Тимофей Прокопенко, лабораторная 1

№ 8.1

Используя генераторы псевдослучайных чисел, смоделировать временные ряды, порождаемые каузальными и обратимыми процессами АРСС, длиной N. Изобразить графически значения временных рядов.

Задание 1. Для временного ряда $x_1, \ldots, x_N$, порождаемого уравнением АР (p)

$$x_t + a_1 x_{t-1} + \ldots + a_p x_{t-p} = W_t, \quad t = p+1, \ldots, N,$$

где $\{W_t\}$ — последовательность независимых гауссовских СВ с нулевым средним и дисперсией σ^2, начальные значения $x_1 = \ldots = x_p = 0$, оценить ковариационную функцию

$$\gamma_s = \frac{1}{N-s} \sum_{t=s+1}^{N} x_t x_{t-s}, \quad s = 0, 1, \ldots, N-1.$$

С помощью алгоритмов 1–3 вычислить ковариационную функцию γ_s. Используя формулу (3.10), определить частную ковариационную функцию $\alpha(n)$, $n = 1, 2, \ldots, p+k$. Применяя формулу (3.10), в которой истинные значения ρ_n заменены на их оценки $\hat{\rho}_n = \hat{\gamma}_n / \hat{\gamma}_0$, оценить частную ковариационную функцию $\hat{\alpha}(n)$, $n = 1, 2, \ldots, p+k$. Изобразить графически ковариационную и частную ковариационную функции.

Решение:

1. Для начала выберем параметры процесса. Пусть p=1, N=100.

   Стационарность авторегрессионного процесса зависит от корней характеристического полинома $a(z) = 1 - \sum_{i=1}^{n} a_i z^i$. Для того чтобы процесс был стационарным[1], достаточно, чтобы *все корни характеристичекого полинома лежали вне единичного круга в комплексной плоскости* $|z| > 1$.
   В частности, для AR(1)-процесса $a(z) = 1 - rz$, следовательно корень этого полинома $z = 1/r$, поэтому условие стационарности можно записать в виде $|r| < 1$, то есть коэффициент авторегрессии (он же в данном случае коэффициент автокорреляции) должен быть строго меньше 1 по модулю.

   Таким образом, можно взять коэффицент a=0.99. Приведем часть кода Python с нужными импортами и заданием переменных:

In [47]:

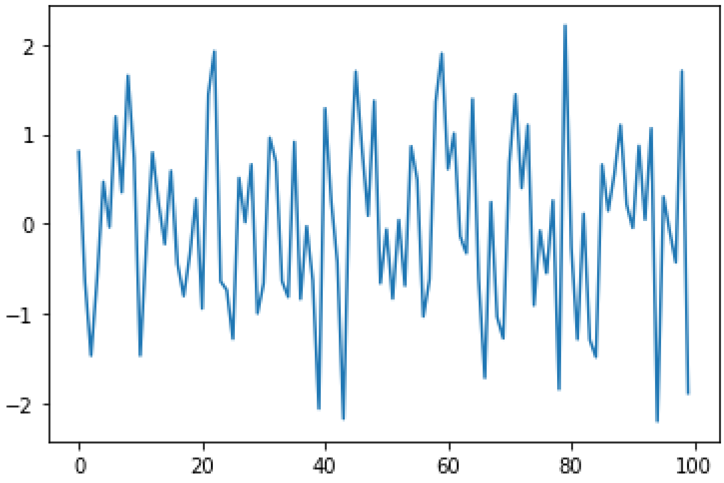
```
from random import random
from statistics import mean
import numpy
import matplotlib.pyplot as plt

N = 100
p = 1
a = 0.99
```

1. Моделирование процесса белого шума:

In [48]:

```
white_noise = numpy.random.normal(0, 1, size=N)  # N elements in the white_noise
# print(white_noise)
plt.plot(white_noise)
plt.show()
```



1. Моделирование данных

In [49]:

```
data = [0]
for i in range(1, N):  # N elements in the data
    data.append(white_noise[i] - a*data[i-1])
#print(data)
```

