Time Series Analysis and Recurrent Neural Networks
Giacomo Barzon - 3626438

Exercise 1

October 29, 2019

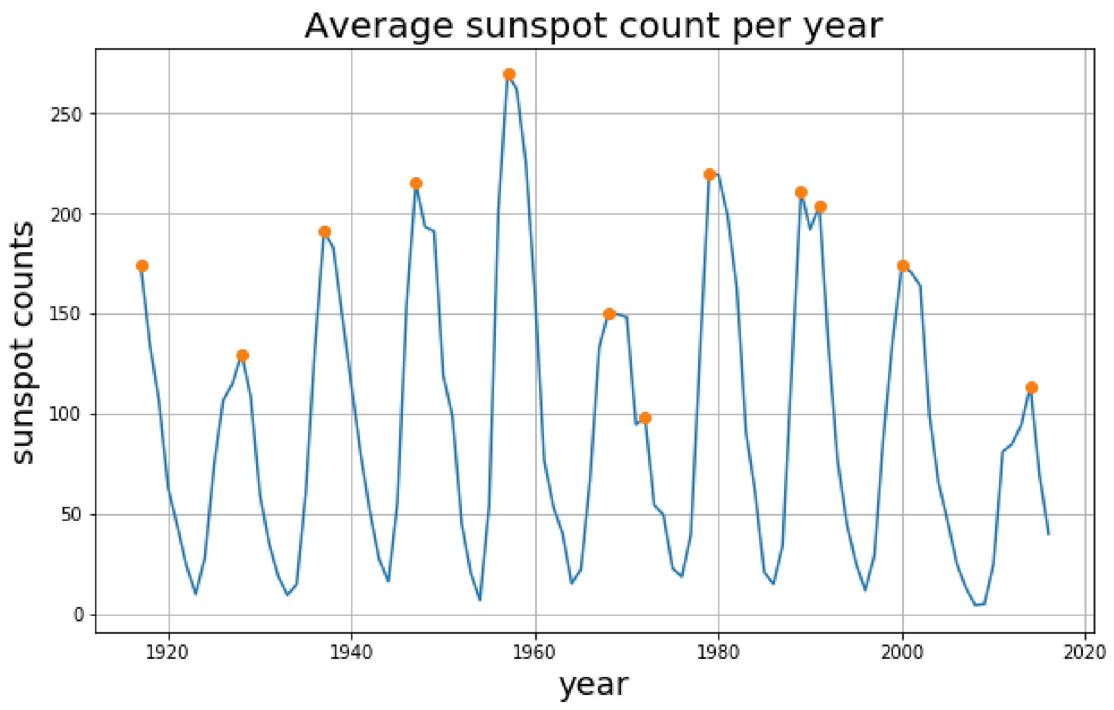
1 Exercise 1

2 Task 1

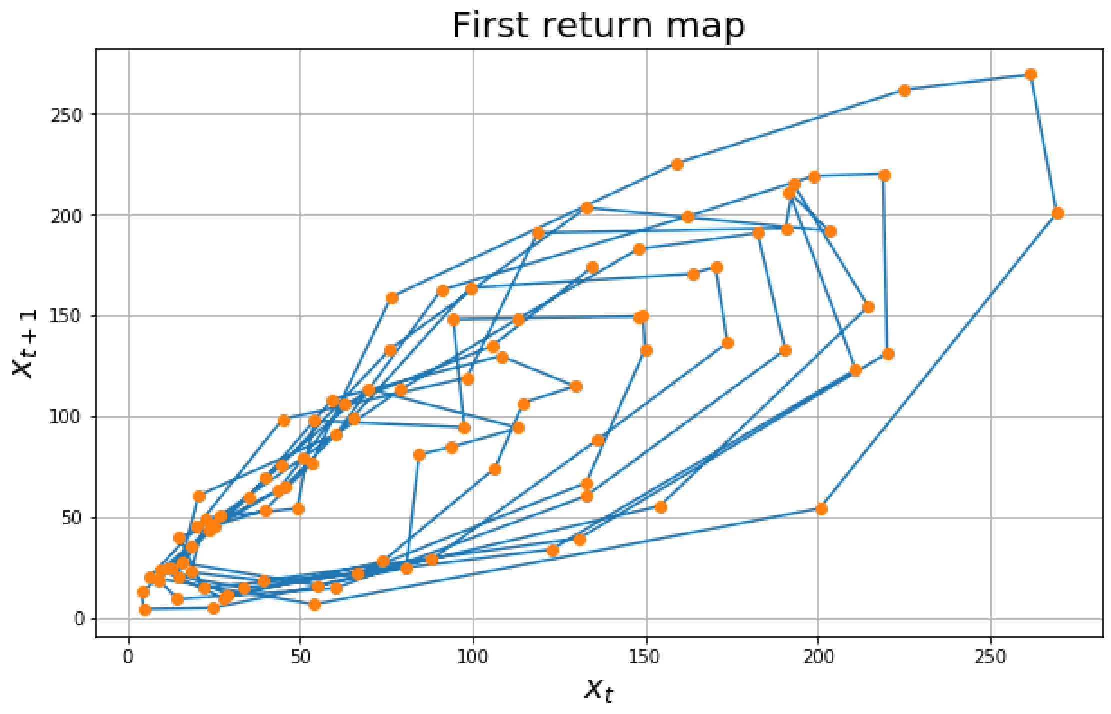
2.0.1 To smooth out day-to-day fluctuations, average observations over yearly periods. Plot the first return-map from the resulting time series. What do you notice? What is the oscillation period in years?

```
In [1]: import pandas as pd
        import matplotlib.pyplot as plt
        import numpy as np

        data = pd.read_excel('sunspotData.xls')

In [2]: # Averaging observation over yearly periods
        meanData = data.groupby(data.years).mean()

        # Find local maxima
        counts = np.array(meanData.sunspots)
        isMax = np.r_[True, counts[1:] > counts[:-1]] & np.r_[counts[:-1] > counts[1:], True]

        # Plot average data
        plt.figure(figsize=[10,6])
        plt.plot(meanData)
        plt.plot(meanData[isMax],'o')
        plt.xlabel('year', fontsize = 18)
        plt.ylabel('sunspot counts', fontsize = 18)
        plt.grid()
        plt.title('Average sunspot count per year', fontsize = 20)

        # Time btw two peaks
        print('Time btw two peaks:')
        print(np.diff(meanData[isMax].index))
        print('Average time btw two peaks:')
        print(np.mean(np.diff(meanData[isMax].index)))
```

```
Time btw two peaks:
[11  9 10 10 11  4  7 10  2  9 14]
Average time btw two peaks:
8.818181818181818
```



