EECS 16A Designing Information Devices and Systems I
Fall 2016 Babak Ayazifar, Vladimir Stojanovic Homework 11

This homework is due Nov 15, 2016, at 1PM.

1. **Homework process and study group**

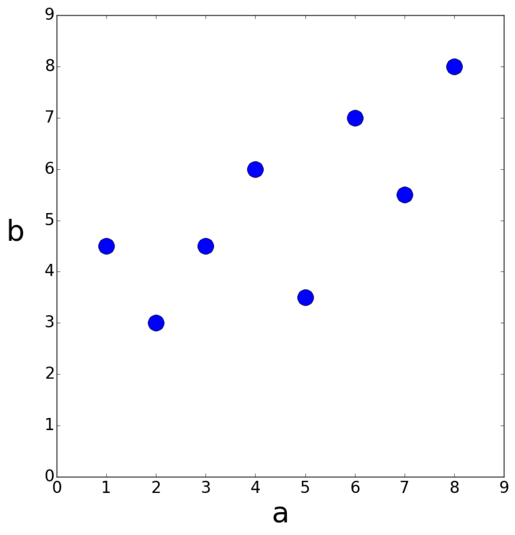
   Who else did you work with on this homework? List names and student ID's. (In case of hw party, you can also just describe the group.) How did you work on this homework?
   Working in groups of 3-5 will earn credit for your participation grade.

   Solution: I worked on this homework with...

   I first worked by myself for 2 hours, but got stuck on Problem 5 so I went to office hours on...

   Then I went to homework party for a few hours, where I finished the homework.

2. **Mechanical: Linear Least Squares**



a	1	2	3	4	5	6	7	8
b	4.5	3	4.5	6	3.5	7	5.5	8

   (a) Consider the above data points. Find the linear model of the form

$$b = xa \tag{1}$$

   that best fits the data, i.e. find x to minimize

$$\left\| \begin{bmatrix} b_1 \\ \vdots \\ b_8 \end{bmatrix} - \begin{bmatrix} a_1 \\ \vdots \\ a_8 \end{bmatrix} x \right\|^2 \tag{2}$$

Do not use Python for this calculation and show your work. (A calculator is okay). Once you've computed x, compute the squared error between your model's prediction and the actual b values as shown in Equation (**??**). Plot the best fit line along with the data points to examine the quality of the fit. (If you're plotting by hand, it is okay if your plot of $b = xa$ is approximate.)

Solution:

Define $\vec{a} = [1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7,\ 8\ ]^T$ and $\vec{b} = [4.5,\ 3,\ 4.5,\ 6,\ 3.5,\ 7,\ 5.5,\ 8\ ]^T$. Applying the linear least squares formula, we get

$$\hat{x} = (\vec{a}^T\vec{a})^{-1}\vec{a}^T b \tag{3}$$

$$= \left(\begin{bmatrix} 1\\2\\3\\4\\5\\6\\7\\8 \end{bmatrix}^T \begin{bmatrix} 1\\2\\3\\4\\5\\6\\7\\8 \end{bmatrix} \right)^{-1} \begin{bmatrix} 1\\2\\3\\4\\5\\6\\7\\8 \end{bmatrix}^T \begin{bmatrix} 4.5\\3\\4.5\\6\\3.5\\7\\5.5\\8 \end{bmatrix} \tag{4}$$

$$= (204)^{-1}(210) = 1.0294 \tag{5}$$

The error between the model's prediction and actual b values is

$$\vec{e} = \vec{b} - \vec{\hat{b}} = \vec{b} - \hat{x}\vec{a} \tag{6}$$

$$= \begin{bmatrix} 4.5\\3\\4.5\\6\\3.5\\7\\5.5\\8 \end{bmatrix} - 1.0294 \begin{bmatrix} 1\\2\\3\\4\\5\\6\\7\\8 \end{bmatrix} = \begin{bmatrix} 3.47\\0.94\\1.41\\1.88\\-1.65\\0.82\\-1.71\\-0.24 \end{bmatrix} \tag{7}$$

and the sum of squared errors is

$$\vec{e}^T\vec{e} = 24.82 \tag{8}$$

See `sol10.ipynb` for plots.

