1 Some overview of what I have learned in summer

This notebook only includes most interesting programms what I have made. It assumes that packages like:

- matplotlib
- numpy
- visual

have been installed before running notebook.

1.1 Solar system (3.4)

```
# -*- coding: utf-8 -*-
"""
Created on Sun Aug 11 21:21:27 2013

@author: akels
"""
from __future__ import division, print_function
from visual import sphere, display, color, rate
from cmath import exp
from math import pi

sphere(pos=(0,0,0),radius=40,color=color.yellow)

c1 = 2e-3
c2 = 0.5

class planet:

       def __init__(self,r_radius,r_orbit,period):
       #self = sphere()

              self.sphere = sphere()
              print("Object {} created".format([r_radius,r_orbit,period]))

              self.T = period
              self.r_orbit = r_orbit #108.2e6/100
              self.sphere.radius = r_radius #6052

              self.move(0)

       def move(self,t):

              omega = 2*pi/self.T
              teta = omega*t

              pos = self.r_orbit*exp(1j*teta)
              self.sphere.pos = [pos.real,pos.imag,0]
```

```
table = [
       [2440, 57.9,  88, color.red],
       [6052, 108.2, 224.7, color.magenta],
       [6371, 149.6 , 365.3, color.blue],
       [3386, 227.9, 687.0, color.orange],
       [69173, 778.5, 4331.6,color.magenta],
       [57316, 1433.4, 10759.2,color.green]]

planets = []
for r_radius,r_orbit,period,color in table:
       s = planet(r_radius*c1,r_orbit,period/c2)
       s.sphere.color = color
       planets.append(s)

d = display()
d.autoscale = False

t = 0
while True:
       rate(30)
       for i in planets:
              i.move(t*c2)
       t+=1
```

```
---------------------------------------------------------------------------
ImportError                               Traceback (most recent call last)

<ipython-input-1-208364024e5d> in <module>()
      6 """
      7 from __future__ import division, print_function
----> 8 from visual import sphere, display, color, rate
      9 from cmath import exp
     10 from math import pi


ImportError: No module named 'visual'
```

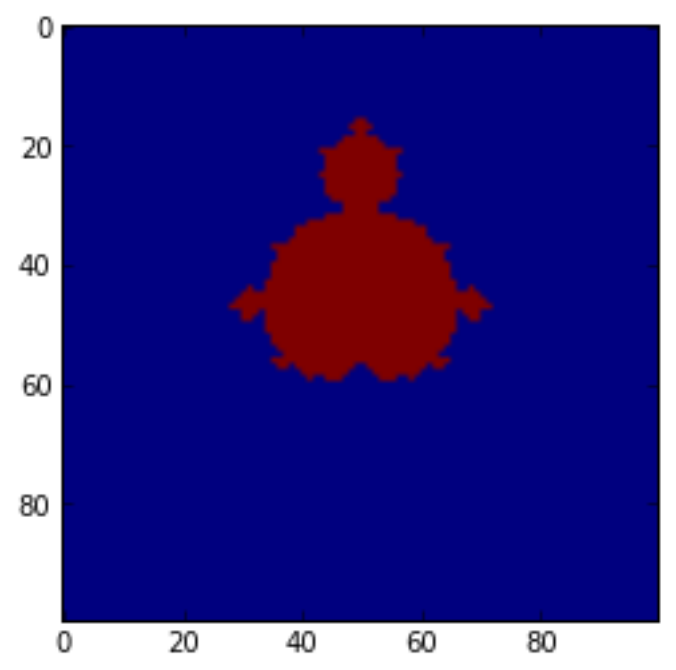
1.2 The Mandelbrot set (3.6)

```
# -*- coding: utf-8 -*-
"""
Created on Mon Aug 5 11:39:10 2013

@author: akels
"""

from numpy import *
from pylab import imshow
```

```
x,y = mgrid[-2:2:100j,-2:2:100j]

c = x + y*1j

def mandelbort(c,n=100):

        z = 0
        for i in range(n):
                z = c + z**2

        z_mod = sqrt(z * z.conjugate())

        return z_mod<2

imshow(mandelbort(c))
```

```
-c:19: RuntimeWarning: overflow encountered in square
-c:19: RuntimeWarning: invalid value encountered in square
-c:21: RuntimeWarning: invalid value encountered in multiply
```

```
<matplotlib.image.AxesImage at 0xb1a61c8c>
```

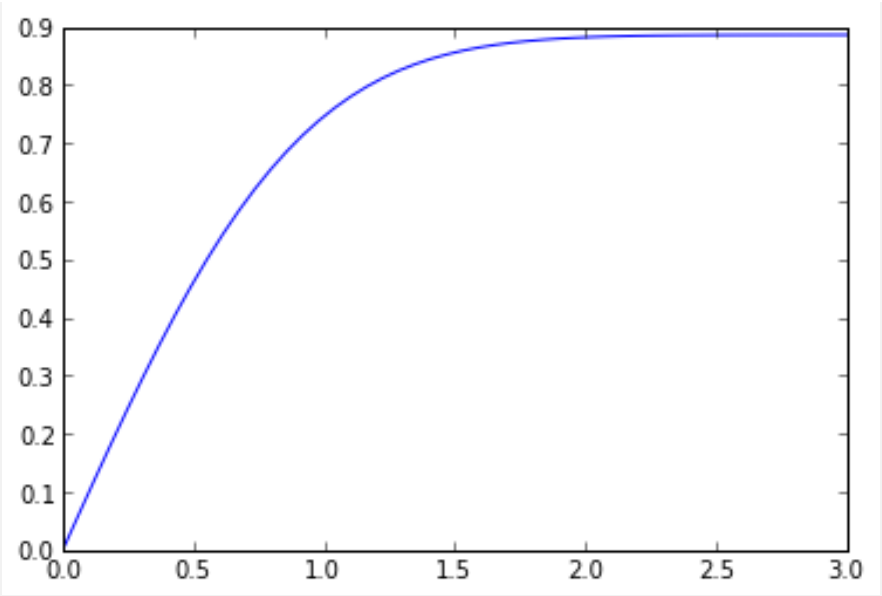


1.3 Integral evaluation $\int_0^x e^{-t^2} dt$ (5.3)

```
# -*- coding: utf-8 -*-
"""
Created on Tue Aug 6 08:03:08 2013
```

```
@author: akels
"""
from __future__ import division, print_function
from numpy import *

f = lambda x: exp(-x**2)

def integrate(f,a,b):

        h = 0.1

        N = int((b-a)/h)


        delta = b-a - N*h

        I = 0

        if N>0:
                s = (f(a) + f(b-delta))/2

                for k in range(1,N):
                        s+= f(a+k*h)

                I = s*h

        I +=  delta*(f(b) + f(b-delta))/2

        return I

x = linspace(0,3)
E = [integrate(f,0,xi) for xi in x]

#E = integrate(f,0,x)

plot(x,E)
#plot(x,f(x))
```

```
[<matplotlib.lines.Line2D at 0xb19bf04c>]
```

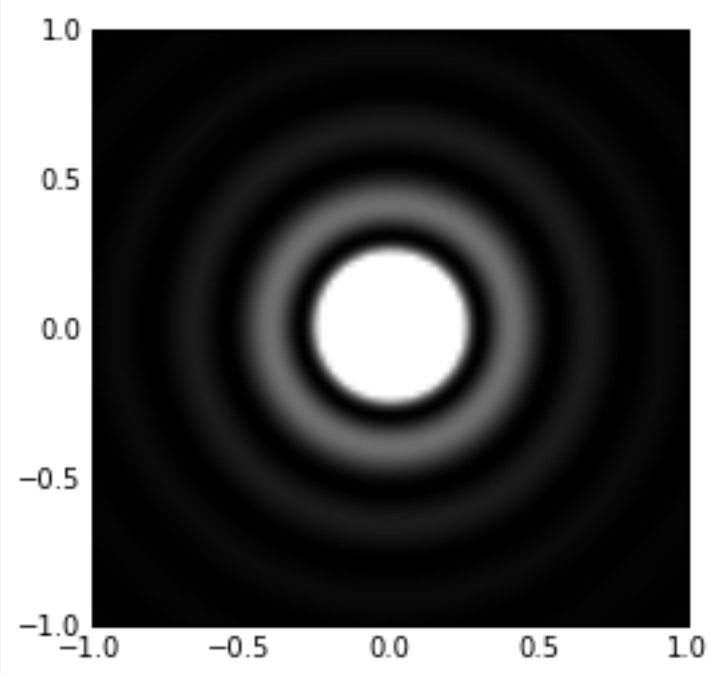


1.4 Circular holes diffraction pattern (5.4)

```
# -*- coding: utf-8 -*-
"""
Created on Tue Aug 6 08:57:06 2013

@author: akels
"""

from __future__ import division, print_function

from numpy import cos,sin,pi,sqrt
from pylab import *

def J(m,x):

        def f(teta):
            return cos(m*teta - x*sin(teta))

        N = 1000
        a = 0.
        b = pi
        h = (b-a)/N

        s = f(a) + f(b) + 4*f(b-h)
        for k in range(1,N//2):
            s += 4*f(a + (2*k-1)*h) + 2*f(a+2*k*h)

        I = h/3*s/pi

        return I
```

```
x,y = mgrid[-1:1:100j,-1:1:100j]
r = sqrt(x**2 + y**2)
wavelength = 0.5
k = 2*pi/wavelength

I = (J(1,r*k)/k/r)**2
gray()
imshow(I,vmax=0.1/10,extent=(-1,1,-1,1))
```

```
<matplotlib.image.AxesImage at 0xb1a3002c>
```