In [3]:
```
# Return map
xt = np.delete(counts, 0)
xt1 = np.delete(counts, meanData.sunspots.shape[0]-1)

# Plot the return map
plt.figure(figsize=[10,6])
plt.plot(xt, xt1)
plt.plot(xt, xt1,'o')
plt.xlabel('$x_t$', fontsize = 18)
plt.ylabel('$x_{t+1}$', fontsize = 18)
plt.title('First return map', fontsize = 20)
plt.grid()
```



It can be easily noticed that there are some visible and almost periodic peaks of the sunspot counts over the years, with an average time period of 9 years. The periodicity of the sunspot activity can also be seen by looking to the ciclic behaviour of the return map.

3 Task 2

3.0.1 1) By using linear regression, remove the trend from the data

```
In [4]: investments = pd.read_excel('investment.xls')
        investments.describe()
```

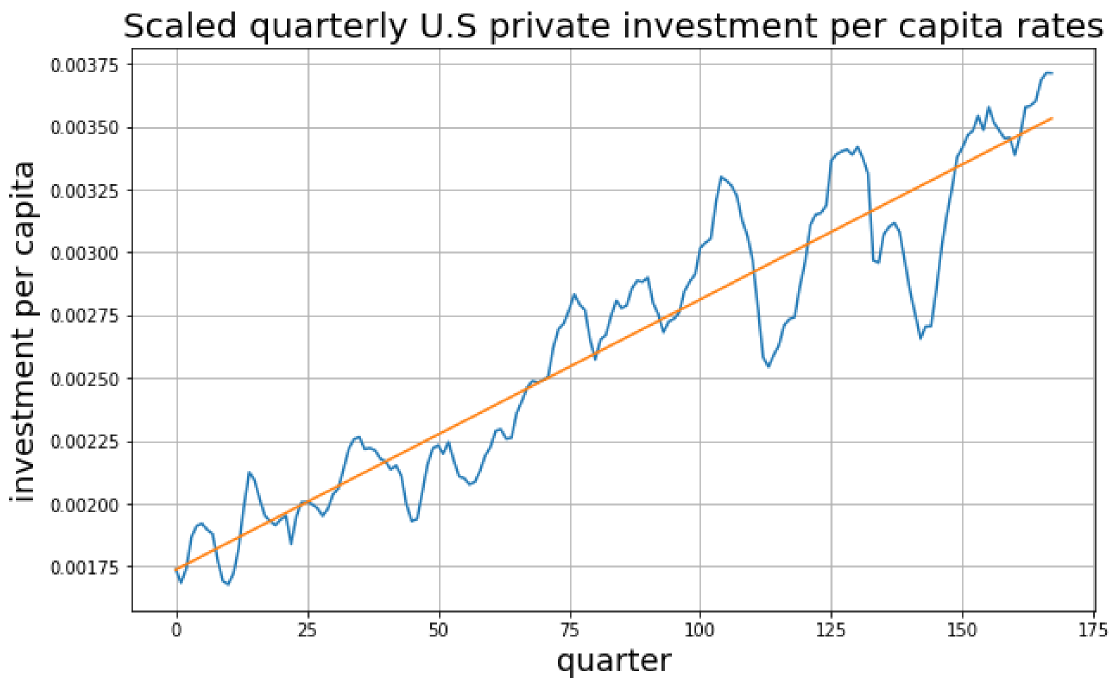
```
Out[4]:        investment
        count  168.000000
        mean     0.002635
        std      0.000558
        min      0.001677
        25%      0.002134
        50%      0.002700
        75%      0.003074
        max      0.003715
```

```
In [5]: # Linear regression
        from sklearn.linear_model import LinearRegression

        quarters = np.array(investments.index)
        y = np.array(investments['investment'])

        model = LinearRegression().fit(quarters.reshape(168,1), y)
```

```
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklearn/linear_mod
  linalg.lstsq(X, y)
```