(b) You will notice from your graph that you can get a better fit by adding a b-intercept. That is we can get a better fit for the data by assuming a linear model of the form

$$b = x_1 a + x_2 \tag{9}$$

In order to do this, we need to augment our A matrix for the least squares calculation with a column of 1's (do you see why?) so that it has the form

$$A = \begin{bmatrix} a_1 & 1 \\ \vdots & \vdots \\ a_8 & 1 \end{bmatrix} \tag{10}$$

Find x_1 and x_2 to minimize

$$\left\| \begin{bmatrix} b_1 \\ \vdots \\ b_8 \end{bmatrix} - \begin{bmatrix} a_1 & 1 \\ \vdots & \vdots \\ a_8 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \right\|^2 \tag{11}$$

Again, do not use python for this calculation and show your work. A calculator is okay but take the inverse by hand using the formula for a 2×2 inverse.

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \frac{1}{ad - bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix} \tag{12}$$

Compute the squared error between your model's prediction and the actual b values as shown in Equation (**??**). Plot your new linear model. Is it a better fit for the data?

Solution:

Let $\vec{x} = [\ x_1,\ x_2\ ]^T$. Using the linear least squares formula with the new augmented A matrix, we calculate the optimal approximation of $\vec{x}$ as

$$\hat{\vec{x}} = \left(A^T A\right)^{-1} A^T \vec{b} \tag{13}$$

$$= \left(\begin{bmatrix} 1 & 1 \\ 2 & 1 \\ 3 & 1 \\ 4 & 1 \\ 5 & 1 \\ 6 & 1 \\ 7 & 1 \\ 8 & 1 \end{bmatrix}^T \begin{bmatrix} 1 & 1 \\ 2 & 1 \\ 3 & 1 \\ 4 & 1 \\ 5 & 1 \\ 6 & 1 \\ 7 & 1 \\ 8 & 1 \end{bmatrix} \right)^{-1} \begin{bmatrix} 1 & 1 \\ 2 & 1 \\ 3 & 1 \\ 4 & 1 \\ 5 & 1 \\ 6 & 1 \\ 7 & 1 \\ 8 & 1 \end{bmatrix}^T \begin{bmatrix} 4.5 \\ 3 \\ 4.5 \\ 6 \\ 3.5 \\ 7 \\ 5.5 \\ 8 \end{bmatrix} \tag{14}$$

$$= \begin{bmatrix} 204 & 36 \\ 36 & 8 \end{bmatrix}^{-1} \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 4.5 \\ 3 \\ 4.5 \\ 6 \\ 3.5 \\ 7 \\ 5.5 \\ 8 \end{bmatrix} \tag{15}$$

$$= \frac{1}{204(8) - 36(36)} \begin{bmatrix} 8 & -36 \\ -36 & 204 \end{bmatrix} \begin{bmatrix} 210 \\ 42 \end{bmatrix} \tag{16}$$

$$\hat{\vec{x}} = \begin{bmatrix} \hat{x}_1 \\ \hat{x}_2 \end{bmatrix} = \frac{1}{336} \begin{bmatrix} 168 \\ 1008 \end{bmatrix} = \begin{bmatrix} 0.5 \\ 3 \end{bmatrix} \tag{17}$$

The linear model's prediction of $\vec{b}$ is given by

$$\hat{\vec{b}} = A\hat{\vec{x}} = \begin{bmatrix} 1 & 1 \\ 2 & 1 \\ 3 & 1 \\ 4 & 1 \\ 5 & 1 \\ 6 & 1 \\ 7 & 1 \\ 8 & 1 \end{bmatrix} \begin{bmatrix} 0.5 \\ 3 \end{bmatrix} = \begin{bmatrix} 3.5 \\ 4 \\ 4.5 \\ 5 \\ 5.5 \\ 6 \\ 6.5 \\ 7 \end{bmatrix} \tag{18}$$

and the error is given by

$$\vec{e} = \vec{b} - \hat{\vec{b}} = \begin{bmatrix} 1 & -1 & 0 & 1 & -2 & 1 & -1 & 1 \end{bmatrix}^T \tag{19}$$

And the summed squared error is

$$\vec{e}^T \vec{e} = 10 \tag{20}$$

See `sol10.ipynb` for plots.

We can see both qualitatively from the plots and quantitatively from the sum of the squared errors that the fit is better with the b-intercept.