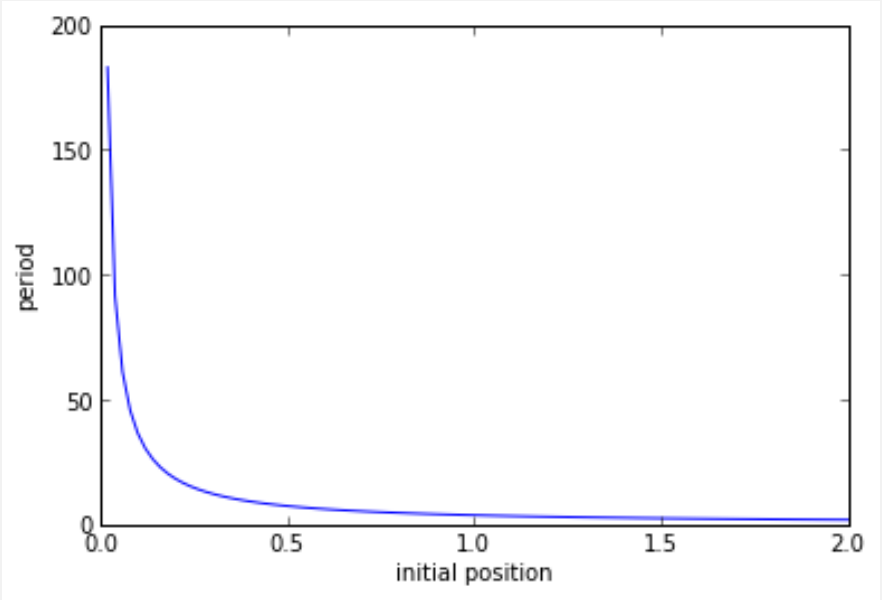


1.5 Period of an anaharmonic oscillator (5.10)

```
# -*- coding: utf-8 -*-
"""
Created on Tue Aug 6 21:25:08 2013

@author: akels
"""

from __future__ import division, print_function

from os import sys
from numpy import linspace
from pylab import *
sys.path.append('cpresources')
from gaussxw import gaussxw

N = 100
```

```
m = 1
V = lambda x: x**4

x,w = gaussxw(N)


def T(a_): # Doing the integration with gaussian quadrature

        a = 0
        b = a_
        xp = 0.5*(b-a)*x + 0.5*(b+a)
        wp = 0.5*(b-a)*w
        E = V(a_)

        y = 1/sqrt(E-V(xp))
        s = sum(y*wp)

        return s*sqrt(8*m)

a = linspace(0,2,100)
periods = [T(ai) for ai in a]

plot(a,periods)
xlabel('initial position')
ylabel('period')
```

```
<matplotlib.text.Text at 0xb0fe4f8c>
```

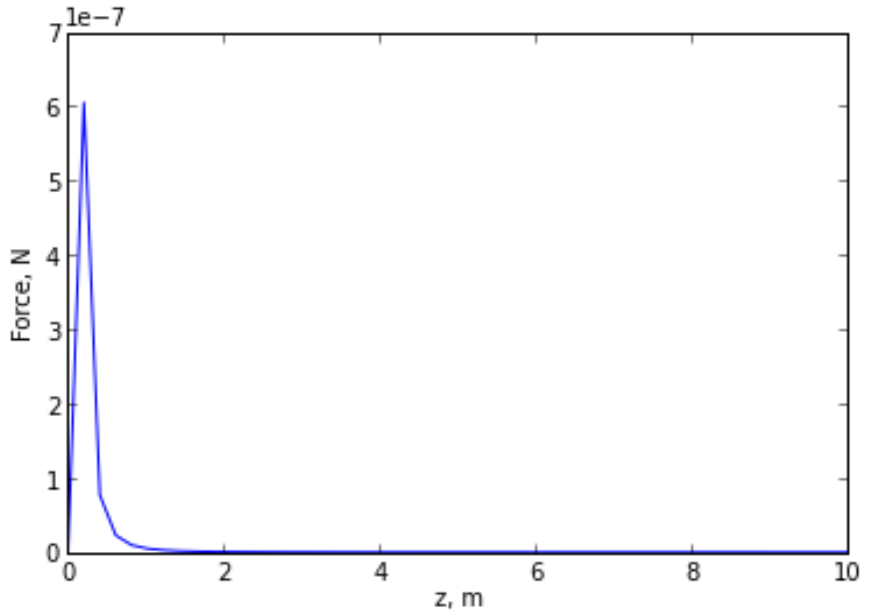


1.6 Gravitation pull of a uniform sheet (5.13)

```
# -*- coding: utf-8 -*-
"""
```

```
Created on Wed Aug 7 10:41:43 2013

@author: akels
"""

from __future__ import division, print_function
from os import sys
from pylab import plot, xlabel,ylabel,linspace
sys.path.append('cpresources')
from gaussxw import gaussxw


G = 6.674e-11
m = 1
ro = 100
f = lambda x,y,z: (x**2 + y**2 + z**2)**-3/2

N = 100
x,w = gaussxw(N)

def force(z):

       a = -5
       b = 5
       xp = 0.5*(b-a)*x + 0.5*(b+a)
       yp = xp

       wp = 0.5*(b-a)*w

       s=0
       for i in range(N):
              for j in range(N):
                     s+=wp[i]*wp[j]*f(xp[i],yp[j],z)

       F = G * m * ro * z *s
       return F

z = linspace(0,10,50)
F = [force(zi) for zi in z]
plot(z,F)
xlabel('z, m')
ylabel('Force, N')
```

```
<matplotlib.text.Text at 0xb0f8c32c>
```

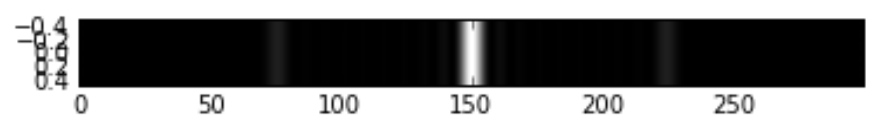


1.7 Difraction grating (5.2)

```
# -*- coding: utf-8 -*-
"""
Created on Wed Aug 7 23:19:31 2013

@author: akels
"""
from __future__ import division, print_function
from os import sys
sys.path.append('cpresources')
from pylab import *
from cmath import exp

d = 20e-6
alpha = pi/d
q = lambda u: sin(alpha*u)**2

w = 10 * d
wavelength = 500e-9
focus = 1
k = 2*pi/wavelength


def I(x):

        f = lambda u,x: sqrt(q(u))*exp(1j*k*u * x/focus)

        N = 100
        a = -w/2
```

```
        b = w/2
        h = (b-a)/N

        s = 0.5*f(a,x) + 0.5*f(b,x)
        for i in range(1,N):
            s += f(a+i*h,x)

        return h**2 * (real(s)**2 + imag(s)**2)

x = linspace(-0.05,0.05,300)
intensity = [I(xi) for xi in x]

imshow([intensity],aspect=25)
```

```
<matplotlib.image.AxesImage at 0xb0f4ebcc>
```

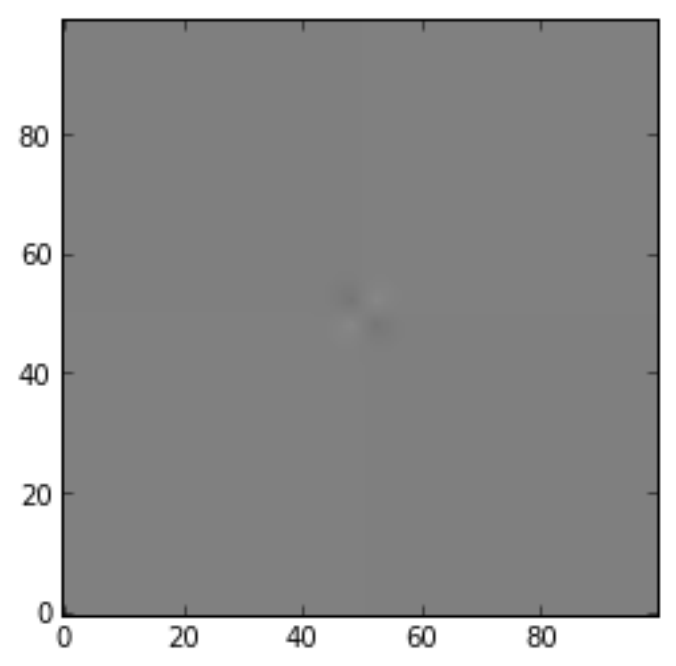


1.8 Potential determination with double integration

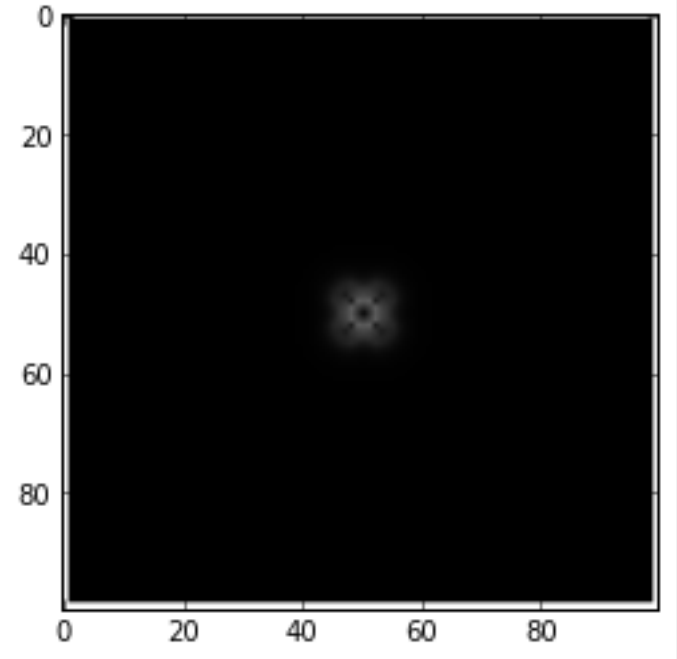
```
# -*- coding: utf-8 -*-
"""
Created on Thu Aug 8 00:02:33 2013

@author: akels
"""
from __future__ import division, print_function
from os import sys
sys.path.append('cpresources')
from gaussxw import gaussxw
from pylab import *

q = 1
epsilon = 1e-7
h = 0.01
fi = empty((100,100))
for i,x in enumerate(arange(-0.5,0.5,h)):
        for j,y in enumerate( arange(-0.5,0.5,h)):
                r1 =  sqrt((x-0.05)**2 + y**2)
                r2 = sqrt((x+0.05)**2 + y**2)
                fi[i,j] = q/4/pi/epsilon*(1/r1 - 1/r2)

style_charges = {
        'potential':{'vmax':1e7,'vmin':-1e7},
        'magnitude':{'vmax':1e8}
        #'direction':{}
}

def results(fi,style=style_charges):
```