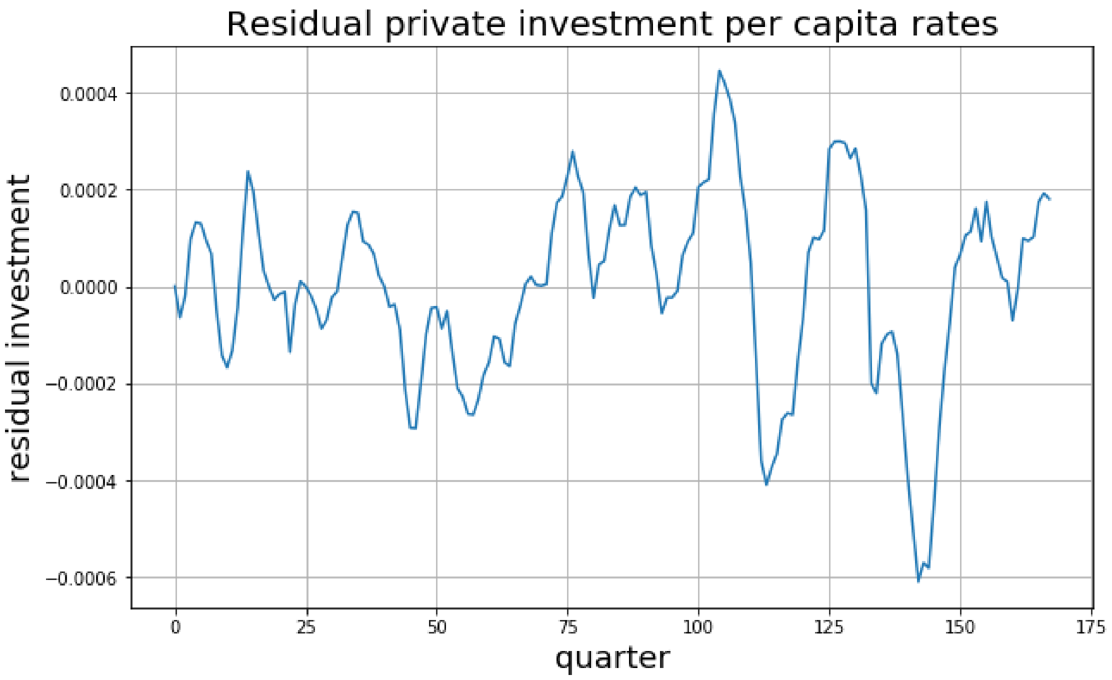
```
In [6]: # Plot average data
        plt.figure(figsize=[10,6])
        plt.plot(investments)
        plt.plot(quarters,model.predict(quarters.reshape(168,1)))
        plt.xlabel('quarter', fontsize = 18)
        plt.ylabel('investment per capita', fontsize = 18)
        plt.grid()
        plt.title('Scaled quarterly U.S private investment per capita rates', fontsize = 20)
```

```
Out[6]: Text(0.5, 1.0, 'Scaled quarterly U.S private investment per capita rates')
```



```
In [7]: # Removing the linear trend from the data
        resInvestments = y - model.predict(quarters.reshape(168,1))

        plt.figure(figsize=[10,6])
        plt.plot(quarters,resInvestments)
        plt.xlabel('quarter', fontsize = 18)
        plt.ylabel('residual investment', fontsize = 18)
        plt.grid()
        plt.title('Residual private investment per capita rates', fontsize = 20)
```

```
Out[7]: Text(0.5, 1.0, 'Residual private investment per capita rates')
```



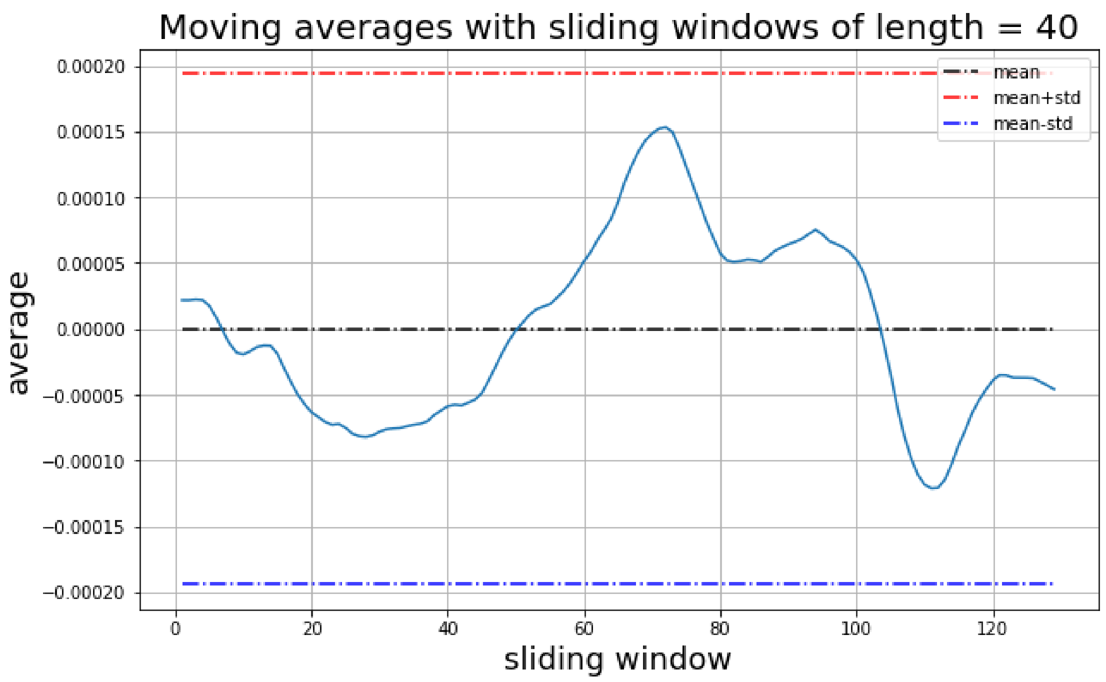
3.0.2 b) Examine whether the residual time series is approximately stationary. Is the time series of first differences stationary? How about the time series of second-order differences?

```
In [8]: # Moving average
        def moving_average(y,N):
            cumsum, moving_aves = [0], []
            n = []

            for i, x in enumerate(y, 1):
                cumsum.append(cumsum[i-1] + x)
                if i>=N:
                    moving_ave = (cumsum[i] - cumsum[i-N])/N
                    #can do stuff with moving_ave here
                    moving_aves.append(moving_ave)
                    n.append(i-N+1)
            return (n, moving_aves)
```

```
In [9]: # Trend removed moving average
        N = 40
```

