# In The Name Of ALLAH

## Advanced Programming 1398 - 2

## Final Exam

## Dr.Jahanshahi

**Please note that unusual similarity between codes would cause both students to fail.**
**Just by attending the classes and doing homeworks you would be able to do well today.**
**Good Luck.**


# Part 1: asyncio

In this question we are going to fetch some urls using `aiohttp` library which is based on `asyncio`. The program should be able to fetch the data from multiple link simultaneously. The links have this

pattern:

```
http://tsetmc.ir/Loader.aspx?ParTree=151311&i={}
```

# Generating links

Please insert the integer codes in `inscode.txt` file in place of the brackets `{}` and generate all the urls.

# Fetch URLS

Fetch all the urls simultaneously. Please see the documentation of `aiohttp` here and here. Please note that the code needs to fetch all urls in parallel. You can use your knowledge about `asyncio` to do this.

It is necessary to check the status code of the html response to be `200`. Similar codes can be found in the documentations provided above.

# Get TradeHistory

In `response.text()` variable, there is a `var TradeHistory=[[...]]` part. For illustration purposes you can see the relevant part in the image below which is captured in browser. The goal is to print the `Arzesh` column only for the last day in numeric format.

سابقه معاملات

| تاریخ | پایانی | | کمترین | بیشترین | تعداد | حجم | ارزش |
|---|---|---|---|---|---|---|---|
| 1399/4/22 - برای مشاهده سابقه معاملات روز جاری کلیک کنید | | | | | | | |
| 1399/4/21 | 24,590 | (3.49) | 24,210 | 26,750 | 63,551 | 223.768 M | 5,501.8 B |
| 1399/4/18 | 25,480 | (4.5) | 25,350 | 28,010 | 49,476 | 209.557 M | 5,339.166 B |
| 1399/4/17 | 26,680 | (0.67) | 25,910 | 28,200 | 67,774 | 192.808 M | 5,144.011 B |
| 1399/4/16 | 26,860 | 4.64 | 26,200 | 26,950 | 52,045 | 253.566 M | 6,812.022 B |
| 1399/4/15 | 25,670 | (0.35) | 24,480 | 27,040 | 65,488 | 224.65 M | 5,766.736 B |
| 1399/4/14 | 25,760 | (4.38) | 25,600 | 27,000 | 65,195 | 231.284 M | 5,958.222 B |
| 1399/4/11 | 26,940 | 3.18 | 26,500 | 27,370 | 56,583 | 168.22 M | 4,531.903 B |
| 1399/4/10 | 26,110 | 4.73 | 25,000 | 26,170 | 38,097 | 147.166 M | 3,841.979 B |

## Generating Output

Please generate an output similar to this:

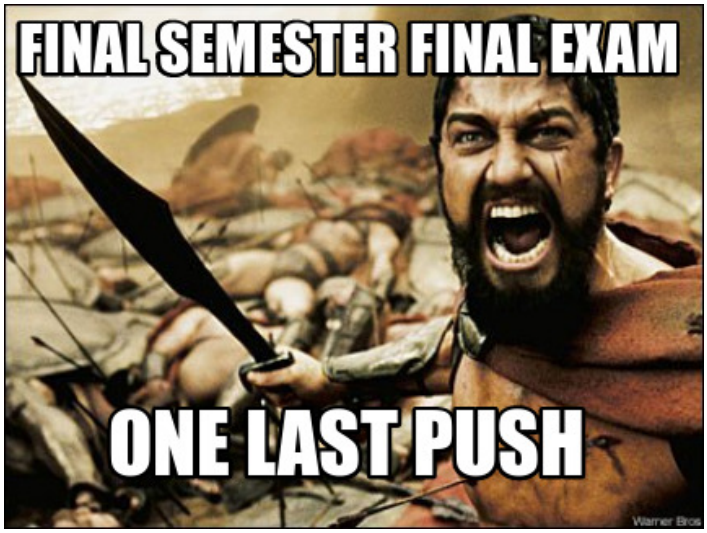
```
Fetch http://tsetmc.ir/Loader.aspx?ParTree=151311&i=33629260529503413 -> Took: 68 ms -> 1026011646873
Fetch http://tsetmc.ir/Loader.aspx?ParTree=151311&i=12329915567896606 -> Took: 75 ms -> 1418842491561
Fetch http://tsetmc.ir/Loader.aspx?ParTree=151311&i=51617145873056483 -> Took: 77 ms -> 17141643852
Fetch http://tsetmc.ir/Loader.aspx?ParTree=151311&i=5987841496184505 -> Took: 79 ms -> 31628088542
.
.
.

Total took: 80 ms
```

