y_sampled[s1]

plt.vlines(t_sampled[s1],[0],pattern)
plt.gca().set_xlim([0.2,0.8])
plt.gca().set_ylim([-1,3]);
plt.xlabel("TIME (sec)")
plt.ylabel("SIGNAL MAGNITUDE")
```

**Solution:**

We can solve by calculating the dot product with respect to every starting interval:

```
similarity = { i : np.dot(pattern,y_sampled[i:i+7]) for i in range(len(y_sampled)-6)
    }
max(similarity,key=similarity.get)
```

From here we find that the largest similarity (largest value for the scalar product is with the signal that start at instant 68.

The following exercises are not relevant for the obligatory assignments and the written exam.

## Exercise 11 Generating Matrices of a Given Rank

In this exercise we consider how to use Python to generate matrices with specified ranks.

- In general, if $A$ is an $m \times n$ matrix with rank $r$, then $r \leq \min(m, n)$. Why? Explain. If the entries of $A$ are random numbers in $[0, 1)$, we would expect that $r = \min(m, n)$. Why? Explain. Check out this generating random $6 \times 6, 8 \times 6$, and $5 \times 8$ matrices using `random.rand` and `numpy.linalg.matrix_rank` (to not confuse with `numpy.rank`). Whenever the rank of an $m \times n$ matrix equals $\min(m, n)$, we say that the matrix has *full rank*. Otherwise we say that the matrix is *rank deficient*.

- Python's `numpy.random.randint` can be used to generate random $m \times n$ matrices with integer entries in a given range $[a, b]$. Generate in this way a matrix $6 \times 8$ whose entries are random integers in the range $[3, 7]$. Using the range $[1, 10]$, create random integer $10 \times 7$, $8 \times 12$, and $10 \times 15$ matrices and in each case check the rank of the matrix. Do these integer matrices all have full rank?

  **Solution:**

```
np.linalg.matrix_rank(np.random.rand(8,6))
np.random.randint(3,7,(8,6))
```

- Suppose we want to use Python to generate matrices with less than full tank. It is easy to generate matrices of rank 1. If $\mathbf{x}$ and $\mathbf{y}$ are nonzero vectors in $\mathbb{R}^m$ and $\mathbb{R}^n$, respectively, then $A = \mathbf{x} \otimes \mathbf{y}^T$ will be an $m \times n$ matrix with rank 1. Why? Explain. Verify this in Python generating a matrix $A$ from two vectors of random integers from $[1, 10]$ and size 8 and 6. Use the function `numpy.outer` for the outer product. Check that the matrix has rank 1.

  **Solution:**

  $A = \mathbf{x} \otimes \mathbf{y}^T$ corresponds to a matrix with columns made by $\mathbf{x}$ each multiplied with a different scalar derived from the corresponding element of $\mathbf{y}$. Hence all columns are scalar product of the same vector, hence the rank is 1.

```
x=np.random.randint(1,10,8)
y=np.random.randint(1,10,6)
np.outer(x,y)
```

- In general

$$\text{rank}(AB) \leq \min(\text{rank}(A), \text{rank}(B))$$

  If you have not done it yet. If $A$ and $B$ are noninteger random matrices, the relation above should be an equality. Generate an $8 \times 6$ matrix $A$ by setting:

```
X=numpy.random.rand(8,2)
Y=numpy.random.rand(6,2)
A=numpy.dot(X,Y.T)
```

  What would you expect the rank to be? Explain. Test the rank of $A$ with Python.

  **Solution:**

```
np.linalg.matrix_rank(A) # 1
```

- Use Python to generate matrices $A$, $B$ and $C$ such that:

- $A$ is $8 \times 8$ with rank 3
- $B$ is $6 \times 9$ with rank 4
- $A$ is $10 \times 7$ with rank 5

**Solution:**

The important idea is to use matrix multiplication of matrices of appropriate dimensions that are full rank. It would be easier using random numbers from $[0, 1)$. However if the set of numbers from which we sample is large enough it succeeds also with integer numbers. For example:

```
X=np.random.randint(1,10,(3,8))
Y=np.random.randint(1,10,(8,4))
A=np.dot(X,Y)
np.linalg.matrix_rank(A) # 3
```

```
X=np.random.randint(1,10,(6,4))
Y=np.random.randint(1,10,(4,9))
A=np.dot(X,Y)
np.linalg.matrix_rank(A) # 4
```

```
X=np.random.randint(1,10,(10,5))
Y=np.random.randint(1,10,(5,7))
A=np.dot(X,Y)
np.linalg.matrix_rank(A) # 5
```

## Exercise 12 Solving Linear Systems in Python

Let $A$ be an $n \times n$ matrix and $b_1, b_2, \ldots, b_k$ a sequence of $n$-vectors. We consider to find $k$ vectors $x_i$ such that

$$Ax_i = b_i.$$

LU factorization is a way to organize the Gaussian elimination method such that the computation is done in two steps:

- A factorization step of the matrix $A$ to get matrices of triangular form
- A relatively cheap backward and forward elimination step which works on $b_i$ and that benefits from the more time consuming factorization step.

In the first step the LU factorization finds a permutation matrix $P$, a lower triangular matrix $L$ and an upper triangular matrix $U$, such that,

$$PA = LU \qquad \text{and} \qquad A = PLU$$

Such a factorization always exists. How it can be carried out is explained in the slides linked from the Lec. 6. Furthermore, $L$ can be determined in such a way that $L_{ii} = 1$. Thus, the essential data from $L$ which has to be stored is $L_{ij}$ with $i > j$. Consequently, $L$ and $U$ can be stored together in an $n \times n$ array, while the information about the permutation matrix $P$ just requires an $n$ integer vector - the pivoting vector.
In SciPy there are two methods to compute the $LU$ factorization. The standard one is `scipy.linalg.lu`, which returns the three matrices $L, U$ and $P$. Another method is `lu_factor`:

```
import scipy.linalg as sl
[LU, pv]=sl.lu_factor(A)
```

Here the matrix $A$ is factorized and an array with the information about $L$ and $U$ is returned together with the pivot vector.
With this information the system can be solved by performing row interchanges of the vector $b_i$ according to the information stored in the permutation vector, backward substitution using $U$ and finally forward substitution using $L$. This is bundled in Python in the method using `lu_solve`. For example:

```
x = sl.lu_solve((LU, pv), b)
```

Try these methods to solve the linear systems seen in the slides or those from the exercises of Sheet 5. Try once to retrieve `x` after `lu_factor` but without using `lu_solve`.

## Exercise 13 Sparse Matrices

Some advanced applications in mathematics and physics need to handle matrices of huge size, eg:

- discretization of (partial) differential equations
- finite element methods
- discrete laplacians
- analysis of large networks, such as, the Internet.

In this exercise we compare the performance of different mathematical softwares, namely: MatLab, Octave, Maple and R. Your task is to collect the results for Python and compare them with those here reported. The common tasks that these solvers will have to perfom is solving a system of linear equations in the best way they have implemented. We use the following data in MatLab format: TestA.mat, Testb.mat.

**MatLab**

```
tic, load TestA;
load Testb; toc
tic, c=A\b; toc
```

```
>> whos
  Name Size Bytes Class Attributes
  A 1000000x1000000 51999956 double sparse
  b 1000000x1 8000000 double
  c 1000000x1 8000000 double
```

```
Elapsed time is 0.191414 seconds.
Elapsed time is 0.639878 seconds.
```

**Octave**

```
octave:1> comparison
Elapsed time is 0.276378 seconds.
Elapsed time is 0.618884 seconds.
```

**MAPLE**

```
> A:=ImportMatrix("TestA.mat",source=MATLAB);
memory used=72.8MB, alloc=72.9MB, time=0.51
memory used=122.8MB, alloc=122.8MB, time=0.59
...
memory used=1465.8MB, alloc=679.2MB, time=58.95
memory used=1546.9MB, alloc=743.1MB, time=69.12
                                        [ 1000000 x 1000000 Matrix ]
                                A := ["A", [ Data Type: float[8] ]]
                                        [ Storage: sparse ]
                                        [ Order: Fortran_order ]
> b:=ImportMatrix("Testb.mat",source=MATLAB);
memory used=1621.2MB, alloc=743.1MB, time=76.11
                                          [ 1000000 x 1 Matrix ]
                                  b := ["b", [ Data Type: float[8] ]]
                                          [ Storage: rectangular ]
                                          [ Order: Fortran_order ]
```

```
> with(LinearAlgebra):
> c:=LinearSolve(A,b);
Error, (in simplify/table) dimensions too large
```

R

```
library(R.matlab)
library(Matrix)
S<-readMat("sparse.mat")
b<-readMat("Testb.mat")

A <- sparseMatrix(S$i,S$j,S$k)

system.time(try(solve(A,b)))

Am <- as.(A,"matrix")
# fails
# Error in asMethod(object) :
# Cholmod error 'problem too large' at file ../Core/cholmod_dense.c, line 105
```

Python

```
import scipy.io
mat = scipy.io.loadmat('TestA.mat')
A=mat['A']
mat = scipy.io.loadmat('Testb.mat')
b=mat['b']

import scipy.sparse.linalg as ssl
ssl.spsolve(A,b,use_umfpack=False)
```

## Exercise 14 Broadcasting

For each of the following operations say: i) whether they are mathematically defined ii) whether they will work or fail with an error, iii) if they work say whether broadcasting occurs and how.

```
# Example 1
a = array([1.0,2.0,3.0])
b = array([2.0,2.0,2.0])
a * b
```

```
# Example 2
a = array([1.0,2.0,3.0])
b = 2.0
a * b
```

```
# Example 3
a = array([[ 0.0, 0.0, 0.0],
           [10.0,10.0,10.0],
           [20.0,20.0,20.0],
           [30.0,30.0,30.0]])
b = array([1.0,2.0,3.0])
a + b
```

**Solution:**

http://wiki.scipy.org/EricsBroadcastingDoc?highlight=%28broadcasting%29

```
# Example 4
a = np.ones(3,4)
b = np.array([1,2,3,4])
a + b
```

**Solution:**

From a mathematical point of view, the sum between a matrix of size $m \times n$ and a row vector of size $n$ is not defined. However, in python the expression produces a result. Reshaping to (1,4) then extending to (3,4).