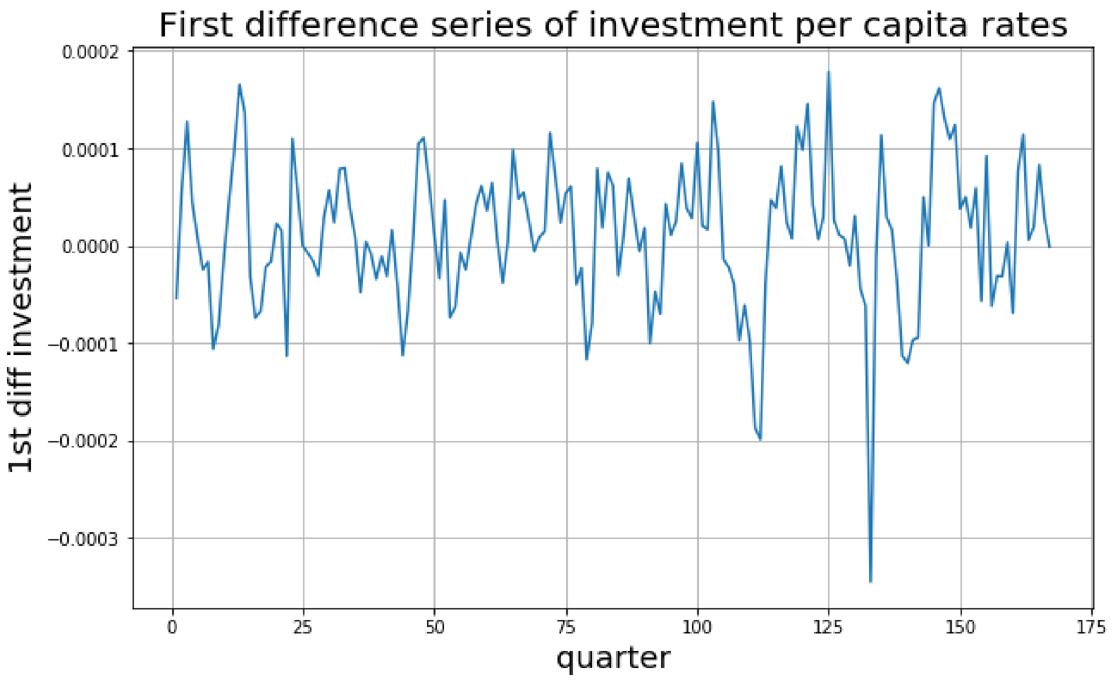
```
(n, moving_aves) = moving_average(resInvestments, N)

mean = np.mean(resInvestments)
dev = np.std(resInvestments)

# Plot moving average
plt.figure(figsize=[10,6])
plt.plot(n,moving_aves)
plt.hlines(mean,n[0],n[-1],linestyles='dashdot',label='mean')
plt.hlines(mean+dev,n[0],n[-1],linestyles='dashdot', label='mean+std', colors='r')
plt.hlines(mean-dev,n[0],n[-1],linestyles='dashdot', label='mean-std', colors='b')
plt.xlabel('sliding window', fontsize = 18)
plt.ylabel('average', fontsize = 18)
plt.grid()
plt.legend()
plt.title('Moving averages with sliding windows of length = %i' %N, fontsize = 20)
```

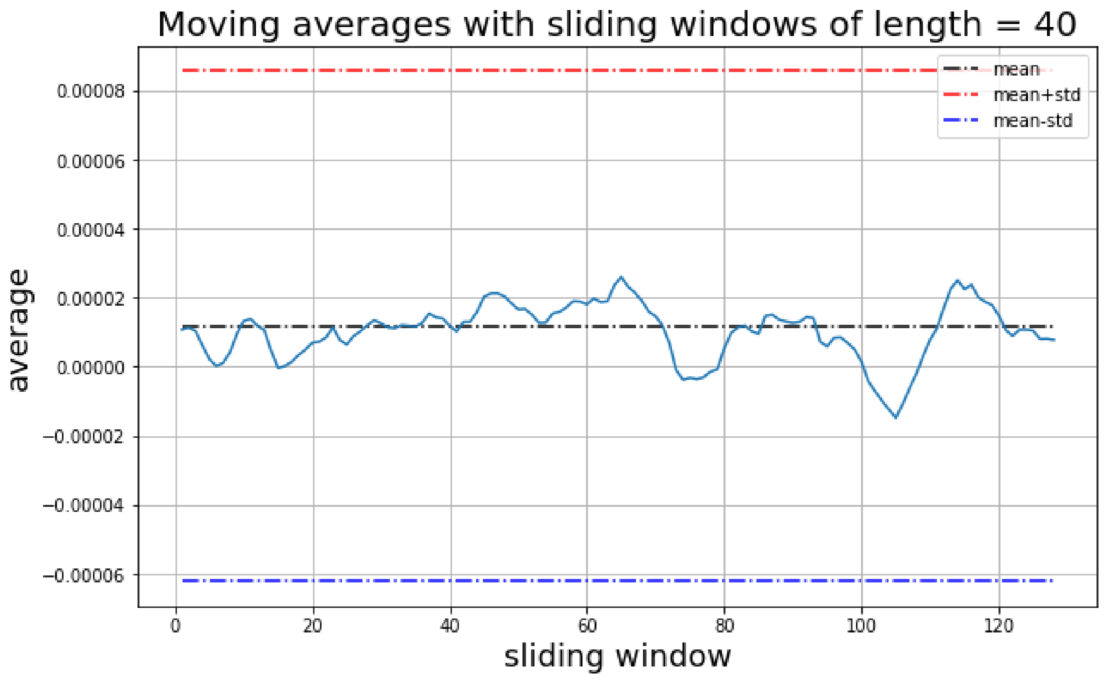
```
Out[9]: Text(0.5, 1.0, 'Moving averages with sliding windows of length = 40')
```



```
In [10]: # First differences series
         first_x = np.delete(quarters,0)
         first_diff_inv = np.diff(y)

         # Plot
         plt.figure(figsize=[10,6])
         plt.plot(first_x,first_diff_inv)
         plt.xlabel('quarter', fontsize = 18)
         plt.ylabel('1st diff investment', fontsize = 18)
         plt.grid()
         plt.title('First difference series of investment per capita rates', fontsize = 20)

         # Trend removed moving average
         N = 40
         (n, moving_aves) = moving_average(first_diff_inv, N)

         mean = np.mean(first_diff_inv)
         dev = np.std(first_diff_inv)

         # Plot moving average
         plt.figure(figsize=[10,6])
         plt.plot(n,moving_aves)
         plt.hlines(mean,n[0],n[-1],linestyles='dashdot',label='mean')
         plt.hlines(mean+dev,n[0],n[-1],linestyles='dashdot', label='mean+std', colors='r')
         plt.hlines(mean-dev,n[0],n[-1],linestyles='dashdot', label='mean-std', colors='b')
         plt.xlabel('sliding window', fontsize = 18)
         plt.ylabel('average', fontsize = 18)
         plt.grid()
         plt.legend()
         plt.title('Moving averages with sliding windows of length = %i' %N, fontsize = 20)
```