*Note that the Total time is not the summation of the fetching times.*

*The total time should only consider fetching all urls and finding Arzesh column, not any other code*

**The code for this excersize is relatively short if carefully thought**




# Part 2: AUT Device

We've made a range estimator device. It transmits a sinusoidal function to an object and receives its reflection. Based on what we get at the receiver we want to find the distance to the object! It's super easy to implement.

# Transmitter Class

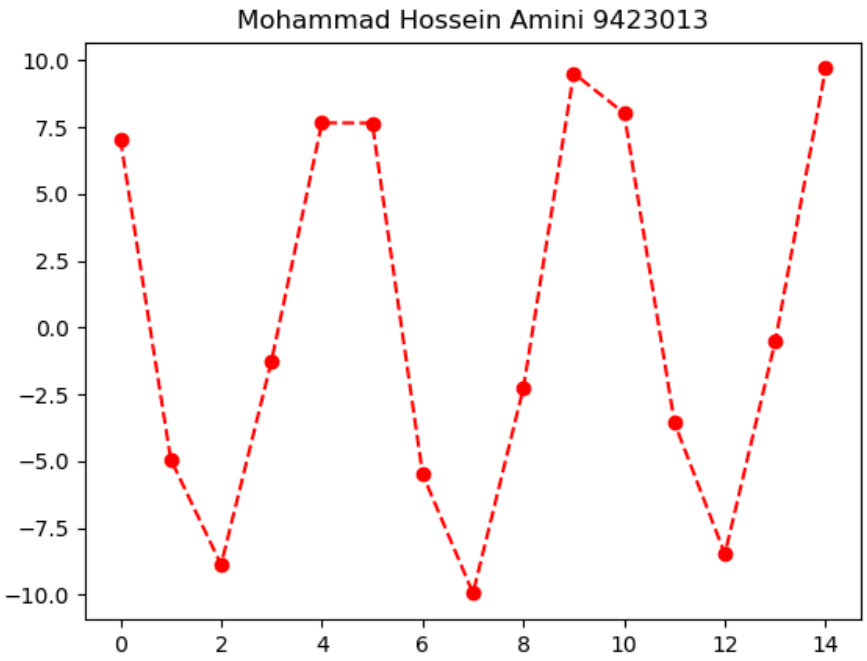
In the **Transmitter** class, you get the amplitude (**A**), the frequency (**f**) and the ($\phi$) phase of the transmitted sinusoidal signal. Of course as the signal is transmitted, it would be noise corrupted. So we assume that a WGN (White Gaussian Noise) with a known standard deviation ($\sigma$) is added to the sinusoidal. So the transmitted signal is:

$$x(n) = A.cos(2\pi fn + \phi) + v(n) \qquad , n = 0, \dots, N-1$$

where v is the noise.

This class has the following functions:

- *_init_*(self, A=10, f=0.2, phi=np.pi/4, sigma=1)
- generate(N)
  This function gets the length of the signal, **N**, and generates the signal we would get on the receiver. It returns the generated signal as a numpy ndarray with shape (N,)
    - Note that this function **must** be implemented in a **single** line (other than its prototype)
- show()
  This function plots the generated signal in 3 cycles like the following figure. Don't forget to put your name as the title of the figure and save it with the **Generated.png** name.




