timeseries_q1

December 16, 2018

1 Time Series Problem Set: Question 1

```
In [1]: import numpy as np
        import pandas as pd
        from statsmodels.tsa.stattools import levinson_durbin
        from scipy.stats import kurtosis, kstat
        from scipy.special import comb
```

```
In [2]: # Read in data
        YEAR_NUMBER = 2000
        df = pd.read_csv(f'../portfolio-analysis/{YEAR_NUMBER}_data.csv', index_col=0)

        # Cut data to 250 days
        sp500 = df.SP500[:250]
        assert len(sp500) == 250
```

1.1 Part (a)

```
In [3]: M = 10
        lagged = np.vstack([sp500[i:240+i] for i in range(M + 1)]).T
        cov = np.cov(lagged.T)
        eigvals = np.linalg.eigvalsh(cov)  # Eigenvalues of a symmetric matrix
        msg = 'R is positive definite.' if (eigvals > 0).all() else 'R is NOT positive definit
        print(msg)
```

```
R is positive definite.
```

1.2 Part (b)

If all reflection coefficients k_m had magnitude less than 1, then the corresponding polynomial is stable. If all reflection coefficients above a certain order are all 0, then the corresponding system is exactly AR.

```
In [4]: # The Levinson-Durbin and least squares coefficients generally agree,
        # except for the first coefficient, which represents \delta_{p0}.

        _, a_lv, _, sigma, _ = levinson_durbin(s=sp500, nlags=10)
```

```
ar_coeff_lv = np.hstack([1, a_lv])
ar_coeff_ls, _, _, _ = np.linalg.lstsq(np.hstack([np.ones([lagged.shape[0], 1]), lagged
                                       lagged[:, 0], rcond=None)

print(f'Levinson-Durbin:\n{ar_coeff_lv}\n')
print(f'Least Squares:\n{ar_coeff_ls}')
```

```
Levinson-Durbin:
[ 1.         -0.02281734 -0.13396212 -0.08700516  0.05224546 -0.02211634
 -0.12596476 -0.05057278 -0.15646007 -0.06893532 -0.06077776]

Least Squares:
[-0.0004077  -0.04056862 -0.12891734 -0.10317117  0.0520129  -0.01242916
 -0.12603856 -0.04806614 -0.16098401 -0.07595534 -0.0628058 ]
```

1.3 Part (c)

In [5]:
```
aic = (2/250)*np.log(sigma[1:]) + [2*i/250 for i in range(10)]
optimal_lag = np.argmin(aic) + 1
print(f'Optimal lag value: {optimal_lag}')
```

```
Optimal lag value: 1
```

1.4 Part (d)

In [6]:
```
sp500_diff = sp500.diff()[1:]
```

In [7]:
```
# Part (a)
M = 10
lagged = np.vstack([sp500_diff[i:240+i] for i in range(M)]).T
cov = np.cov(lagged.T)
eigvals = np.linalg.eigvalsh(cov)  # Eigenvalues of a symmetric matrix
msg = 'R is positive definite.' if (eigvals > 0).all() else 'R is NOT positive definite
print(msg)

print(20*'-')

# Part (b)
_, a_lv, _, sigma, _ = levinson_durbin(s=sp500_diff, nlags=10)
ar_coeff_lv = np.hstack([1, a_lv])
ar_coeff_ls, _, _, _ = np.linalg.lstsq(np.hstack([np.ones([lagged.shape[0], 1]), lagged
                                       lagged[:, 0], rcond=None)

print(f'Levinson-Durbin:\n{ar_coeff_lv}\n')
print(f'Least Squares:\n{ar_coeff_ls}')

print(20*'-')
```

```
# Part (c)
aic = (2/250)*np.log(sigma[1:]) + [2*i/250 for i in range(10)]
optimal_lag = np.argmin(aic) + 1
print(f'Optimal lag value: {optimal_lag}')
```

```
R is positive definite.
--------------------
Levinson-Durbin:
[ 1.         -0.85819164 -0.82982115 -0.73263829 -0.51955483 -0.39889518
 -0.40963004 -0.34030536 -0.35560864 -0.26722563 -0.18607999]

Least Squares:
[-3.66174262e-05 -8.53382533e-01 -8.01809585e-01 -7.29542355e-01
 -4.97469393e-01 -3.66518338e-01 -3.55344234e-01 -2.33551819e-01
 -2.25345080e-01 -1.23021851e-01]
--------------------
Optimal lag value: 1
```