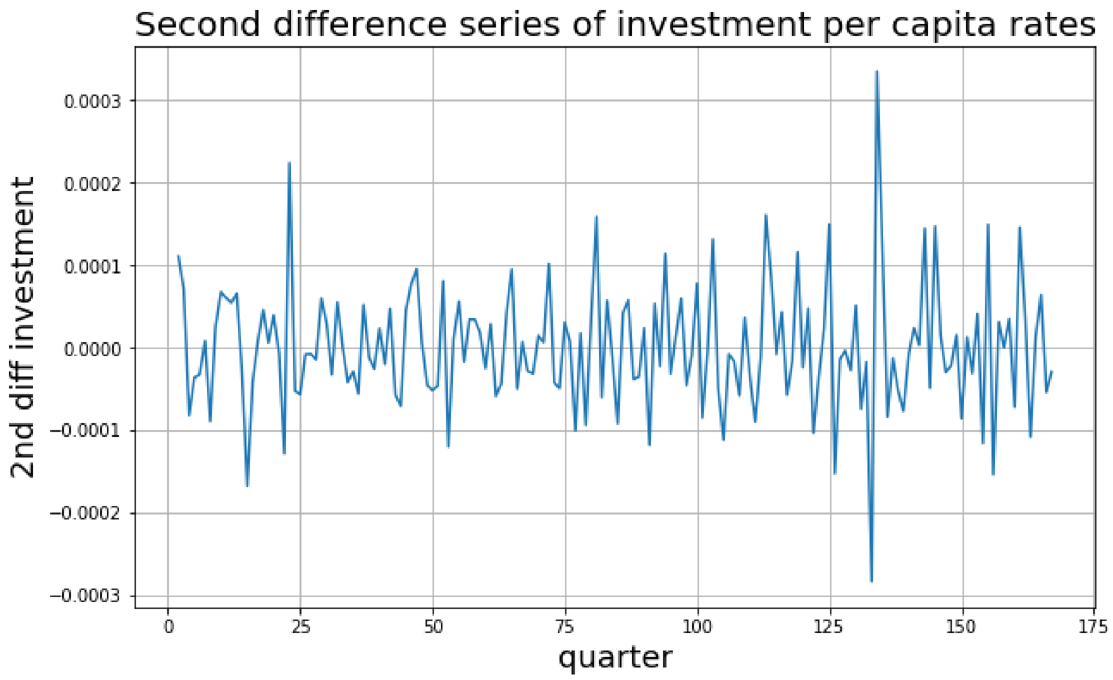
```
Out[10]: Text(0.5, 1.0, 'Moving averages with sliding windows of length = 40')
```

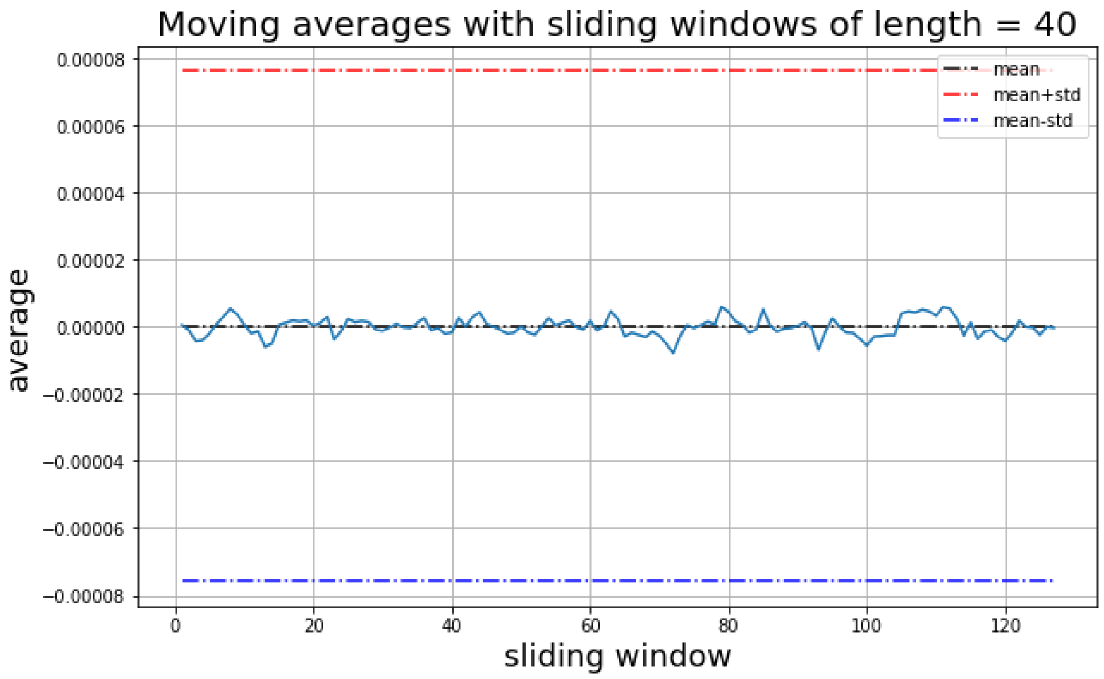




```
In [11]: # Second order differences series
         second_x = np.delete(first_x,0)
         second_diff_inv = np.diff(y, n=2)

         # Plot
         plt.figure(figsize=[10,6])
         plt.plot(second_x,second_diff_inv)
         plt.xlabel('quarter', fontsize = 18)
         plt.ylabel('2nd diff investment', fontsize = 18)
         plt.grid()
         plt.title('Second difference series of investment per capita rates', fontsize = 20)

         # Trend removed moving average
         N = 40
         (n, moving_aves) = moving_average(second_diff_inv, N)

         mean = np.mean(second_diff_inv)
         dev = np.std(second_diff_inv)

         # Plot moving average
         plt.figure(figsize=[10,6])
         plt.plot(n,moving_aves)
         plt.hlines(mean,n[0],n[-1],linestyles='dashdot',label='mean')
         plt.hlines(mean+dev,n[0],n[-1],linestyles='dashdot', label='mean+std', colors='r')
         plt.hlines(mean-dev,n[0],n[-1],linestyles='dashdot', label='mean-std', colors='b')
         plt.xlabel('sliding window', fontsize = 18)
         plt.ylabel('average', fontsize = 18)
         plt.grid()
         plt.legend()
         plt.title('Moving averages with sliding windows of length = %i' %N, fontsize = 20)
```

```
Out[11]: Text(0.5, 1.0, 'Moving averages with sliding windows of length = 40')
```