Your class should be able to handle the following codes:

```
xmt = Transmitter()
x = xmt.generate(100)
xmt.show()
```

# Receiver Class

In the **Receiver** class, we get the received signal and estimate the range. For the range estimation we need to find out the phase of the received signal first. By assuming that we know the frequency of the the received signal (**f**) we have

$$x(n) = A.cos(2\pi fn + \phi) + v(n) \qquad , n = 0, \dots, N-1$$
$$x(n) = A.cos(\phi)cos(2\pi fn) - A.sin(\phi)sin(2\pi fn) + v(n)$$

Let's define
$\alpha = A.cos(\phi)$
$\beta = -A.sin(\phi)$

So we have

$$x(n) = \alpha cos(2\pi fn) + \beta sin(2\pi fn) + v(n)$$

This equation can be solved using the normal equation approach, which we saw in HW2, with the following relations

$$x = \begin{bmatrix} x(0) \\ x(1) \\ \vdots \\ x(N-1) \end{bmatrix}$$

$$H = \begin{bmatrix} cos(2\pi f.(0)) & sin(2\pi f.(0)) \\ cos(2\pi f.(1)) & sin(2\pi f.(1)) \\ \vdots & \vdots \\ cos(2\pi f.(N-1)) & sin(2\pi f.(N-1)) \end{bmatrix}$$

$$\theta = \begin{bmatrix} \alpha \\ \beta \end{bmatrix}$$

$$x = H\theta + v$$

$$\theta = (H^T H)^{-1} H^T x$$

By finding $\theta$ vector we would have the $\alpha$ and $\beta$. So now it's easy to find the phase.

$$\phi = tan^{-1}(-\frac{\beta}{\alpha})$$

By having the $\phi$ we can estimate range with the following formula.

$$r = \frac{c\phi}{4\pi f_c}$$

where c is the speed of light propagation which we assume c=3e8 and $f_c$ is the frequency of the generated sinusoidal before transmitting and corrupting in noise. We must know $f_c$ to estimate the range. So it would be given to you.

Implement the following functions in the **Receiver** class.

- _H(f, N)
  This function gets **f** and **n** and returns the **H** matrix mentioned above.
    - Note that this function **must** be implemented in a **single** line (other than its prototype)
- estimatePhase(f)
  This function must return the estimated phase ($\phi$) with the above formulas.
- estimateRange(f, fc)
  This function estimates the range with the above formula. It returns a tuple in which the first element is the estimated range and the second element is the estimated phase ($\phi$) used to find out the range in the formula.

You must be able to get a proper output with the following codes

```
xmt = Transmitter()
x = xmt.generate(100)
rcv = Receiver(x)
print('Range : ', rcv.estimateRange(0.2, 2e5)) # The range must be about 100
```

Since you've done your job very well, we've decided to add a new feature to our device. Now we want to estimate the received signal's frequency. This would be super easy too. To estimate the frequency, we need to compute the auto correlation of a signal. The auto correlation of a signal x with length of N is computed as

$$r(k) = \frac{1}{N-k} \sum_{i=0}^{i=N-k-1} x(i)x(i+k)$$

Since there may be large observation vectors, we would implement the above formula in our favorite language, **C++**, and use it in python. So implement the **correlator** function with the following prototype.

```
double correlator(double* x, size_t N, size_t k);
```

You must generate a dynamic link library file (**.dll** or *.so*) with the name of **final.so** to be able to use it in python. So your class must have a **correlator** function which is just a wrapper for the C++ version. (Don't forget to put your C++ codes in your submission zipfile)

Now we're able to estimate the frequency with the following formula.

$$f = \frac{1}{2\pi} cos^{-1}\left(\frac{r(1)}{r(0)}\right)$$

You must implement the **estimateFrequency** function that returns the estimated frequency using the above formula.

Make sure you've done everything well by the following code.

```
xmt = Transmitter()
x = xmt.generate(100)
rcv = Receiver(x)
print(rcv.estimateFreq())
```

Congratulations if you've reached here. Just one more tiny task. We want to see how does length of observation improve the accuracy in frequency estimation. So implement the **compare** function. This function gets some numbers as the observation lengths. For example we call it like **compare(5, 10, 100)**. First of all it generates a sinusoidal signal (with default parameters) with the length of N = 5. Use the **Transmitter** class for this. Then it estimates it's frequency using the **Receiver** class. Then repeat the same procedure for N = 10 and again for N = 100. Then you must plot a figure like the following and save it as **Freq.png**. Note that the number of arguments to your function is arbitrary. For example it might be **compare(50, 500)** or **compare(10, 20, 50, 100, 1000)**.