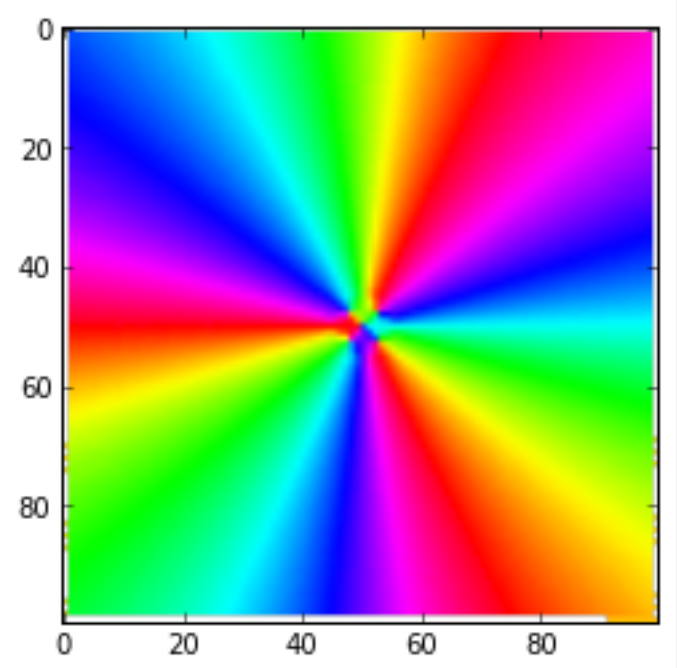
```
        imshow(fi,origin='lower',**style['potential'])
        show()

        # Partial derivatives
        h = 0.01

        fi_x = empty((100,100))
        fi_y = empty((100,100))
        #magnitude = empty((100,100))
        for i in range(1,100-1):
                for j in range(1,100-1):
                        fi_x[i,j] = (fi[i+1,j] - fi[i-1,j])/h/2
                        fi_y[i,j] = (fi[i,j+1] - fi[i,j-1])/h/2

        magnitude = sqrt(fi_x**2 + fi_y**2)
        direction = angle(fi_x + 1j*fi_y)

        imshow(magnitude,**style['magnitude'])
        show()

        hsv()
        imshow(direction)
        show()

#results(fi)

# Part C

q0 = 10
L = 0.1
ro = lambda x,y: q0*sin(2*pi*x/L)* sin(2*pi*y/L)

# x_ and y_ specifies place where the potential is calculated

#f = lambda x,y,x_,y_: ro(x,y)/sqrt((x_-x)**2 + (y_ - y)**2)

N = 10
x,w = gaussxw(N)
a = -L/2
b = L/2
xp = 0.5*(b-a)*x + 0.5*(b+a)
yp = xp
wp = 0.5*(b-a)*w

def fi_f(x_,y_):
        s=0

# Should try to use cpython to improve speed
        for i,xi in enumerate(xp):
                for j,yj in enumerate(yp):
```

```
                        r = sqrt((x_-xi)**2 + (y_ - yj)**2)

                        if r>1e-6: #Othervise potential will blow up
                                f = ro(xi,yj)/r
                        else:
                                f = 0

                        s+=wp[i]*wp[j]*f

        return s/4/pi/epsilon

fi_continius = empty((100,100))

for i,xi in enumerate(arange(-0.5,0.5,h)):
        for j,yi in enumerate( arange(-0.5,0.5,h)):
                fi_continius[i,j] = fi_f(xi,yi)

results(fi_continius)
```




```
-c:53: RuntimeWarning: overflow encountered in square
```



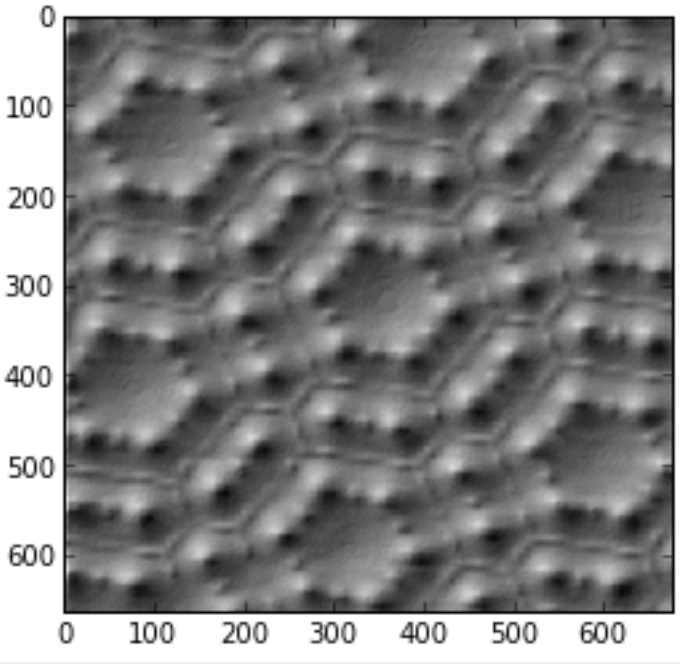


1.9 Image processing and the STM (5.5)

```
# -*- coding: utf-8 -*-
"""
Created on Thu Aug 8 12:34:59 2013
```

```
@author: akels
"""
from __future__ import division, print_function
from os import sys
sys.path.append('cpresources')
from pylab import *

from numpy import loadtxt

#data = loadtxt('cpresources/altitude.txt')
#h = 30000

data = loadtxt('cpresources/stm.txt')
h = 2.5

Nx,Ny = data.shape
f_x = empty((Nx,Ny))
f_y = empty((Nx,Ny))
I = empty((Nx,Ny))

Intensity = lambda w_x,w_y,fi=pi/4: (cos(fi)*w_x + sin(fi)*w_y)/ \
              sqrt(w_x**2 + w_y**2 +1)


for i in range(1,Nx-1):
       for j in range(1,Ny-1):
              f_x[i,j] = (data[i+1,j] - data[i-1,j])/2/h
              f_y[i,j] = (data[i,j+1] - data[i,j-1])/2/h

              I[i,j] = Intensity(f_x[i,j],f_y[i,j],fi=pi/4)

gray()
imshow(I)
```

```
<matplotlib.image.AxesImage at 0xb0f1c1cc>
```



1.10 Using Kichhoff's in the complex form (6.4)

```
# -*- coding: utf-8 -*-
"""
Created on Sun Aug 11 19:18:05 2013

@author: akels
"""
from __future__ import division, print_function
from os import sys
sys.path.append('cpresources')
from pylab import *

from numpy.linalg import solve

A = array([[1/1e3 + 1/2e3 + 1e3j*1e-6,  -1e3j*1e-6, 0],
              [-1000j*1e-6, 1/2e3 + 1/1e3 + 1000j*1e-6*1.5, -1000j*0.5e-6],
              [0, -1000j*0.5e-6, 1/1e3 + 2/1e3 + 1000j*0.5e-6]]
,dtype=complex)

v = [3/1e3,3/2e3,3/1e3]

x = solve(A,v)

from cmath import polar

for i,xi in enumerate(x):
       r,theta = polar(xi)
       print("""{}: V={}  phase={}""".format(i,r,theta*180/pi))
```

```
0:        V=1.7381099137654352                phase=-8.851555013072987
1:        V=1.3254956068840054                phase=6.777800780195954
2:        V=0.984744868402597                phase=3.23140390784674
```

1.11 The Lagrange point (6.12)

```
# -*- coding: utf-8 -*-
"""
Created on Tue Aug 13 09:05:53 2013

@author: akels
"""
from __future__ import division, print_function
from os import sys
sys.path.append('cpresources')
from pylab import *

accuracy = 1e-12

G = 6.674e-11
M = 5.974e24
m = 7.348e22
R = 3.844e8
omega = 2.662e-6

f = lambda r: G*M/r**2 - G*m/(R-r)**2 - omega**2*r

r1 = 1
r2 = 2
delta = 1.0
while abs(delta)>accuracy:
       r3 = r2 - f(r2)*(r2-r1)/(f(r2) - f(r1))
       r1,r2 = r2,r3

       delta = r2 - r1

print(r2/R)
```

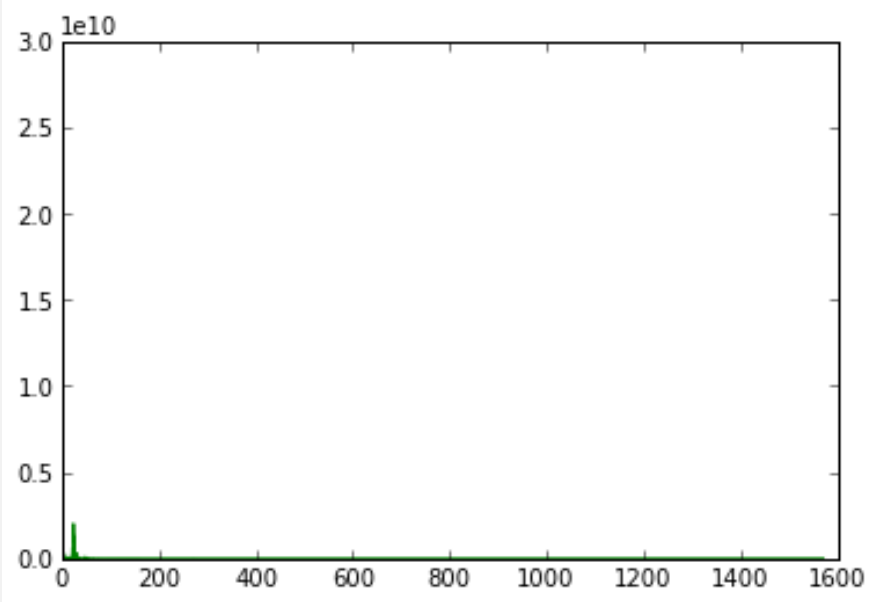
```
0.8481921739473347
```

1.12 Detecting periodicity of sunspots (7.2)

```
# -*- coding: utf-8 -*-
"""
Created on Tue Aug 13 18:42:50 2013

@author: akels
"""
```

```
from __future__ import division, print_function
from os import sys
sys.path.append('cpresources')
from pylab import *

data = loadtxt('cpresources/sunspots.txt',float)
y = data[:,1]
plot(y[500:1000])

def dft(y):
    N = len(y)
    c = zeros(N//2+1,complex)
    for k in range(N//2+1):
        for n in range(N):
            c[k] += y[n]*exp(-2j*pi*k*n/N)
    return c

c = dft(y)
power = real(c)**2 + imag(c)**2
plot(power)
```