By looking at the plots of the moving-average of the three series (without linear trend, first differences, second differences), it can be noticed that the all the obtained values are close to the average of the whole sample (contained in the interval $[\overline{x} - \sigma, \overline{x} + \sigma]$) and there aren't systematic trend across time: this shows that the average is almost constant in time and it's a reason why the series can be considered stationary.

3.0.3 c) Compute the autocorrelation function of the time series with the linear trend removed. Are business cycles (corresponding to peaks in the autocorrelation function) periodic?

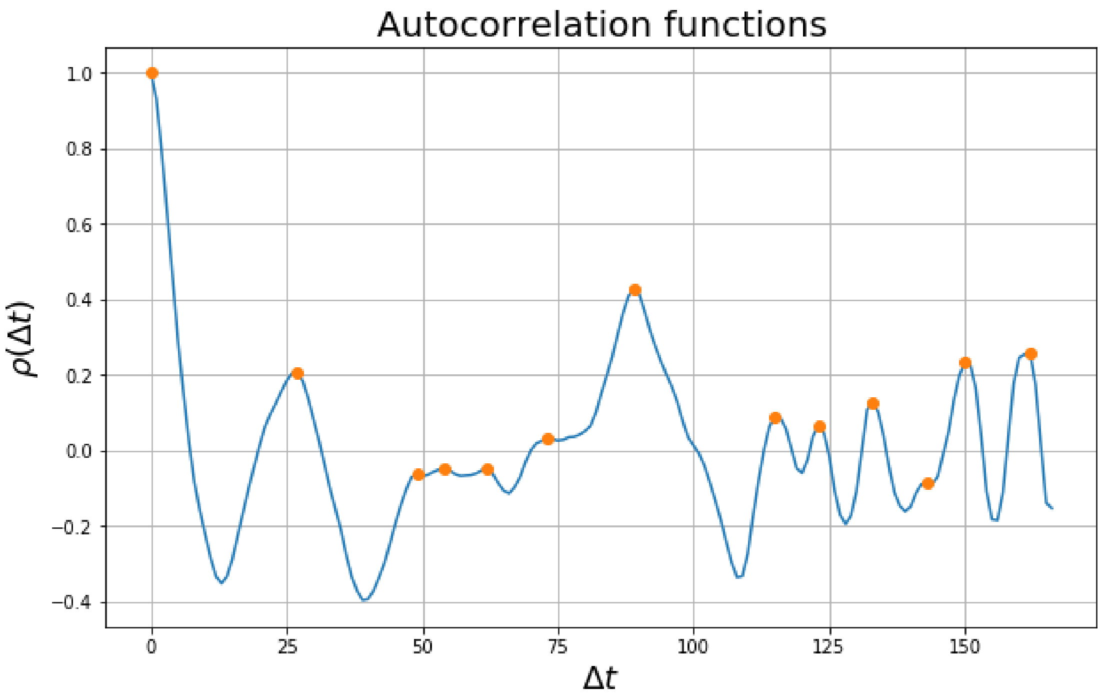
The time series is assumed to be stationary and ergodic, so the empirical estimation of the auto-covariance and auto-correlation are computed as: $acov(\Delta t) = \gamma(\Delta t) = \frac{1}{T - \Delta t} \sum_{i=1}^{T-\Delta t} (x_t - \overline{x})(x_{t+\Delta t} - \overline{x})$ $acorr(\Delta t) = \rho(\Delta t) = \frac{\gamma(\Delta t)}{\gamma(0)}$

```
In [12]: def acov(x, dt):
             T = len(x)
             x_mean = np.mean(x)
             x = x - x_mean

             cov = 0

             # maybe try convolution function
             for i in range(T-dt):
                 cov += x[i]*x[i+dt]
```

```
            return cov / (T-dt)


        def acorr(x, dt):
            return acov(x,dt) / acov(x,0)
```

```
In [13]: # Compute autocorrelation function
         corr_inv = np.zeros(len(quarters)-1)

         for i in range(len(quarters)-1):
             corr_inv[i] = acorr(resInvestments, quarters[i])

         # Find local maxima
         isMax = np.r_[True, corr_inv[1:] > corr_inv[:-1]] & np.r_[corr_inv[:-1] > corr_inv[1:],

         # Plot
         plt.figure(figsize=[10,6])
         plt.plot(quarters[:-1],corr_inv)
         #plt.plot(quarters[:-1],corr_inv,'.')
         plt.plot(quarters[:-1][isMax],corr_inv[isMax],'o')
         plt.xlabel('$\Delta t$', fontsize = 18)
         plt.ylabel(r'$\rho (\Delta t)$', fontsize = 18)
         plt.grid()
         plt.title('Autocorrelation functions', fontsize = 20)

         # Time btw two peaks
         print('Time btw two peaks:')
         print(np.diff(quarters[:-1][isMax]))
         print('Average time btw two peaks:')
         print(np.mean(np.diff(quarters[:-1][isMax])))
```

```
Time btw two peaks:
[27 22  5  8 11 16 26  8 10 10  7 12]
Average time btw two peaks:
13.5
```



