To solve the numerical problems you will have to include the scitools-package (which most of you know from INF1100) in your programs. This package contains the numpy-module, math-module, plot-module etc. If you want to install this package on your personal computer, go to the web-page http://www.ifi.uio.no/∼inf3330/software/. To inlcude all functions in scitools, write the following line in the beginning of your python-program;

```
from scitools.all import *
```

To solve the numerical problems you will often have to read in data from file, write data to file, and plot results. The following standard python/scitools methods might be very useful.

Read data from file

```
#In this case, the data is written in columns. One column for each data type.
def read_data(x0, x1, ..., xN):
   file = open('filename', 'r')   #Open the file, r stands for read
   for line in file:              #For-loop: Go through every line, one by one
      data = line.split()         #Split line in columns and store in array
      x0.append(float(data[0]))   #Store the values in lists
      ...
      ...
      xN.append(float(data[N]))

   file.close()                   #Close the file
   x0 = array(x0)                 #Convert from list to array
   ...
   xN = array(xN)
   return x0, ... , xN            #Return arrays
```

Write data to file

```
#In this case, the arguments are arrays of equal length
def write_file(x0, ... , xN):
   file = open('filename', 'w')        #'w' - overwrite, 'a' - append
   for i in range(len(x0)):
       #Writecommand, here floatnumbers with 3 decimals, \n = lineshift
       file.write('%.3f (...) %.3f \n' % (x0[i], ... , xNi[i]))
   file.close()
```

Plot

```
def plot_function(x, y):
   plot(x,y)                             #Plot-command, x and y are arrays
   xlabel('...')                         #Label x-axis
   ylabel('...')                         #Label y-axis
   title('...')                          #Title
   axis([xmin,xmax,ymin,ymax])           #Axis length (if you want to define)
   hardcopy('filename.eps', color=True)  #Make an eps-file of the plot
```

Using scipy arrays (vectors) in python

```
from scipy import *
```

```
# Read a table of data from a file. It returns a two-dimensional scipy array
# with the numbers from the file.
def read_table(file):
return array([[float(w) for w in line.split()] for line in open(file,"r")])
# To use this to read data from file "data.txt", I would do:
data = read_table("data.txt")

# Scipy arrays can be easily sliced. So if the first column of my file is
# the time, and I want that alone in its own array, I would do
time = data[:,0]

# The good thing about scipy arrays is that you can avoid lots of loops with them.
# For example, if I want a new array of the sin of all of the times in my "time"
# array, I could do something like a = array([sin(t) for t in time]), or the
# equivalent but less elegant
#   a = []
#   for t in time: a.append(sin(t))
#   a = array(a)
# but a much smarter choice would be to just do
a = sin(time)

# I could have done this directly, of course, without going through the
# intermediate variable time:
a = sin(data[:,0])

# You can do pretty much whatever you would expect to with these. The example
# below calculates the sum of the squares of the difference between our
# array a and the second column of the data file we read in.
chisq = sum((a-data[:,1])**2)

# These arrays also let you work with vectors. Let's say you have n particles
# in m dimensions, so that each particle would have coordinates x, y, and z if
# m = 3, for example. Instead of actually having variables like x1, y1, z1
# for particle 1, x2, y2, z2 for particle 2 and so on, which would become
# very unwieldy, you can make a positions array like this for 1000 particles
# in 2 dimensions
n = 1000
m = 2
positions = zeros([n, m])

# The x-position of the first particle would then be positions[0,0].
# The distance between the first and second particle would be
# (using Pythagoras):
dist = sum((positions[0,:] - positions[1,:])**2)**0.5
# As you can see, there is no reason not to use vectors.

# Here is a bonus iterator, which you'll have to figure out what the
# point of is yourselves:
def grid(lens):
i, it = 0, array(lens)*0
while i < len(it):
yield it
i = 0
while i < len(it):
```

```
it[i] += 1
if it[i] < lens[i]: break
it[i], i = 0, i+1
```