University of Zurich - HS 2013

Computational Science I
Exercise notes: Vampire & Schroedinger equation

Tobias Grubenmann

December 1, 2013

Exercise 1

The following code solves the Vampire equation for the initial condition:

$$f(x) = \begin{cases} 0 & \text{if } x < 0.4 \\ \sin(x) & \text{if } 0.4 \leq x \leq 0.6 \\ 0 & \text{if } x > 0.6 \end{cases}$$

The timestep is 10^{-5}.

```
import matplotlib.pyplot as plt

def F(f, i, dx):
    return f[i] * (1 - f[i]) + (f[i+1]+f[i-1]-2*f[i])/(dx
       **2)

numberOfGridPoints = 101
dx = 1.0/(numberOfGridPoints-1)
f = []
dt = 0.00001
numberOfTimeSteps = 2000

fig = plt.figure()
ax1 = fig.add_subplot(111)

# initial conditions
```

```
for i in range(numberOfGridPoints):
    if i > 40 and i < 60:
        f += [sin((i-40)*dx*5*pi)]
    else:
        f += [0]

# set boundaries to 0
f[0] = 0
f[-1] = 0

for i in range(numberOfTimeSteps):

    # set boundaries to 0
    fHalf = [0]

    for j in range(numberOfGridPoints-2):

        fHalf += [f[j+1] + 0.5*dt*F(f, j+1, dx)]

    # set boundaries to 0
    fHalf += [0]

    for j in range(numberOfGridPoints-2):

        f[j+1] = f[j+1] + dt*F(fHalf, j+1, dx)

    if i%400 == 0:
        x = np.arange(0., 1. + dx, dx)
        ax1.plot(x, f, label='t = ' + str(i*dt))

leg = ax1.legend(loc=1)

plt.show()
```

The following plot shows the solution for the Vampire equation for specific time:

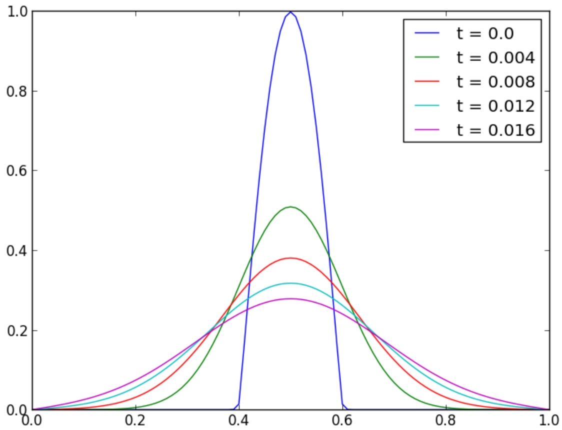

Figure 1: Solution of the Vampire equation after specific time steps

Exercise 2

This codes solves the Schroedinger equation for the potential $V(x) = \frac{1}{2}(x-1)^2$. The initial condition is the square root of a gaussian $N(-4,1)$ distribution.

```
import matplotlib.mlab as mlab
import numpy.fft as npfft
from Tkinter import *
import matplotlib.pyplot as plt

def V(x):
    return 0.5 * (x-1.0)**2

def frame(psi, numberOfGridPoints, L, dt, ax1, ax2):
    for i in range(numberOfGridPoints):

        x = i*dx - L/2.0

        psi[i] = exp(-1j * V(x)*dt/2)*psi[i]

    # FFT
    psi2 = npfft.ifftshift(npfft.fft(npfft.fftshift(psi)))

    for i in range(numberOfGridPoints):

        k = i*2*pi/L - 2 * pi * numberOfGridPoints / (2 * L
            )
```

```
        psi2[i] = exp(-1j * 0.5*(k**2)*dt)*psi2[i]

    # iFFt
    psi = npfft.ifftshift(npfft.ifft(npfft.fftshift(psi2)))

    for i in range(numberOfGridPoints):

        x = i*dx - L/2.0

        psi[i] = exp(-1j * V(x)*dt/2)*psi[i]

    ax1.cla()
    ax1.plot([z.real for z in psi])
    ax1.plot([z.imag for z in psi])
    # Potential
    ax1.plot([V(j*dx - L/2.0) for j in range(
       numberOfGridPoints)])
    ax2.cla()
    ax2.plot([z.real for z in psi2])
    ax2.plot([z.imag for z in psi2])
    plt.ylim([-1,1])
    plt.gcf().canvas.draw()

    screen.after(16, frame, psi, numberOfGridPoints, L, dt,
        ax1, ax2)

screen = Tk()

f, (ax1, ax2) = plt.subplots(2, sharex=True, sharey=True)

numberOfGridPoints = 2001
L = 30
psi = []
dx = L*1.0/(numberOfGridPoints-1)
dt = 0.01

# Gaussian distribution for initial condition
for i in range(numberOfGridPoints):
    psi += [sqrt(mlab.normpdf(i*dx - L/2.0, -4, 1))]

screen.after(1, frame, psi, numberOfGridPoints, L, dt, ax1,
    ax2)
```

```
#canv.update()

screen.mainloop()
```

The code generates an animated graph for the function ψ (top plot) as well as the Fourier Transform $\tilde{\psi}$ (bottom plot). For both, ψ and $\tilde{\psi}$, the real component (blue) and the imaginary component (green) are plotted. In the top plot the potential $V(x)$ is also plotted (red plot).

The following plot shows a screenshot of the animated plot:

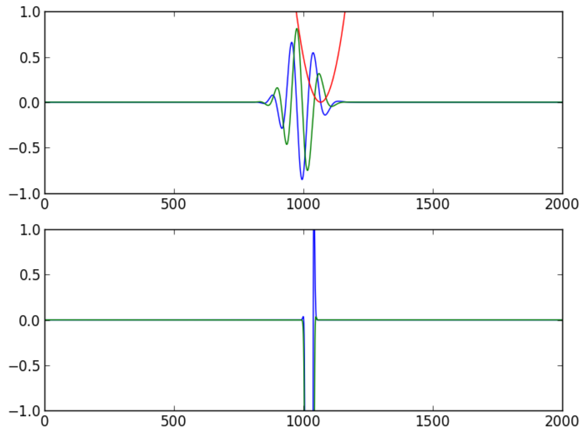

Figure 2: Solution of the Vampire equation after specific time steps