The autocorrelation functions has many peaks, that correspond to business cycles. As it can be noticed above, the values of time between two peaks is very variable: this means that the business cycles are not periodic.

4 Task 3

4.0.1 1) Create your own AR time series of length T = 200 and order p = 4, with the following coefficients given:

4.0.2 a0 = 0.0, a1 = -0.8, a2 = 0.0, a3 = 0.0, and a4 = 0.4, with $\epsilon_t \sim$ N(0,1), and with the initial value of the time series being x_0 = 0.0

AR series: $x_t = a_0 + \sum_{i=1}^{p} a_i x_{t-1} + \epsilon_t$

```
In [14]: # Parameters
         T = 200
         p = 4

         # Coefficients
         a0 = 0.0
         a1 = -0.8
         a2 = 0.0
         a3 = 0.0
         a4 = 0.4

         a = np.array([a1, a2, a3, a4])
         # Initial value
         x0 = 0.0
```

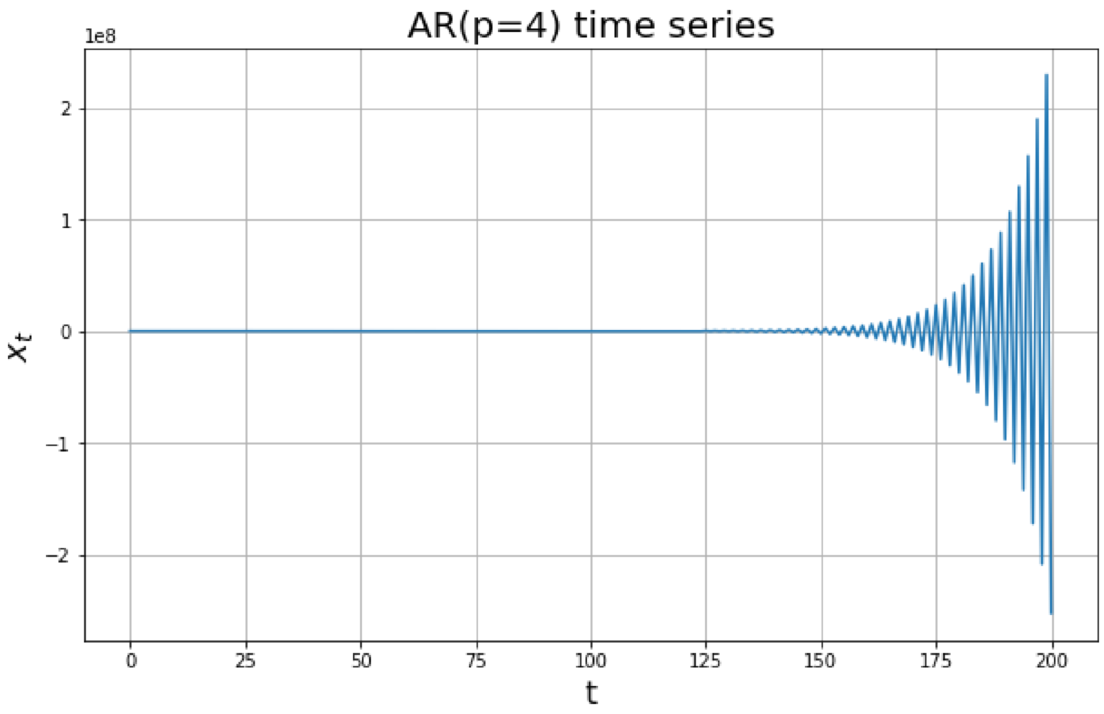
```
In [15]: # Noise
         seed = 30091996
         np.random.seed(seed)
         epsilon = np.random.normal(size=T)
```

```
In [16]: # Generate AR series
         x = np.zeros(T+1)
         x[0] = x0          # setting initial value

         for t in range(1,T+1):
             temp = 0
             for i in range(0,p):
                 if(t-i-1 > 0):
                     temp += a[i]*x[t-i-1]
             x[t] = a0 + temp + epsilon[t-1]
```