(c) Let $\hat{\vec{x}}$ be the solution to a general linear least squares problem,

$$\hat{\vec{x}} = \underset{\vec{x}}{\operatorname{argmin}} \left|\left|\vec{b} - A\vec{x}\right|\right|^2 \tag{21}$$

Show that the error vector $\vec{b} - A\hat{\vec{x}}$ is orthogonal to the columns of A by direct manipulation. *(i.e. plug in the formula for the linear-least-squares estimate into the error vector. And then see if this vector is such that A^T times it is zero.)*

Solution:

We want to show that the error in the linear least squares estimate is orthogonal to the columns of the A, i.e. we want to show that $A^T(\vec{b} - A\hat{\vec{x}})$ is the zero vector. Plugging in the linear least squares formula for $\hat{\vec{x}}$, we get

$$A^T\left(\vec{b} - A\hat{\vec{x}}\right) = A^T\left(\vec{b} - A\left(A^TA\right)^{-1}A^T\vec{b}\right) \tag{22}$$

$$= A^T\vec{b} - A^TA\left(A^TA\right)^{-1}A^T\vec{b} \tag{23}$$

$$= A^T\vec{b} - IA^T\vec{b} \tag{24}$$

$$= A^T\vec{b} - A^T\vec{b} = \mathbf{0} \tag{25}$$

3. Labelling patients using gene expression data

Least-squares techniques are useful for many different kinds of prediction problems. The core ideas we learned in class have been extensively further developed. These ideas are commonly used in machine learning for finance, healthcare, advertising, image processing and many other fields. Here we'll explore how least squares can be used for classification of data in a medical context.

Gene expression data of patients, along with other factors such as height, weight, age, family history, is often used to understand the likelihood that a patient might develop certain common diseases such as diabetes. Gene expression profiles can be read using DNA microarray technology, which uses tissue samples from a patient. This data, along with the patient specific characteristics above, can be combined into a vector to get a set of features that describe each patient.

Many scientific studies look at models in mice to understand how gene expression relates to diabetes. Previous studies have shown that the expression of the tomosin2 and ts1 genes are correlated to the onset of diabetes in mice. How can we predict whether or not a mouse will develop diabetes based on data about this expression as well as other factors of the mouse? We will use some (fake) data to explore this.

We are given information about the age and weight of the mouse, and additionally have access to data about whether the genes tomosin2, ts1 and chn1 (a third gene) were expressed or not. The gene expression data is captured using vectors that are $+1$ if the gene is expressed and -1 if the gene is not expressed. Similarly, whether or not a mouse has diabetes is also captured using a $+1,-1$ vector, where $+1$ indicates that the mouse has diabetes. Using this data we would like to develop a linear model that predicts whether or not a mouse will have diabetes.

$$\alpha_1(\text{age}) + \alpha_2(\text{weight}) + \alpha_3(\text{tomosin2}) + \alpha_4(\text{ts1}) + \alpha_5(\text{chn1}) \tag{26}$$

We would like the above expression to be positive if the mouse has diabetes and negative if the mouse does not have diabetes.

(a) In problems such as this, it is common to use some *training* data to generate a model. Turns out, a good heuristic for this can be developed using a least squares technique. Set up a linear model for the problem in a format we have used for least-squares problems $A\vec{x} = \vec{b}$. Here, $\vec{b}$ will be a vector with $+1,-1$ entires. α_i's are your unknowns.

Solution:

The unknowns that we want to find are α_i, where $\{i = 1,\ldots,5\}$. The data we are given has the age, weight and expressions of the genes tomosin2, ts1, and chn1. So the matrix A would be the matrix with each row containing the data of each mouse. The vector $\vec{b}$ is a column vector indicating if the mouse has diabetes or not. And the vector $\vec{x}$ is the vector of unknows $\vec{x} = \begin{bmatrix}\alpha_1 & \alpha_2 & \alpha_3 & \alpha_4 & \alpha_5\end{bmatrix}^T$. Hence the setup is:

$$\underbrace{\begin{bmatrix} \text{age} & \text{height} & \text{tomosin2} & \text{ts1} & \text{chn1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix}}_{A} \underbrace{\begin{bmatrix}\alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \\ \alpha_5\end{bmatrix}}_{\vec{x}} = \underbrace{\begin{bmatrix} b_1 \\ b_2 \\ \vdots \end{bmatrix}}_{\vec{b}}$$

(b) Using the (fake) *training* data `diabetes_train.npy`, generate the linear model using the least squares technique, i.e. find the unknown model parameters for the given data set. Include the unknown parameter values in the writeup of your homework. Use the provided ipython notebook file.

