Solutions to Part B of Problem Sheet 4

Solution (4.4)

(a) Given complex numbers $z_1 = a + ib$ and $z_2 = c + id$, we can express the real and imaginary parts of the product $z_3 = z_1 z_2$ as

$$\begin{pmatrix} \mathrm{re}(z_3) \\ \mathrm{im}(z_3) \end{pmatrix} = \begin{pmatrix} a & -b \\ b & a \end{pmatrix} \begin{pmatrix} c \\ d \end{pmatrix}.$$

In the same fashion, a system of equations $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, with $\boldsymbol{A}$ and $\boldsymbol{x}$ complex, we can be written as

$$\begin{pmatrix} \mathrm{re}(\boldsymbol{b}) \\ \mathrm{im}(\boldsymbol{b}) \end{pmatrix} = \begin{pmatrix} \mathrm{re}(\boldsymbol{A}) & -\mathrm{im}(\boldsymbol{A}) \\ \mathrm{im}(\boldsymbol{A}) & \mathrm{re}(\boldsymbol{A}) \end{pmatrix} \begin{pmatrix} \mathrm{re}(\boldsymbol{c}) \\ \mathrm{im}(\boldsymbol{c}) \end{pmatrix}.$$

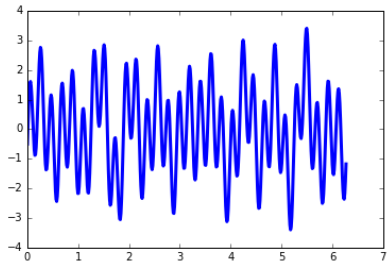
Since we know that the target vector $\boldsymbol{b}$ is real, we only need the upper half of this system. Once this is solved, we can assemble the complex $\boldsymbol{c}$ from it.

(b)+(c) The code could look something like this:

In [1]:
```
import numpy as np
import numpy.linalg as la
import numpy.random as rnd
import numpy.fft as fft
import matplotlib.pyplot as plt
import cvxpy as cvx
```

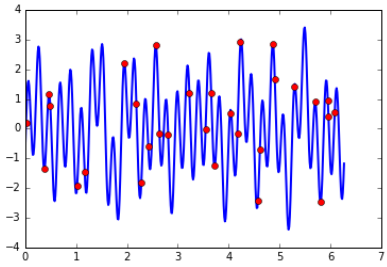
In [2]:
```
def f(x):
    return 1.7*np.sin(30.*x)+0.5*np.cos(9.*x)+0.5*np.sin(6.*x)
    -np.cos(11.*x)+0.2*np.sin(13.*x)
```

In [3]:
```
n = 512
T = 2*np.pi/n
xx = np.linspace(0,2*np.pi-T,n)
yy = f(xx)
% matplotlib inline
plt.plot(xx,yy,linewidth=3)
plt.show()
```



In [4]:
```
m = 30
p = rnd.permutation(n)
points = xx[p[:m]]
samples = f(points)
plt.plot(xx,yy,linewidth=2)
plt.plot(points,samples,'o', color='red')
plt.show()
```

The red dots indicate the points that we see. We know nothing else about the signal!



We now show how to reconstruct the whole blue curve from the knowledge of the red dots alone. We do this by setting up an optimization problem of the form

$$\text{minimize} \quad \|\boldsymbol{x}\|_1 \qquad \text{subject to} \quad \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$$

for suitable matrix $\boldsymbol{A}$ and vector $\boldsymbol{b}$. How $\boldsymbol{A}$ and $\boldsymbol{b}$ are constructed is described in the problem description. Below is the implementation.

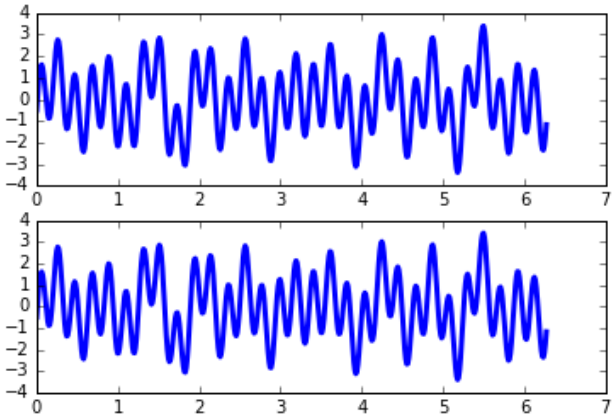
In [5]:
```
D = fft.ifft(np.eye(n))
rD = np.concatenate((D.real, D.imag), axis=1)
A = rD[p[:m],:]
fy = fft.fft(yy)
b = np.dot(A,np.concatenate((fy.real, fy.imag), axis=0))
```

In [6]:
```
x = cvx.Variable(2*n)
constraints = [A*x == b]
obj = cvx.Minimize(cvx.norm(x,1))
prob = cvx.Problem(obj, constraints)
prob.solve()

x = np.array(x.value).transpose()[0]
```

In [7]:
```
newy_im = fft.ifft(x[:n]+1j*x[n:])
newy = newy_im.real
print la.norm(newy-yy,1)

plt.subplot(2,1,1)
plt.plot(xx,yy,linewidth=3)
plt.subplot(2,1,2)
plt.plot(xx,newy,linewidth=3)
plt.show()
```



The error obtained is of order 10^{-7}. Now the interesting question is: **how much undersampling can we get away with?** To find out, we can repeat the previous experiment with values of m between 1 and 512 and find out where the method starts working. Obviously sampling only one point will not work (not enough information), and sampling all 512 points will work (we have all the information). As we say, 30 points is already sufficient, but can we do with less?