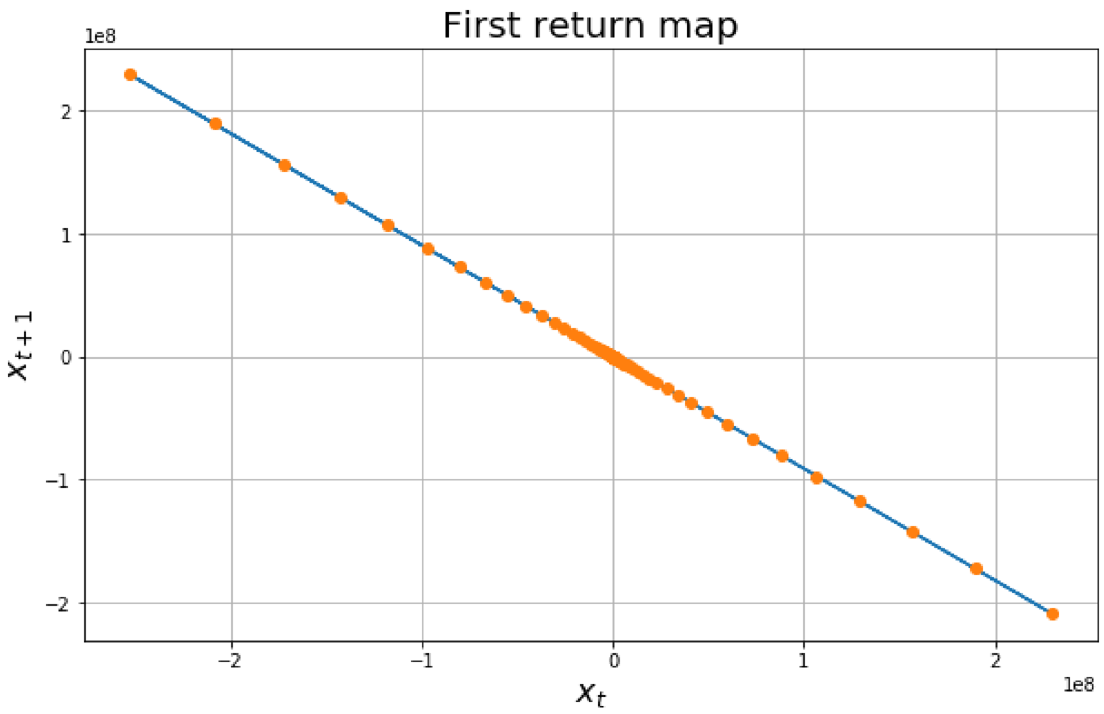
4.0.3 2) Plot the time series in time as well as the first return-map. What do you notice?

```
In [17]: # Plot the time series
         plt.figure(figsize=[10,6])
         plt.plot(np.arange(0,T+1),x)
         #plt.plot(quarters[:-1],corr_inv,'.')
         plt.xlabel('t', fontsize = 18)
         plt.ylabel(r'$x_t$', fontsize = 18)
         plt.grid()
         plt.title('AR(p=%i) time series' %p, fontsize = 20)
```

```
Out[17]: Text(0.5, 1.0, 'AR(p=4) time series')
```



```
In [18]: # Return map
         xt = np.delete(x, 0)
         xt1 = np.delete(x, len(x)-1)

         # Plot the return map
         plt.figure(figsize=[10,6])
         plt.plot(xt, xt1)
         plt.plot(xt, xt1,'o')
         plt.xlabel('$x_t$', fontsize = 18)
         plt.ylabel('$x_{t+1}$', fontsize = 18)
         plt.grid()
         plt.title('First return map', fontsize = 20)

Out[18]: Text(0.5, 1.0, 'First return map')
```



It can be noticed that the trend of the time series seems to be an exponential with values that continuously jump from negative to positive and the other way around. This is reflected in the plot of the first return map, where the points are distributed along a line with negative slop.

5 Task 4

5.0.1 We have derived that the criterion for an AR(1) process, $x_t = a_0 + a_1 x_{t-1} + \varepsilon_t$, $\epsilon_t \sim W(0, \sigma^2)$, to be stationary is |a1|<1. Show that for an AR(p) process, p>1, the mean of the time series is stationary if |ai|<1, by expanding the process in time

By expanding the process in time I obtain:
$x_t = a_0 + \sum_{i=1}^{p} a_i x_{t-i} + \epsilon_t$
$= a_0 + \sum_{i=1}^{p} a_i \{a_0 + \sum_{j=1}^{p} a_j x_{t-i-j} + \epsilon_{t-1}\} + \epsilon_t$
$= a_0 + \sum_{i=1}^{p} a_i \{a_0 + \sum_{j=1}^{p} a_j [a_0 + \sum_{k=1}^{p} a_k x_{t-i-j-k} + \epsilon_{t-2}] + \epsilon_{t-1}\} + \epsilon_t$

By calculating the mean value, all the term linear in ϵ_t vanishes, because of the definition of white noise ($E[\epsilon_t] = \overline{\epsilon}_t = 0 \; \forall t$), so:
$\overline{x}_t = a_0 + \sum_{i=1}^{p} a_i \overline{x}_{t-i} + \overline{\epsilon}_t = a_0 + \sum_{i=1}^{p} a_i \overline{x}_{t-i}$
$= a_0 + \sum_{i=1}^{p} a_i \{a_0 + \sum_{j=1}^{p} a_j \overline{x}_{t-i-j}\}$
$= a_0 + \sum_{i=1}^{p} a_i \{a_0 + \sum_{j=1}^{p} a_j [a_0 + \sum_{k=1}^{p} a_k \overline{x}_{t-i-j-k}]\}$ <br/> $= a_0 + a_0 \sum_{i=1}^{p} a_i + a_0 \sum_{i=1}^{p} a_i \sum_{j=1}^{p} a_j + \sum_{i=1}^{p} a_i \sum_{j=1}^{p} a_j \sum_{k=1}^{p} a_k (a_0 + ...)$
$= a_0 * [1 + \sum_{i=1}^{p} a_i + (\sum_{i=1}^{p} a_i)^2 + ... + (\sum_{i=1}^{p} a_i)^{T-p} + ...]$

The first (T-p) terms in the summation can be rewritten as: $a_0 \sum_{n=0}^{T-p} (\sum_{i=1}^{p} a_i)^n$ while the other term depends on the first (finite) p values of the AR process.

By taking the limit for $T \rightarrow 0$ and using the property of the geometric series the expectation value becomes:
$\overline{x}_t = a_0 \sum_{n=0}^{\infty} (\sum_{i=1}^{p} a_i)^n = \dfrac{a_0}{1 - \sum_{i=1}^{p} a_i}$ if $| \sum_{i=1}^{p} a_i| < 1$