Solution:

To solve for $\vec{x}$ in $A\vec{x} = \vec{b}$ using the least squares technique, we find $(A^TA)^{-1}A^T\vec{b}$. The result is

$$\vec{x} = \begin{bmatrix}\alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \\ \alpha_5\end{bmatrix} = \begin{bmatrix} -0.15646169 \\ 0.09239418 \\ 0.48053974 \\ -0.5847018 \\ -0.35350734 \end{bmatrix}$$

(c) Now it is time to use the model you have developed to make some predictions! It is interesting to note here that we are not looking for a real number to model whether each mouse has diabetes or not, we are looking for a binary label. So we will use the *sign* of the expression above to assign a ± 1 value to each mouse. Predict whether each mouse with the characteristics in the *test* data set `diabetes_test.npy` will get diabetes. There are four mice in the test data set. Include the ± 1 vector that indicates whether or not they have diabetes in your writeup.

Solution:

Using the values of α_i calculated in the previous part on the test data, we see that the prediction is $\vec{b} = \begin{bmatrix} 1 & -1 & -1 & 1 \end{bmatrix}^T$ which is exactly the values given to us in `diabetes_test.npy`

4. Image analysis

Applications in medical imaging often require an analysis of images based on the pixels of the image. For instance, we might want to count the number of cells in a given sample. One way to do this is to "take a picture" of the cells and use the pixels to determine the locations and thus the number of cells. Alternatively, automatic detection of shape is useful in image classification as well (e.g. consider a robot autonomously trying to find out where a mug is in it's field of vision).

Let us focus back on the medical imaging scenario. You are interested in finding the exact position and shape of a cell in an image, so you want to find the equation of the ellipse that bounds the cell relative to a given coordinate system that is represented by the image. Your collaborator uses edge detection techniques to find a bunch of points that are approximately along the edge of the cell. We assume that the origin is in the center of the image with standard axes and collect the following points: $(.3, -.7)$, $(.5, .91)$, $(.9, -.99)$, $(1, 1.01)$, $(1.2, -.93)$, $(1.5, .8)$, $(2, 0)$. Submit your code for all parts of this problem, feel free to add it to the original iPython notebook.

Recall that a quadratic equation of the form

$$ax^2 + bxy + cy^2 + dx + ey = 1. \tag{27}$$

can be used to represent an ellipse (if $b^2 - 4ac < 0$), and a quadratic equation of the form

$$a(x^2 + y^2) + dx + ey = 1 \tag{28}$$

is a circle if $d^2 + e^2 - 4a > 0$. The circle has fewer parameters.

(a) How can you find the equation of a circle that surrounds the cell? First, provide a setup and formulate a set of matrix equations to do this, i.e. an equation of the form $A \cdot \vec{x} = \vec{b} + \vec{e}$, where $\vec{b}$ represents your observations and $\vec{e}$ represents the unknown errors.

Solution: The setup is:

$$\begin{bmatrix} x^2 + y^2 & x & y \\ \vdots & \ddots & \vdots \end{bmatrix} \cdot \begin{bmatrix} a \\ d \\ e \end{bmatrix} = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}$$

We plug in numbers to get:

$$\begin{bmatrix} 0.58 & 0.3 & -0.7 \\ 1.0781 & 0.5 & 0.91 \\ 1.791 & 0.9 & -0.99 \\ 2.0201 & 1 & 1.01 \\ 2.3049 & 1.2 & -0.93 \\ 2.89 & 1.5 & 0.8 \\ 4 & 2 & 0 \end{bmatrix} \cdot \begin{bmatrix} a \\ d \\ e \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

(b) How can you find the equation of an ellipse that surrounds the cell? Provide a setup and formulate a set of matrix equations to do this as above.