1.5 Part (e)

In [8]:
```
# For the direct model, M = 1

M = 1
lagged = np.vstack([sp500[i:250-M-1+i] for i in range(M + 1)]).T
x = lagged[:, 1:]
y = lagged[:, 0]
_, ar_coeff, _, sigma, _ = levinson_durbin(s=sp500_diff, nlags=M)
resid = pd.Series(y - x @ ar_coeff)

reflection_coeff = ar_coeff[-1]
cov = np.array([resid.autocorr(lag=i) for i in range(1, 11)])

print(f'Reflection coefficient: {reflection_coeff}')
print(f'Covariance coefficients: {cov}')
```

```
Reflection coefficient: -0.43893010076607053
Covariance coefficients: [ 0.31902029 -0.13973842 -0.07165218  0.06467166  0.00120351 -0.12638
 -0.12899869 -0.14733625 -0.1000778  -0.03963263]
```

In [9]:
```
# For the direct model, M = 10

M = 10
lagged = np.vstack([sp500[i:250-M-1+i] for i in range(M + 1)]).T
x = lagged[:, 1:]
y = lagged[:, 0]
_, ar_coeff, _, sigma, _ = levinson_durbin(s=sp500_diff, nlags=M)
```

```
        resid = pd.Series(y - x @ ar_coeff)

        reflection_coeff = ar_coeff[-1]
        cov = np.array([resid.autocorr(lag=i) for i in range(1, 11)])

        print(f'Reflection coefficient: {reflection_coeff}')
        print(f'Covariance coefficients: {cov}')
```

```
Reflection coefficient: -0.18607999182760335
Covariance coefficients: [ 0.7915239   0.61016588  0.44747918  0.32905856  0.22430906  0.122245
  0.03448174 -0.04191936 -0.08132558 -0.08543381]
```

```
In [10]: # For the first difference model, M = 1

         M = 1
         lagged = np.vstack([sp500_diff[i:250-M-1+i] for i in range(M + 1)]).T
         x = lagged[:, 1:]
         y = lagged[:, 0]
         _, ar_coeff, _, sigma, _ = levinson_durbin(s=sp500_diff, nlags=M)
         resid = pd.Series(y - x @ ar_coeff)

         reflection_coeff = ar_coeff[-1]
         cov = np.array([resid.autocorr(lag=i) for i in range(1, 11)])

         print(f'Reflection coefficient: {reflection_coeff}')
         print(f'Covariance coefficients: {cov}')
```

```
Reflection coefficient: -0.43893010076607053
Covariance coefficients: [-0.15351775 -0.39415528 -0.05181163  0.15617874  0.05112625 -0.099354
  0.00612662 -0.05025972 -0.01387615 -0.05110238]
```

```
In [11]: # For the first difference model, M = 10

         M = 10
         lagged = np.vstack([sp500_diff[i:250-M-1+i] for i in range(M + 1)]).T
         x = lagged[:, 1:]
         y = lagged[:, 0]
         _, ar_coeff, _, sigma, _ = levinson_durbin(s=sp500_diff, nlags=M)
         resid = pd.Series(y - x @ ar_coeff)

         reflection_coeff = ar_coeff[-1]
         cov = np.array([resid.autocorr(lag=i) for i in range(1, 11)])

         print(f'Reflection coefficient: {reflection_coeff}')
         print(f'Covariance coefficients: {cov}')
```

```
Reflection coefficient: -0.18607999182760335
Covariance coefficients: [-0.06918262 -0.04577444 -0.05501915 -0.03831847 -0.0117778  -0.06144
```

```
 -0.03576841 -0.09111944 -0.08440679 -0.05051641]
```

1.6 Part (f)

```
In [12]: # Taken from https://www.statsmodels.org/dev/_modules/statsmodels/stats/moment_helper
         def mnc2cum(mnc):
             '''convert non-central moments to cumulants
             recursive formula produces as many cumulants as moments

             http://en.wikipedia.org/wiki/Cumulant#Cumulants_and_moments
             '''
             mnc = [1] + list(mnc)
             kappa = [1]
             for nn,m in enumerate(mnc[1:]):
                 n = nn+1
                 kappa.append(m)
                 for k in range(1,n):
                     kappa[n] -= comb(n-1,k-1,exact=1) * kappa[k]*mnc[n-k]

             return kappa[1:]
```

```
In [13]: # Kurtosis is not close to 3, which would be expected for a Gaussian variable.
         # It looks like the residuals are not Gaussian!
         kurt = kurtosis(resid)
         non_central_moments = [np.mean(resid**k) for k in range(3, 7)]
         cumul = mnc2cum(non_central_moments)

         print(f'Kurtosis: {kurt}')
         print(f'Cumulants: {cumul}')
```

```
Kurtosis: 1.3075291165509402
Cumulants: [2.4711531716496534e-07, 1.8642644171130766e-07, 1.7091544675485145e-09, 3.45968670
```