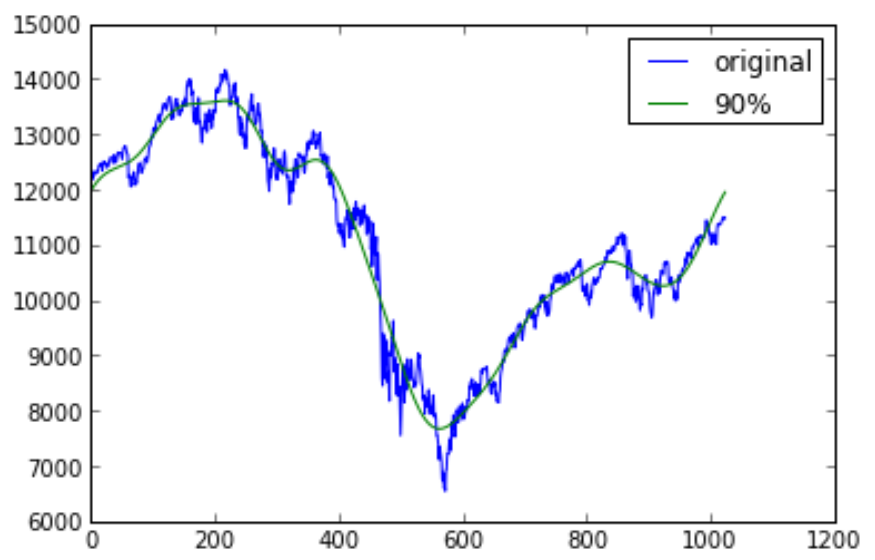
```
[<matplotlib.lines.Line2D at 0xb0f633ac>]
```



1.13 Fourier filtering and smoothing (7.3)

```
# -*- coding: utf-8 -*-
"""
Created on Wed Aug 14 15:17:14 2013

@author: akels
"""
from __future__ import division, print_function
```

```
#from os import sys
#sys.path.append('cpresources')
from pylab import *

from numpy import loadtxt
from numpy.fft import rfft, irfft

data = loadtxt('cpresources/dow.txt')
N = data.shape[0]

plot(data,label='original')

c = rfft(data)
c[N//100:]=0
y=irfft(c)
plot(y,label='90%')
legend()
show();
```



1.14 Really basic implementation of fast fourier transform

```
# -*- coding: utf-8 -*-
"""
Created on Wed Aug 14 18:24:01 2013

@author: akels
"""
from __future__ import division, print_function
from os import sys
sys.path.append('cpresources')
from pylab import *
from numpy import zeros
```

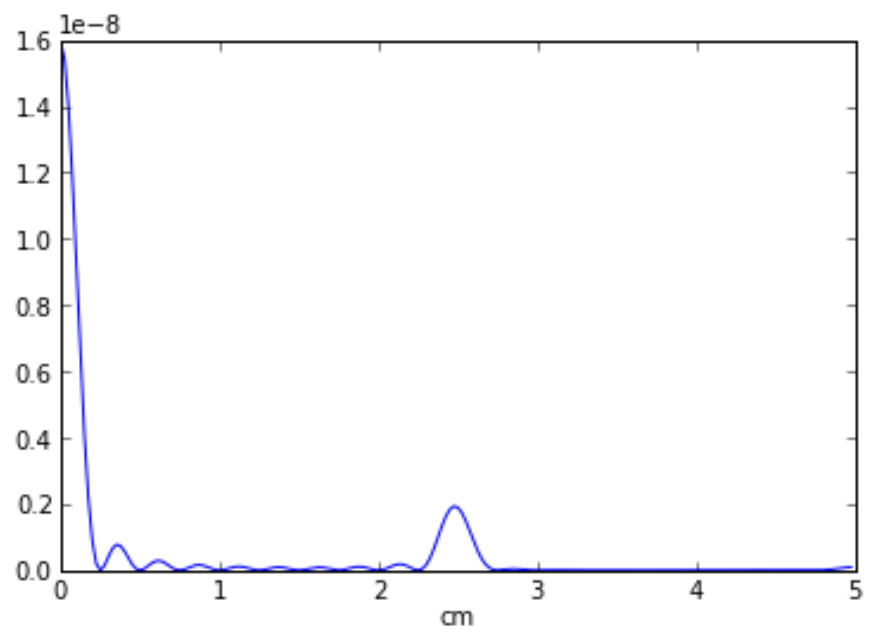
```
from cmath import exp

#E = empty((N,4),complex)
#E[:,3]=y

class E_wrap:
	def __init__(self,N):
		self.array = zeros((1 + log2(N),N),complex)
		self.N = N

	def __getitem__(self,(m,j,k)):

		k %= self.N/2**m # By using 7.45
		return self.array[m,j+2**m*k]

	def __setitem__(self,(m,j,k),value):

		k %= self.N/2**m # By using 7.45
		self.array[m,j+2**m*k]=value

def fft(y):
	N = len(y)
	E = E_wrap(N)
	E.array[-1,:]=y

	for m in range(int(log2(N))-1,-1,-1):
		for j in range(2**m):
			for k in range(N//2**m):
				E[m,j,k]=E[m+1,j,k]+exp(-2j*pi*k/(N/2**m))*E[m+1,j+2**m,k]
	return E.array[0,:]

y = loadtxt('cpresources/pitch.txt')
c = fft(y)
plot(abs(c))
```

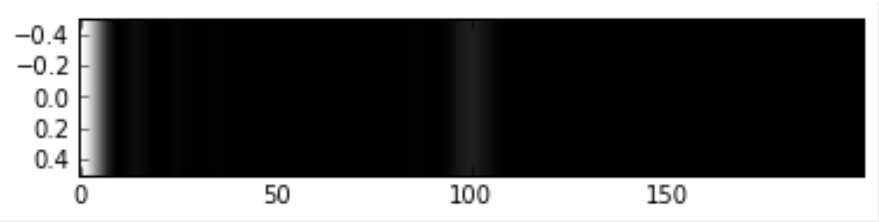
```
    File "<ipython-input-19-bdff49a26df0>", line 22
      def __getitem__(self,(m,j,k)):
                           ^
  SyntaxError: invalid syntax
```

1.15 Difraction gratings with fft (7.7)

```
# -*- coding: utf-8 -*-
"""
Created on Wed Aug 14 21:45:46 2013

@author: akels
"""
```

```
from __future__ import division, print_function
#from os import sys
#sys.path.append('cpresources')
from pylab import *

from numpy.fft import fft

d = 20e-6
alpha = pi/d
w = 200e-6
W = 10*w
wavelength = 500e-9
f = 1
q = lambda u: sin(alpha*u)**2

kmax = int(W*5e-2/wavelength)
x = wavelength*f/W*arange(200)

N = 1000
n = arange(N)
u = n*W/N - W/2
y = sqrt(q(u))
y[abs(u)>w/2]=0
c = fft(y)
I = (real(c)**2 + imag(c)**2)*W**2/N**2

plot(x*100,I[0:200])
xlabel('cm')
show()
imshow((I[:200],),aspect=40);
```

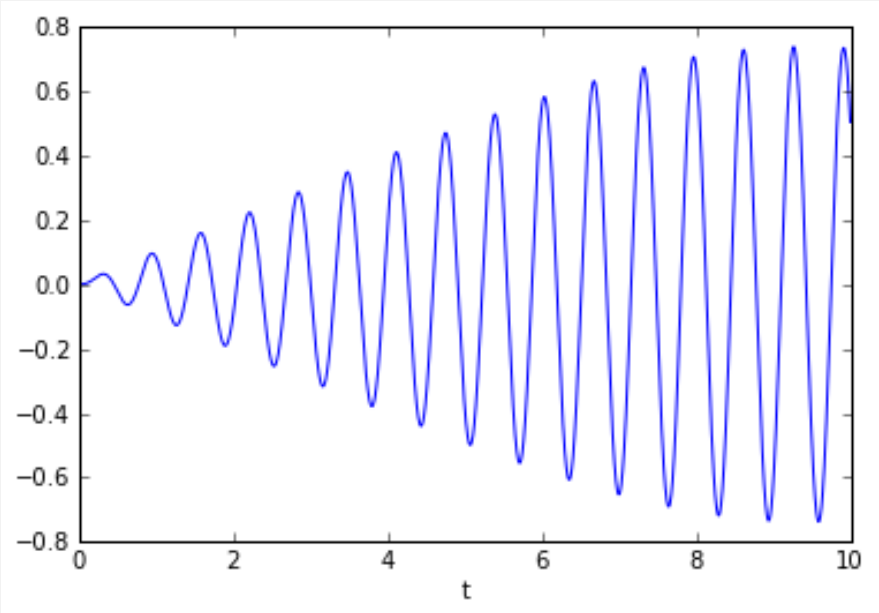




1.16 The driven pendulum (8.5)

```
# -*- coding: utf-8 -*-
"""
Created on Thu Aug 15 12:04:20 2013

@author: akels
"""
from __future__ import division, print_function
from math import sin,cos,pi
from numpy import array,arange
from pylab import plot,xlabel,show

g = 9.81
l =0.1
C = 2
Omega = 5*2

def f(r,t):
    theta = r[0]
    omega = r[1]
    ftheta = omega
    fomega = -g/l*sin(theta) + C*cos(theta)*sin(Omega*t)
    return array([ftheta,fomega],float)

a = 0.0
b = 10.0
N = 3000
h = (b-a)/N

tpoints = arange(a,b,h)
theta = []

r = array([0,0],float)
for t in tpoints:
    theta.append(r[0])
    k1 = h*f(r,t)
    k2 = h*f(r+0.5*k1,t+0.5*h)
    k3 = h*f(r+0.5*k2,t+0.5*h)
    k4 = h*f(r+k3,t+h)
    r += (k1+2*k2+2*k3+k4)/6
plot(tpoints,theta)
#plot(tpoints,ypoints)
```

```
xlabel("t")
show()
```