Solution: The setup is:

$$\begin{bmatrix} x^2 & xy & y^2 & x & y \\ \vdots & \ddots & \ddots & \ddots & \vdots \end{bmatrix} \cdot \begin{bmatrix} a \\ b \\ c \\ d \\ e \end{bmatrix} = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}$$

We then plug in values to get:

$$\begin{bmatrix} 0.09 & -0.21 & 0.49 & 0.3 & -0.7 \\ 0.25 & 0.455 & 0.8281 & 0.5 & 0.91 \\ 0.81 & -0.891 & 0.9801 & 0.9 & -0.99 \\ 1 & 1.01 & 1.0201 & 1 & 1.01 \\ 1.44 & -1.116 & 0.8649 & 1.2 & -0.93 \\ 2.25 & 1.2 & 0.64 & 1.5 & 0.8 \\ 4 & 0 & 0 & 2 & 0 \end{bmatrix} \cdot \begin{bmatrix} a \\ b \\ c \\ d \\ e \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

(c) Write a short program in iPython to fit a circle using these points. If you model your system of equations as $A\vec{x} = \vec{b} + \vec{e}$ where $\vec{e}$ is the error vector and the number of data points is N, what is $\frac{\|\vec{e}\|}{N}$? Plot your points and the best fit circle in iPython.

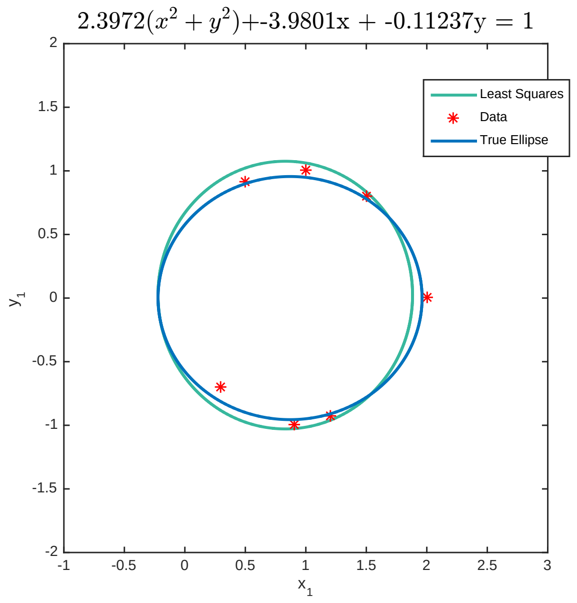
Solution:

```
import numpy as np
a = np.matrix('0.58 0.3 -0.7; \
              1.0781 0.5 0.91; \
              1.791 0.9 -0.99; \
              2.0201 1 1.01; \
              2.3049 1.2 -0.93; \
              2.89 1.5 0.8; \
              4 2 0')

b = np.ones((7, 1))
result = np.linalg.inv(a.transpose() * a) * a.transpose() * b
print(result)
```

The solution vector is:

$$\begin{bmatrix} 2.4012 \\ -3.9881 \\ -0.1120 \end{bmatrix}$$

Thus, we would predict the equation of the circle to be: $2.4012(x^2+y^2)-3.9881x-0.1120y=1$. This gives the normalized error: $\frac{1.145}{7}=0.1636$.



(d) Write a short program in iPython to fit an ellipse using these points. If you model your system of equations as $A\vec{x}=\vec{b}+\vec{e}$ where $\vec{e}$ is the error vector and the number of data points is N, what is $\frac{\|\vec{e}\|}{N}$? Plot your points and the best fit ellipse in iPython. How does this error compare to the one in the previous subpart? Which technique is better?

Solution:

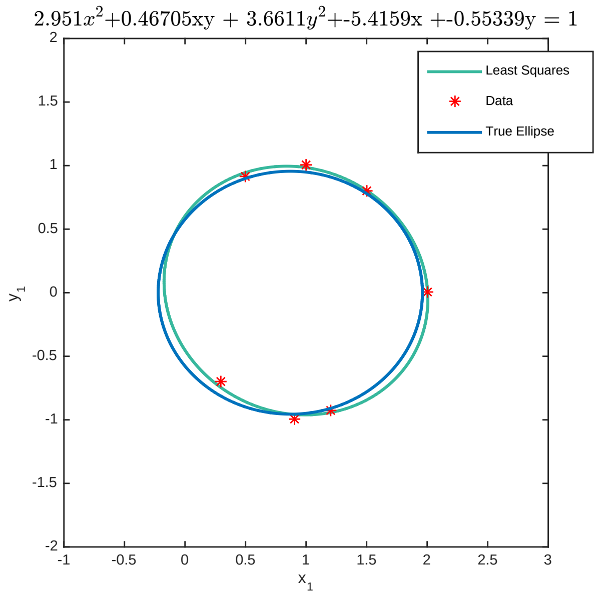
```
import numpy as np
a = np.matrix('0.09 -0.21 0.49 0.3 -0.7; \
              0.25 0.455 0.8281 0.5 0.91; \
              0.81 -0.891 0.9801 0.9 -0.99; \
              1 1.01 1.0201 1 1.01; \
              1.44 -1.116 0.8649 1.2 -0.93; \
              2.25 1.2 0.64 1.5 0.8; \
              4 0 0 2 0')

b = np.ones((7, 1))
result = np.linalg.inv(a.transpose() * a) * a.transpose() * b
print(result)
```