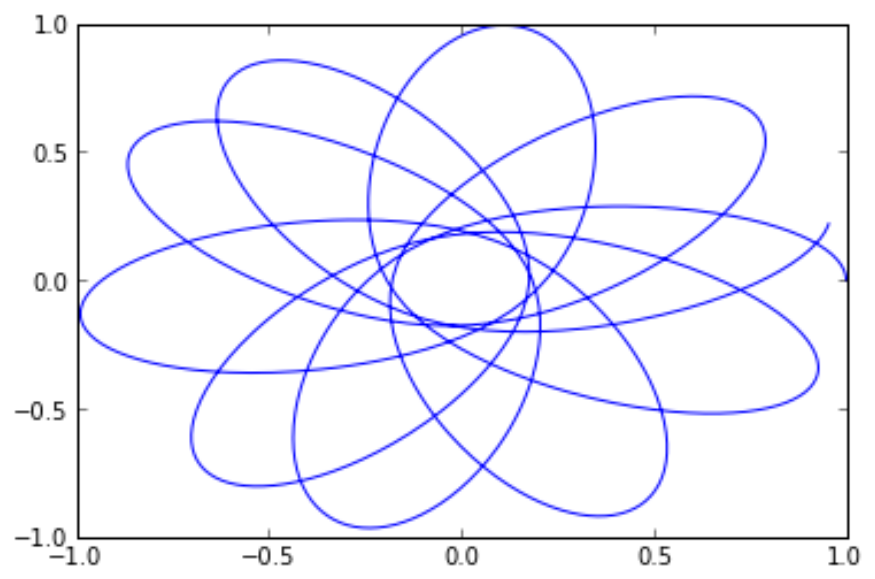
1.17 Space garbage (8.8)

```
# -*- coding: utf-8 -*-
"""
Created on Fri Aug 16 22:08:48 2013

@author: akels
"""
from __future__ import division, print_function
from numpy import arange,empty, array
from pylab import *
from cmath import exp

class rksolve:

        def __init__(self,f):
                self.f = f
                self.initial_conditions = None
                self.solution = None

        def iterate(self,a,b,N=1000):

                f = self.f
                r0 = array(self.initial_conditions,float)

                h = (b-a)/N

                tpoints = arange(a,b,h)
```

```
                solution = empty(tpoints.shape + r0.shape,float)

                #r_points[0] = r0
                r = r0
                for i,t in enumerate(tpoints):
                    solution[i]=r
                    k1 = h*f(r,t)
                    k2 = h*f(r+0.5*k1,t+0.5*h)
                    k3 = h*f(r+0.5*k2,t+0.5*h)
                    k4 = h*f(r+k3,t+h)
                    r += (k1+2*k2+2*k3+k4)/6

                self.h = h
                self.solution = solution
                self.t = tpoints

def array_decorator(f,*args,**kwargs):
        print('function decorated to return array')
        g = lambda *args,**kwargs: array(f(*args,**kwargs),float)
        return g


G = 1
M = 10
L = 2

@array_decorator
def f(r,t):

        x,y,vx,vy = r

        Dx = vx
        Dy = vy

        R = sqrt(x**2+y**2)
        a = - G*M/R/sqrt(R**2 + L**2/4)

        Dvx = a * x/R
        Dvy = a * y/R

        return [Dx,Dy,Dvx,Dvy]

#============================================================================
#
# @array_decorator
# def f(r,t):
#
#  r,theta,r_d,theta_d = r
#
#  Dr = r_d
#  Dtheta = theta_d
```

```
#
#  Dr_d = r*theta_d**2 - G*M/r/sqrt(r**2+L**2/4)
#  Dtheta_d = - r_d/r*(1+theta_d)
#
#  return Dr,Dtheta,Dr_d,Dtheta_d
#============================================================================

prob = rksolve(f)
prob.initial_conditions = [1,0,0,1]
prob.iterate(0,10)

x = prob.solution[:,0]
y = prob.solution[:,1]

plot(x,y)
show()
#polar(theta,r)
```

```
function decorated to return array
```



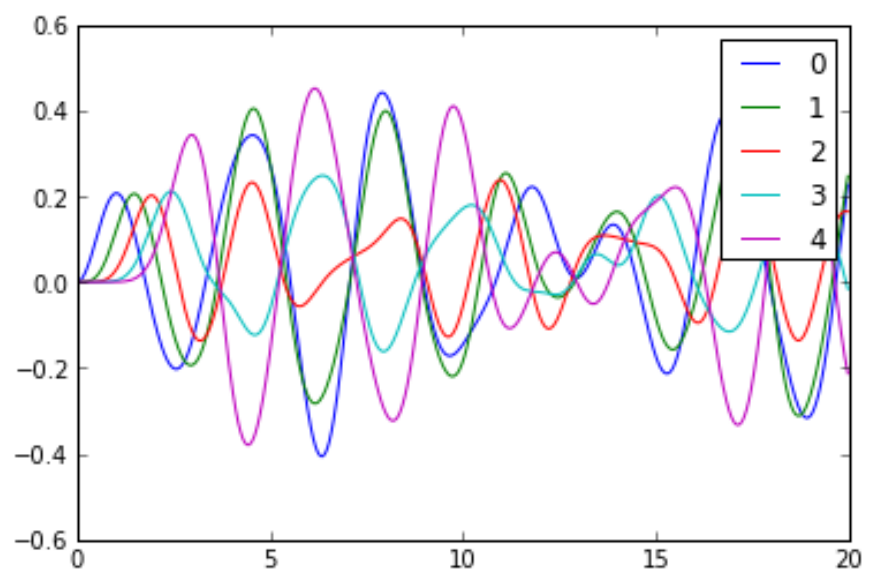
1.18 Vibration in one dimensional system (8.9)

```
# -*- coding: utf-8 -*-
"""
Created on Fri Aug 16 22:42:39 2013

@author: akels
"""
from __future__ import division, print_function
#from numpy import arange,empty, array,zeros
from pylab import *
```

```
class rksolve:

	def __init__(self,f):
		self.f = f
		self.initial_conditions = None
		self.solution = None

	def iterate(self,a,b,N=1000):

		f = self.f
		r0 = array(self.initial_conditions,float)

		h = (b-a)/N

		tpoints = arange(a,b,h)
		solution = empty(tpoints.shape + r0.shape,float)

		#r_points[0] = r0
		r = r0
		for i,t in enumerate(tpoints):
		    solution[i]=r
		    k1 = h*f(r,t)
		    k2 = h*f(r+0.5*k1,t+0.5*h)
		    k3 = h*f(r+0.5*k2,t+0.5*h)
		    k4 = h*f(r+k3,t+h)
		    r += (k1+2*k2+2*k3+k4)/6

		self.h = h
		self.solution = solution
		self.t = tpoints

def array_decorator(f,*args,**kwargs):
	print('function decorated to return array')
	g = lambda *args,**kwargs: array(f(*args,**kwargs),float)
	return g


k = 6
m = 1
omega = 2

#@array_decorator
def f(r,t):

	N = len(r)//2
	ksi = r[:N]
	vel = r[N:]

	D = empty(r.shape,float)
	Dksi = D[:N]
```

```
        Dvel = D[N:]
        # Lets assign velocities as position derivatives

        Dksi[:] = vel

        Dvel[0] = cos(omega*t)/m + k/m*(ksi[1]-ksi[0])
        Dvel[-1] = k/m*(ksi[-2]-ksi[-1])

        for i in range(1,N - 1):
                Dvel[i] = k/m*( (ksi[i+1]-ksi[i]) + (ksi[i-1] - ksi[i]) )

        return D

prob = rksolve(f)
N = 5
r0 = zeros(N*2,float)
#r0[0] = 10
prob.initial_conditions = r0
prob.iterate(0,20,1000)


#i = 0
for i in range(N):
        ksi_i = prob.solution[:,i]
        plot(prob.t,ksi_i,label=i)

legend()
show()

# Making a visualisation

from visual import sphere, rate

class grid:

        def __init__(self,N):

                self.N = N

                self.s = empty(N,sphere)
                for i in range(N):
                        self.s[i]=sphere(radius=0.1)

                self.eq = linspace(-5,5,N)
                self.update()

        def update(self,ksi=0):

                new_pos = self.eq + ksi
                for i in range(self.N):
                        self.s[i].pos = (new_pos[i],0,0)
```

```
                    #print(new_pos[i])


system = grid(N)
for ksi_i in prob.solution[::10,:N]:
        rate(30)
        system.update(ksi_i)
```




```
---------------------------------------------------------------------------
ImportError                               Traceback (most recent call last)

<ipython-input-29-a486f1c6a82b> in <module>()
     90 # Making a visualisation
     91
---> 92 from visual import sphere, rate
     93
     94 class grid:


ImportError: No module named 'visual'
```

1.19 Quantum oscillators (8.11)

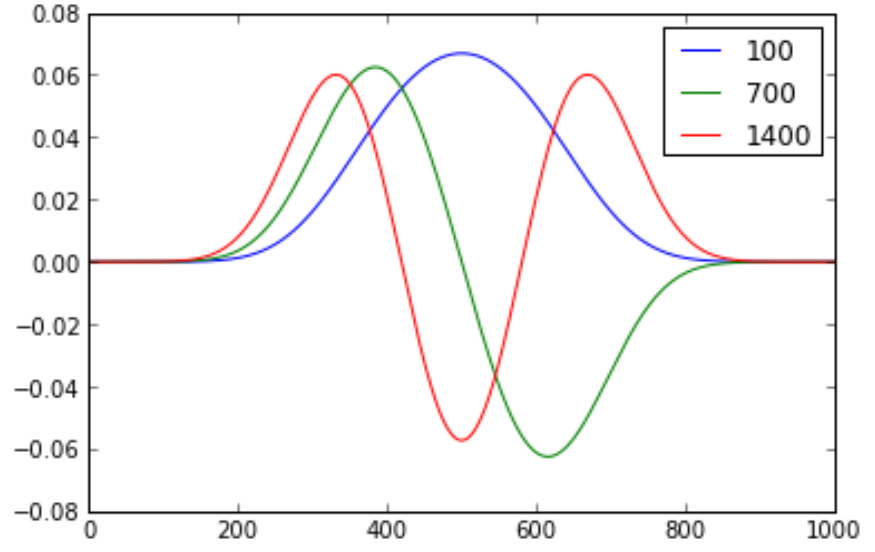
```
# -*- coding: utf-8 -*-
"""
Created on Sun Aug 18 18:14:55 2013

@author: akels
"""
from __future__ import division, print_function
```

```
from numpy import array,arange,copy
from pylab import *

# Didn't work?

def f_(r,u,E):
       r"""
       The function is

       .. math::
              \frac{d^2 \psi}{du^2} = \frac{2m}{h^2(1-u)^4}[E - V(\frac{u}{1-u})] + \
                  frac{2}{1-u}\frac{d\psi}{du}

       """
       ksi, teta = r
       Dksi = teta
       Dteta = 2*m/hbar**2/(1-u)**4*(E - V(u/(1-u))*ksi + 2/(1-u)*teta )
       return array([Dksi,Dteta],float)

# Constants
m = 9.1094e-31 # Mass of electron
hbar = 1.0546e-34 # Planck's constant over 2*pi
e = 1.6022e-19 # Electron charge
a = 1e-11
V0 = 50*e
L = 5*a # Bohr radius
N = 1000
h = 2*L/N

# Potential function
def V(x):
    return V0*x**4/a**4

def f(r,x,E):
    psi = r[0]
    phi = r[1]
    fpsi = phi
    fphi = (2*m/hbar**2)*(V(x)-E)*psi
    return array([fpsi,fphi],float)

# Calculate the wavefunction for a particular energy
#solution = []
def solve(E):
    psi = 0.0
    phi = 1.0
    r = array([psi,phi],float)
# solution = []
    for x in arange(-L,L,h):
        #solution.append(r)
        k1 = h*f(r,x,E)
        k2 = h*f(r+0.5*k1,x+0.5*h,E)
```

```
        k3 = h*f(r+0.5*k2,x+0.5*h,E)
        k4 = h*f(r+k3,x+h,E)
        r += (k1+2*k2+2*k3+k4)/6

    return r[0] #array(solution) #r[0]

# Main program to find the energy using the secant method
def energy(E=0):

        E*=e
        E1 = E#0.0
        E2 = E + e
        psi2 = solve(E1)

        target = e/1000
        while abs(E1-E2)>target:
            psi1,psi2 = psi2,solve(E2)
            E1,E2 = E2,E2-psi2*(E2-E1)/(psi2-psi1)

        print("E =",E2/e,"eV")
        return E2/e

Energies = [205.3,735.7,1443.6]


def solution(E):
    E *=e
    psi = 0.0
    phi = 1.0
    r = array([psi,phi],float)
    result = []
    for x in arange(-L,L,h):
        result.append(copy(r))
        k1 = h*f(r,x,E)
        k2 = h*f(r+0.5*k1,x+0.5*h,E)
        k3 = h*f(r+0.5*k2,x+0.5*h,E)
        k4 = h*f(r+k3,x+h,E)
        r += (k1+2*k2+2*k3+k4)/6

    return array(result,float)[:,0] #array(solution) #r[0]

precise = lambda E: solution(energy(E))

from scipy.integrate import simps

for Ei in [100,700,1400]:
        y = precise(Ei)
        Iy = simps(y**2)
        plot(y/sqrt(Iy),label=Ei)

legend()
```

```
show();
```

```
E = 205.306903529 eV
E = 735.691243084 eV
E =
```



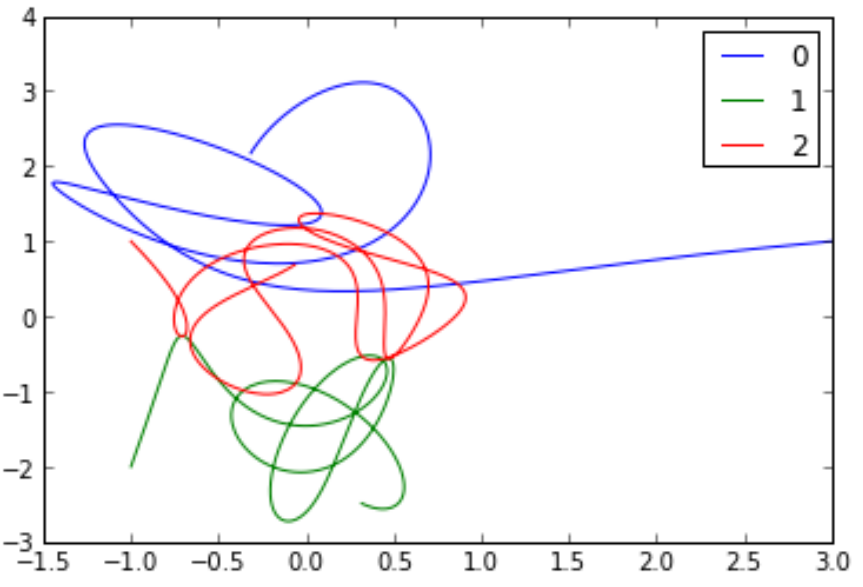
1.20 Three body problem (8.2)

```
# -*- coding: utf-8 -*-
"""
Created on Mon Aug 19 00:02:39 2013

@author: akels
"""
from __future__ import division, print_function
from os import sys
sys.path.append('cpresources')
from pylab import *


def r(r1,r2):

        delta = r1 - r2
        length = sqrt(delta[0]**2 + delta[1]**2)

        return delta/length**3

G = 1
m1 = 150
m2 = 200
m3 = 250
```

```
def f(r_,t):
        r1,r2,r3,v1,v2,v3 = r_

        Dr1 = v1
        Dr2 = v2
        Dr3 = v3

        Dv1 = m2*r(r2,r1) + m3*r(r3,r1)
        Dv2 = m1*r(r1,r2) + m3*r(r3,r2)
        Dv3 = m1*r(r1,r3) + m2*r(r2,r3)

        return array([Dr1,Dr2,Dr3,Dv1,Dv2,Dv3],float)



class rksolve:

        def __init__(self,f):

                self.f = f #self.array_decorator(f)

                self.initial_conditions = None
                self.solution = None

        def iterate(self,a,b,N=1000):

                #f = self.f
                r0 = array(self.initial_conditions,float96)

                h = (b-a)/N

                tpoints = arange(a,b,h)
                solution = empty(tpoints.shape + r0.shape,float)

                #r_points[0] = r0
                r = r0
                for i,t in enumerate(tpoints):
                    solution[i]=r
                    r += self.estimate_delta(r,t,h)

                self.h = h
                self.solution = solution
                self.t = tpoints

        def estimate_delta(self,r,t,h):

                f = self.f
                k1 = h*f(r,t)
                k2 = h*f(r+0.5*k1,t+0.5*h)
                k3 = h*f(r+0.5*k2,t+0.5*h)
                k4 = h*f(r+k3,t+h)
```

```
                return (k1+2*k2+2*k3+k4)/6

class rksolve_adaptive(rksolve):

        def iterate(self,a,b,delta=1):


                r0 = array(self.initial_conditions,float96)

                h = (b-a)/10000
                solution = []
                time = []
                h_list = []
                r = r0
                t = a

                solution.append(copy(r))
                time.append(t)
                h_list.append(h)

                def distance(r1,r2):
                        r1 = r1[:3]
                        r2 = r2[:3]

                        return sqrt(sum((r1-r2)**2))

                ro = 1
                while t<b:
                        if ro<2:
                                h = h*ro**(1/4)
                        else:
                                h*=2

                        if h>1e-3: h = 1e-3
                        # estimating ro
                        r1 = r + self.estimate_delta(r,t,h)
                        r1 += self.estimate_delta(r1,t+h,h)
                        r2 = r + self.estimate_delta(r,t,2*h)

                        #difference = r1 - r2
                        ro = 30*h*delta/distance(r1,r2) #sqrt(difference[0]**2 + difference
                            [1]**2)

                        if ro>1:
                                t +=2*h
                                r = r1
                                solution.append(copy(r))
                                time.append(t)
                                h_list.append(h)

                self.h = h_list
```

```
                self.solution = array(solution)
                self.t = time

prob = rksolve_adaptive(f)

r1 = [3,1]
r2 = [-1,-2]
r3 = [-1,1]
prob.initial_conditions = [r1,r2,r3,[0,0],[0,0],[0,0]]

prob.iterate(0,2,delta=1e-3)

for i in range(3):
        x = prob.solution[:,i,0]
        y = prob.solution[:,i,1]
        plot(x,y,label=i)
legend()
show()

from visual import sphere, rate

def radius(m):

        return m**(1/3)/100

s1 = sphere(radius=radius(150))
s2 = sphere(radius=radius(200))
s3 = sphere(radius=radius(250))

s = [s1,s2,s3]

C = 0.1
for h,pos in zip(prob.h,prob.solution[:,:3]):
        rate(int(C/h))

        for si,posi in zip(s,pos):
                rx,ry = posi
                si.pos = rx,ry,0
```