The solution vector is:

$$\begin{bmatrix} 2.9510 \\ 0.4671 \\ 3.6611 \\ -5.4159 \\ -0.5534 \end{bmatrix}$$

We predict the general equation to be: $2.9510x^2+0.4671xy+3.6611y^2-5.4159x-0.5534y=1$. This gives the normalized error: $\frac{0.4767}{7}=0.0681$.

The ellipse is a better fit because it has more parameters, so the least squares technique can tune the parameters to be closer to the observations. This is similar to how a higher degree polynomial has greater degrees of freedom.



5. GPS Receivers

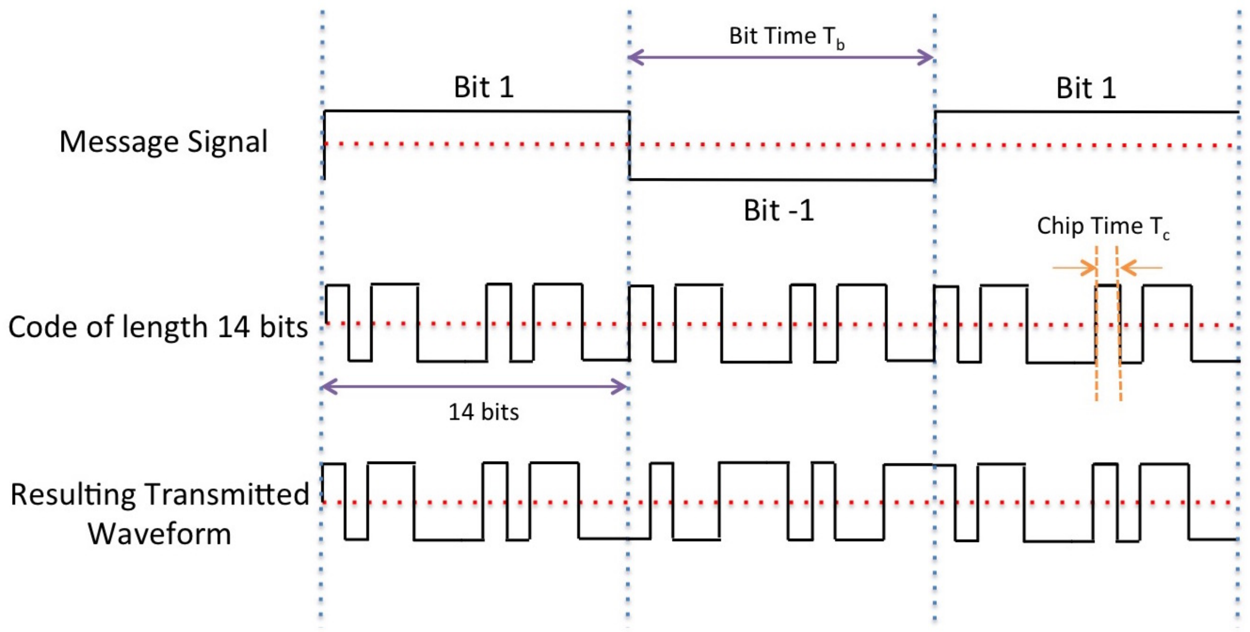
The Global Positioning System (GPS) is a space-based satellite navigation system that provides location and time information in all weather conditions, anywhere on or near the Earth where there is an unobstructed line of sight to four or more GPS satellites. Gold All satellites use the same carrier frequency to transmit the signals. To permit this without undue interference between the users, the satellites employ "spread-spectrum" technology (very similar to CDMA) and a special coding scheme where each transmitter is assigned a code that serves as a "signature". An example is depicted in the figure below. The *message signal* to be transmitted changes at a much slower timescale than the timescale of the signature code, which is very fast.