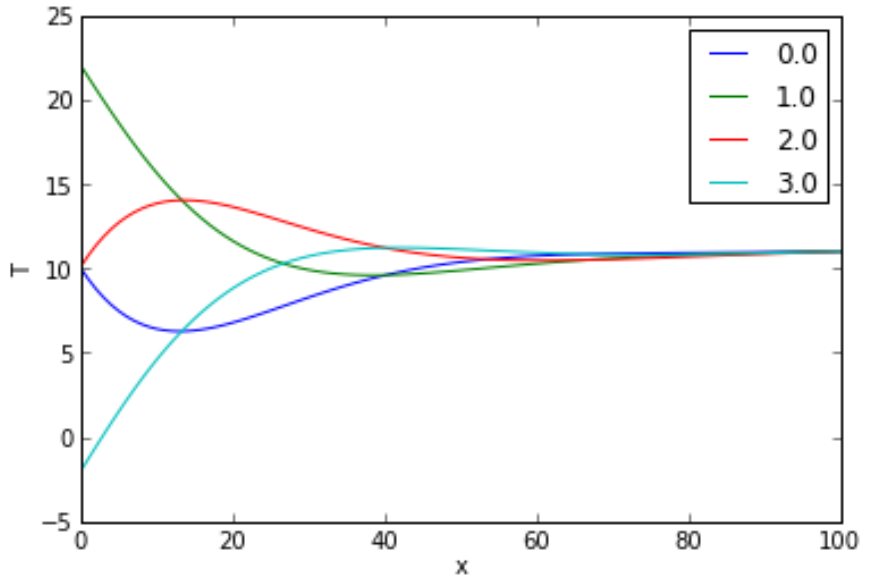



```
---------------------------------------------------------------------------
ImportError                               Traceback (most recent call last)

<ipython-input-31-b98cb19a78f8> in <module>()
    143 show()
    144
--> 145 from visual import sphere, rate
    146
    147 def radius(m):


ImportError: No module named 'visual'
```

1.21 Thermal diffusion in Earth's crust (9.4)

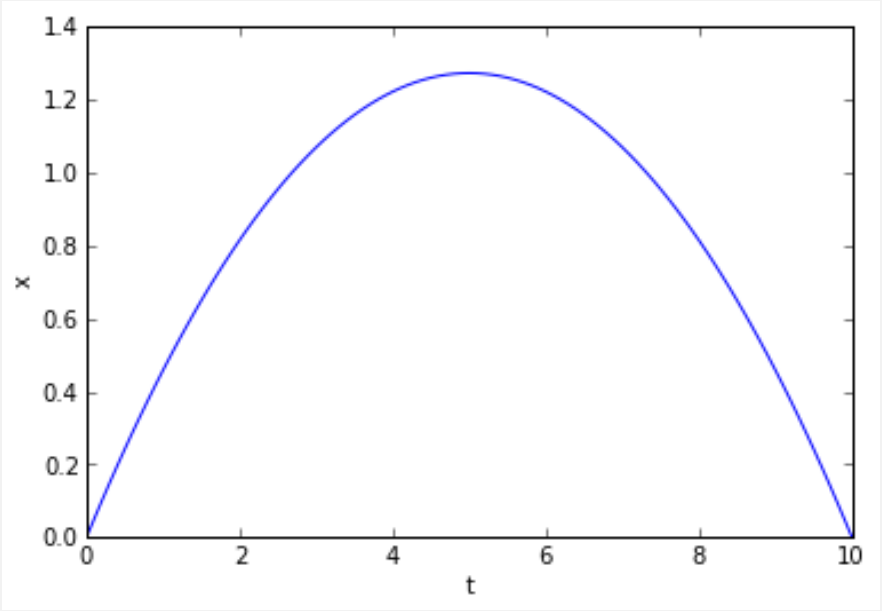
```
# -*- coding: utf-8 -*-
"""
Created on Mon Aug 19 13:35:52 2013

@author: akels
"""
from __future__ import division, print_function
from pylab import *

A = 10
B = 12
tau = 365
D = 0.1
def T0(t):
```

```
        return A + B*sin(2*pi*t/tau)

L = 20 # Thickness of steel in meters
D = 0.1 # Thermal diffusivity
N = 100 # Number of divisions in grid
a = L/N # Grid spacing
h = 0.01 # Time-step
#epsilon = h/1000

T = zeros(N+1,float)
T[1:N]=10


def iterate(T,t_min,t_max):
        # Main loop
        t = t_min
        c = h*D/a**2

        while t<t_max:

            # Calculate the new values of T
                T[0] = T0(t)
                T[N] = 11
                T[1:N] = T[1:N] + c*(T[2:N+1]+T[0:N-1]-2*T[1:N])

                #T,Tp = Tp,T
                t += h
        return T


T9 = iterate(T,0,365*9)

T9_i = T9
t_min = 365*9
for t_max in [365*9 + i*(365//4) for i in range(4)]:
        #t_max = t_min + 365//4
        T9_i = iterate(T9_i,t_min,t_max)
        plot(T9_i,label=t_max%365/(365//4))
        t_min = t_max

legend()
xlabel("x")
ylabel("T")
show();
```



1.22 The relaxation method for ordinary differential equations (9.7)

```
# -*- coding: utf-8 -*-
"""
Created on Tue Aug 20 09:11:24 2013

@author: akels
"""
from __future__ import division, print_function
from pylab import *

N = 100
L_t = 10
h = L_t/N
g = 9.81

x = zeros(N+1,float)

x[0]=0
x[N]=0

delta = 1
while delta>1e-6:
        xp = h**2/g/2 + 1/2*(x[2:N+1] + x[0:N-1])
        delta = max(abs(xp-x[1:N]))
        x[1:N] = xp

plot(linspace(0,10,N+1),x)
xlabel('t')
ylabel('x')
show()
```



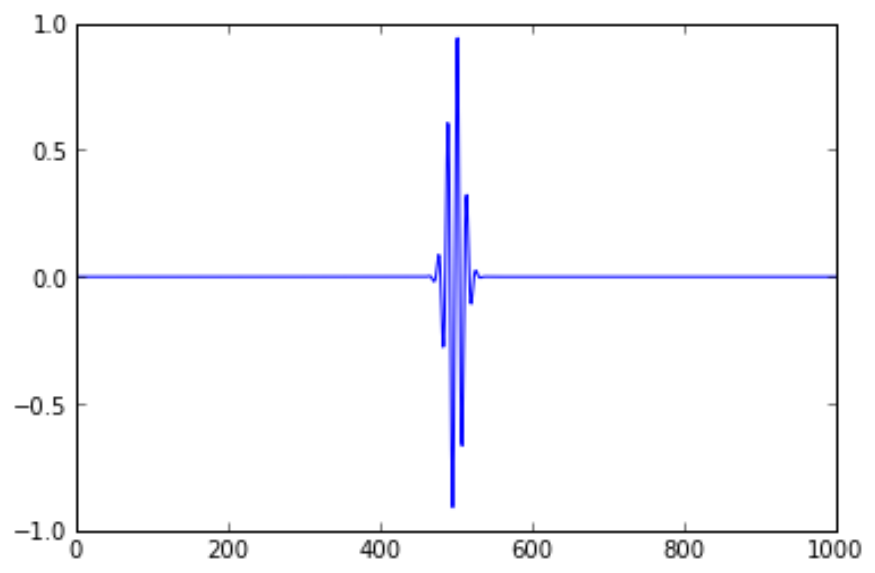
1.23 The Schrodinger equation and the spectral method (9.9)

```
# -*- coding: utf-8 -*-
"""
Created on Tue Aug 20 15:28:20 2013

@author: akels
"""
from __future__ import division, print_function
from os import sys
sys.path.append('cpresources')
from pylab import *


h = 2e-18*10
hbar = 1.0546e-36
L = 1e-8
M = 9.109e-31
N = 1000 # Grid slices

a = L/N

def complex_arg(trans):
       def f(y):
              return trans(real(y)) + 1j*trans(imag(y))

       return f

@complex_arg
def dst(y):
```

```
    """
        Perform dst transform for real argument
    """
    N = len(y)
    y2 = empty(2*N,float)
    y2[0] = y2[N] = 0.0
    y2[1:N] = y[1:]
    y2[:N:-1] = -y[1:]
    a = -imag(rfft(y2))[:N]
    a[0] = 0.0

    return a


##############################################################################
# 1D inverse DST Type-I

@complex_arg
def idst(a):
    N = len(a)
    c = empty(N+1,complex)
    c[0] = c[N] = 0.0
    c[1:N] = -1j*a[1:]
    y = irfft(c)[:N]
    y[0] = 0.0

    return y



ksi = zeros(N+1,complex)

def ksi0(x):
        x0 = L/2
        sigma = 1e-10
        k = 5e10
        return exp(-(x-x0)**2/2/sigma**2)*exp(1j*k*x)

x = linspace(0,L,N+1)
ksi[:] = ksi0(x)
ksi[[0,N]]=0

b0 = dst(ksi)

t = 1e-18
b_ = b0*exp(1j*pi**2*hbar*arange(1,N+2)**2/2/M/L**2*t)

ksi_ = idst(b_)
plot(ksi_)
show()
```

```
from visual import curve, rate

ksi_c = curve()
ksi_c.set_x(x-L/2)

#ksi = banded(A,v,1,1)
t = 0
while True:
        rate(30)
        b_ = b0*exp(1j*pi**2*hbar*arange(1,N+2)**2/2/M/L**2*t)
        ksi_ = idst(b_)

        ksi_c.set_y(real(ksi_)*1e-9)
        ksi_c.set_z(imag(ksi_)*1e-9)

        t +=h*5
```

```
/usr/lib/python3/dist-packages/numpy/core/numeric.py:321: ComplexWarning: Casting complex value
  return array(a, dtype, copy=False, order=order)
```

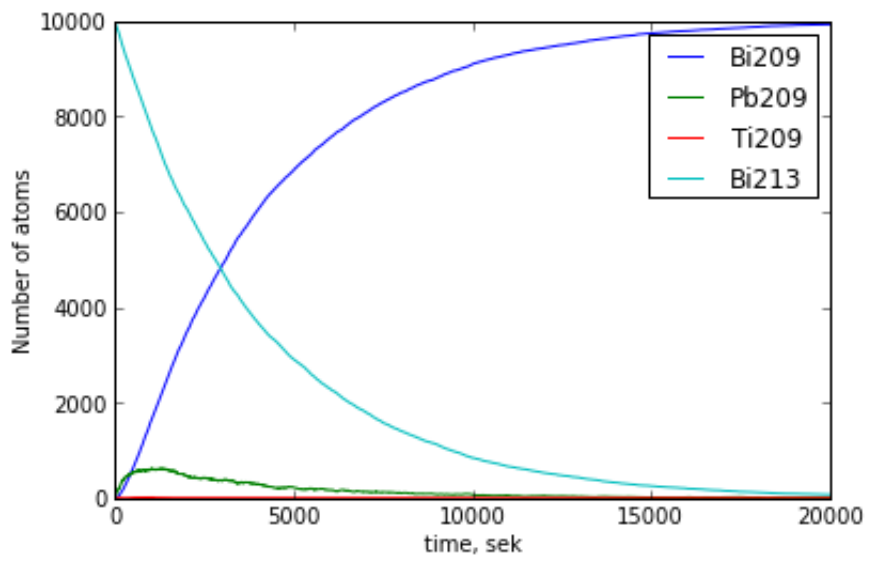


```
---------------------------------------------------------------------------
ImportError                               Traceback (most recent call last)

<ipython-input-36-7f7edb1b1b1c> in <module>()
     79
     80
---> 81 from visual import curve, rate
     82
     83 ksi_c = curve()


ImportError: No module named 'visual'
```

1.24 Radioactive decay chain (10.2)

```
# -*- coding: utf-8 -*-
"""
Created on Tue Aug 20 21:11:59 2013

@author: akels
"""
from __future__ import division, print_function
#from os import sys
#sys.path.append('cpresources')
from pylab import *

from random import random
random()

h = 1

Bi209 = 0
Pb209 = 0
Ti209 = 0
Bi213 = 10000

pPb = 1 - 2**(-h/3.3/60)
pTi = 1 - 2**(-h/2.2/60)
pBi = 1 - 2**(-h/46/60)

Bi209_list = []
Pb209_list = []
Ti209_lsit = []
Bi213_list = []

t = arange(0,2e4,h)
for ti in t:
       Bi209_list.append(Bi209)
```

```
        Pb209_list.append(Pb209)
        Ti209_lsit.append(Ti209)
        Bi213_list.append(Bi213)

        for i in range(Pb209):
                if random()<pPb:
                        Pb209-=1
                        Bi209+=1

        for i in range(Ti209):
                if random()<pTi:
                        Ti209-=1
                        Pb209+=1

        for i in range(Bi213):
                if random()<pBi:
                        Bi213 -=1
                        if random()<0.9791:
                                Pb209+=1
                        else:
                                Ti209+=1

plot(t,Bi209_list,label='Bi209')
plot(t,Pb209_list,label='Pb209')
plot(t,Ti209_lsit,label='Ti209')
plot(t,Bi213_list,label='Bi213')
legend()
xlabel('time, sek')
ylabel('Number of atoms')
show();
```



1.25 Monte carlo integration (10.8)

```
# -*- coding: utf-8 -*-
"""
Created on Thu Aug 22 06:47:08 2013

@author: akels
"""
from __future__ import division, print_function
#from os import sys
#sys.path.append('cpresources')
from pylab import *

N = 10000000

z = random(N)
x = z**2

def g(x):

        return 1/(1+exp(x))

I = sum(g(x))/N*2


print('I = {}'.format(I))
```

```
I = 0.8388647822732387
```

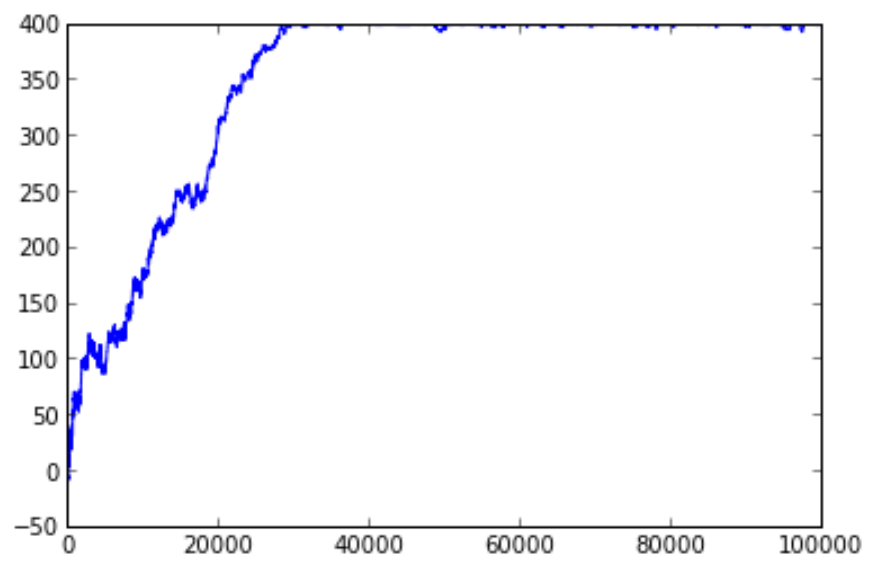
1.26 The Ising model (10.9)

```
# -*- coding: utf-8 -*-
"""
Created on Thu Aug 22 17:07:57 2013

@author: akels
"""
from __future__ import division, print_function
#from os import sys
#sys.path.append('cpresources')
from pylab import *

from numpy.random import *
#seed(1)
seed(5)

N = 20
J = 1
T = 1
kb = 1
```

```
beta = 1
steps = 100000#0000

s = empty((N,N),int)

for i in range(N):
	for j in range(N):
		if random()<0.5:
			s[i,j]=1
		else:
			s[i,j]=-1

def energy(s):

	s1 = s[:-1,:]*s[1:,:]
	s2 = s[:,:-1]*s[:,1:]

	E = -J*(sum(s1) + sum(s2))

	return E

def energy_check(s):

	I = 0
	for i in range(N-1):
		for j in range(N):
			I+=s[i,j]*s[i+1,j]

	for i in range(N):
		for j in range(N-1):
			I+=s[i,j]*s[i,j+1]

	return -J*I

eplot = []
Mplot = []
E1 = energy(s)
M = sum(s)
for k in range(steps):
	i = randint(N)
	j = randint(N)

	s[i,j] *=-1

	E2 = energy(s)

	dE = E2 - E1
	#print(dE)

	if dE>0:
		if random()<exp(-beta*dE):
```

```
                        E1 = E2 #flip is acepted
                        M = sum(s)
                else:
                        s[i,j]*=-1
                        #E1 = E2
        else:
                E1 = E2 #flip is accepted because energy falls
                M = sum(s)

        #eplot.append(E1)
        Mplot.append(M)

plot(Mplot);
```

```
[<matplotlib.lines.Line2D at 0xb094e28c>]
```



1.27 The global minimum of a function (10.10)

```
# -*- coding: utf-8 -*-
"""
Created on Thu Aug 22 20:29:10 2013

@author: akels
"""
from __future__ import division, print_function
from pylab import *
from numpy.random import random, standard_normal

Tmax = 1
Tmin = 1e-3
tau = 1e4
x0 = 2
```

```
def g(x):
        if x>0 and x<100:
                return cos(x) + cos(sqrt(2)*x + cos(sqrt(3)*x))
        else:
                return 1e10

def swap_function(f):

        def h(x): return g(x)

        return h


@swap_function
def f(x):
        return x**2 - cos(4*pi*x)

fx = f(x0)
t = 0
T = Tmax
x = x0

while T>Tmin:

        t+=1
        T = Tmax*exp(-t/tau)

        oldx = x
        oldfx = fx
        r = standard_normal()
        x += r
        fx = f(x)

        delta_fx = fx - oldfx

        if random()>exp(-delta_fx/T):
                x = oldx
                fx = oldfx

print('x = {} with f(x) = {}'.format(x,fx))
```

```
x = 46.48428621618932 with f(x) = -1.7900313105761207
```

1.28 Difusion-limited aggregation (10.13)

```
# -*- coding: utf-8 -*-
"""
Created on Fri Aug 23 13:32:33 2013

@author: akels
```

```
"""
from __future__ import division, print_function
from pylab import *

from visual import sphere, display
from numpy.random import randint


L = 201
tau = L*L//6#1e4

#i = 50
#j = 50

def add(i,j,time=tau):
        grid[i,j]=True
        s = sphere(radius=0.5)
        s.pos = i,j #i-50,j-50

        color = 1 #time/tau#(1-1*exp(-t/tau))
        s.color = color,color,color
        #print(i,j)

def circle(r=0):
        """
        Generates i,j randomly on the circle with radius r+1
        """
        teta = 2*pi*random()
        x = (r+1)*cos(teta) + L//2
        y = (r+1)*sin(teta) + L//2

        i = int(x) + 1
        j = int(y) + 1
        print(r)
        return i,j

def radius(i,j):

        dx = i - L//2
        dy = j - L//2

        distance = sqrt(dx**2 + dy**2)
        #print(dx,dy)
        return distance


# Configuring view
d = display(center=(L//2,L//2))
s = sphere()
s.pos = L,L
d.autoscale = False
```

```
s.visible=False

grid = zeros((L,L),bool)
center = L//2,L//2 #50,50
add(*center)
t = 0
r = 1
while r*2<L//2:
	t+=1

	i,j = circle(r)
	ri = 0
	while ri<=2*r:
		a = randint(4)
		# If next position is False!=grid[i,j] then I should add sphere
		newi,newj = i,j
		if a==0: #move up
			newi+=1
		elif a==1:
			newi-=1
		elif a==2:
			newj+=1
		elif a==3:
			newj-=1

		if grid[newi,newj]==True:
			add(i,j,time=t)
			if r<ri: r = int(ri)
			break
		else:
			i,j = newi, newj
		ri = radius(i,j)

else:
	print('Sucesfully filled')
```

```
---------------------------------------------------------------------------
ImportError                               Traceback (most recent call last)

<ipython-input-46-70eb5f848617> in <module>()
      8 from pylab import *
      9
---> 10 from visual import sphere, display
     11 from numpy.random import randint
     12


ImportError: No module named 'visual'
```