In the example figure, we are showing a message signal that is a stream of $+1$ or -1 values. The signature code is also a stream of $+1$ or -1 values of length 14 bits. The signature code is multiplied by the appropriate *message signal* to get the final transmitted waveform. Let T_b be the "bit time", i.e. time for each message bit and and T_c be the "chip time" which is the time for each new symbol of the code to be generated. In the figure, $T_b = 14T_c$.

The GPS satellites use "Gold codes" which are 1023 bits long. So, for our problem, $T_b = 1023T_c$. (In reality, $T_b = 20 \times 1023 \times T_c$.) The Gold codes have special properties:

- Their auto-correlation of a Gold code (correlation with itself) is very high.
- The cross-correlation between different codes is very low.

These codes are generated using a linear feedback shift register (LFSR).You can read more about this if you are interested but for the problem you don't need to know how this works. The take-away is that the Gold codes are vectors of $+1$ and -1 values that are are "almost orthogonal".



For the purpose of this question we only consider 24 GPS satellites. Download the file `prob10.ipynb` and the corresponding data files for the following questions:

(a) Auto-correlate the Gold code of satellite 10 with itself and plot it. Python has functions for this. What do you observe?

Solution: Solutions in `sol11.ipynb`

(b) Cross-correlate the Gold code of satellite 10 with satellite 13 and plot it. What do you observe?

Solution: Solutions in `sol11.ipynb`

(c) Now, consider a random signal, i.e. a signal that is not generated due to a specific code but is a random ± 1 sequence. Cross-correlate it with the Gold code of satellite 10. What do you observe? How does this compare to the cross-correlation of satellite 10 and satellite 13? What does this mean about our ability to identify satellites?

Solution: Solutions in `sol11.ipynb`

(d) The signals received by a receiver include signals from the satellites as well as an additional noise term. This is often modeled as a Gaussian noise term. You don't need to understand Gaussians here but we use them since they form a good model for how the transmitted signal might be perturbed (large perturbations are very unlikely, and small perturbations are more likely).

Use the Gaussian noise generator to generate a random vector of length 1023, and cross correlate this with the Gold code of satellite 10. What do you observe?

For the next subparts of this problem, the signal is corrupted by Gaussian noise. Use the observation from this subpart for solving the rest of the question.

Solution: Solutions in `sol11.ipynb`

(e) Now, assume that signals from multiple satellites are added at the receiver. So the signatures of multiple different satellites are present in the code. In addition, noise might be added to the signal. What are the satellites present in `data1.npy`?

Solution: Solutions in `sol11.ipynb`

(f) Let's assume that you can hear only one satellite, Satellite A, at the location you are in (though this never happens in reality). Let's also assume that this satellite is transmitting a length 5 sequence of $+1$ and -1 after modulating it onto the 1023 bit Gold code corresponding to Satellite A. Find out from `data2.npy` which satellite it is and what sequence of ± 1 it is transmitting.

Solution: Solutions in `sol11.ipynb`

(g) For the purpose of this problem, we'll assume that all the satellites transmit the same unique sequence of $+1$s and -1s. These are transmitted using the procedure described in the figure (called modulation, which we will learn more about soon.)

Signals from different transmitters arrive at the receiver with different propagation delays. So effectively the signals from different satellites are superimposed on each other with different offsets at the start. This propagation delay is used to find out how far the satellite is from the receiver. To find out exactly what the offset due to propagation delay is, you want to figure out the starting point of the signal transmission, and you can do this by cross-correlating the signature codes with the received signal at different offsets. What do you expect to observe when you cross-correlate the signature for a particular satellite (say Satellite A) with the received signal at the offset corresponding to the propagation delay of Satellite A?

What satellites are you able to see in `data3.npy` and what are the relative delays assuming that the message signal that was being sent was exactly $\begin{bmatrix}1 & 1 & -1 & -1 & -1\end{bmatrix}$.

Solution: Solutions in `sol11.ipynb`

6. Your Own Problem Write your own problem related to this week's material and solve it. You may still work in groups to brainstorm problems, but each student should submit a unique problem. What is the problem? How to formulate it? How to solve it? What is the solution?