1. Вычисление ковариационной функции. Выберем второй алгоритм, учитывая, что:

$$\psi_0 = b_0 = 1,$$

Алгоритм 2. Основан на разностных уравнениях для γ_s, $s = 0, 1, \ldots,$ которые получаются умножением каждого слагаемого в (3.1) на x_{t-s} и использованием операции математического ожидания:

$$\gamma_s + a_1\gamma_{s-1} + \ldots + a_p\gamma_{s-p} = \sigma^2 \sum_{k=s}^{q} b_k\psi_{k-s} \quad \text{для } 0 \leq s < \max(p, q+1). \qquad (3.6)$$

$$\gamma_s + a_1\gamma_{s-1} + \ldots + a_p\gamma_{s-p} = 0 \quad \text{для } s \geq \max(p, q+1). \qquad (3.7)$$

При p=1 формула 3.6 породит только один элемент равный σ^2. Остальные элементы будут порождены формулой 3.7. Пусть 0 < s < N/3.

In [50]:

```
s_max = N#int(N/3)
sigma_2 = mean([x*x for x in white_noise])
covariance = [sigma_2]
for i in range(1, s_max):  # N/3 elements in the covariance vector
    covariance.append(-a*covariance[i-1])
#print(covariance)
```

1. Оценки ковариационной функции. По формуле:

$$\gamma_s = \frac{1}{N-s}\sum_{t=s+1}^{N} x_t x_{t-s}, \quad s = 0, 1, \ldots, N-1.$$

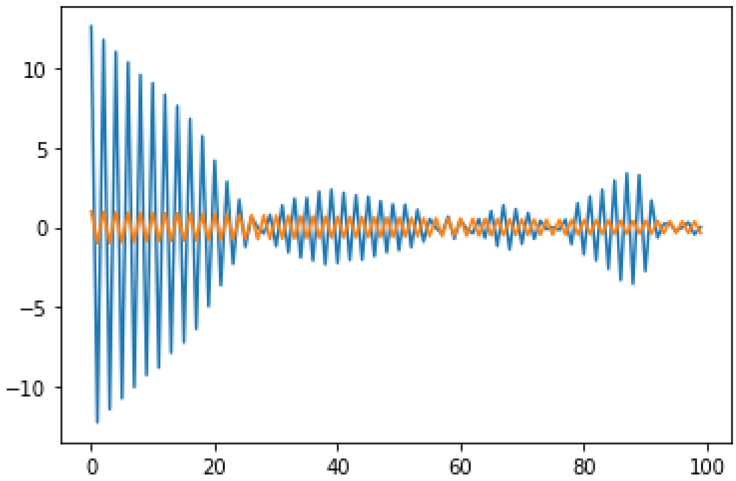
После изобразим график ковариационной функции (оранжевым цветом) и ее оценку (синим цветом).

In [70]:

```
covariance_est = []
for s in range(0, s_max):
    local_sum = sum([data[t]*data[t-s] for t in range(s+1, N)])
    covariance_est.append((1/(N-s))*local_sum)
#print(covariance_est)
plt.plot(covariance_est)
plt.plot(covariance)
```

Out[70]:

```
[<matplotlib.lines.Line2D at 0x15c38d28cc0>]
```



1. Частная ковариационная функция. Она будет найдена по следующим формулам:

$$\begin{pmatrix} \rho_0 & \rho_1 & \dots & \rho_{n-1} \\ \rho_1 & \rho_0 & \dots & \rho_{n-2} \\ \vdots & \vdots & \dots & \vdots \\ \rho_{n-1} & \rho_{r-2} & \dots & \rho_0 \end{pmatrix} \begin{pmatrix} a_{n1} \\ a_{n2} \\ \vdots \\ a_{nn} \end{pmatrix} = \begin{pmatrix} \rho_1 \\ \rho_2 \\ \dots \\ \rho_n \end{pmatrix}, \quad n \geq 1, \qquad (3.10)$$

где $\rho_j = \gamma_j/\gamma_0$. Частная ковариационная функция в этом случае записывается в виде $\alpha(n) = a_{nn}$, $n \geq 1$, где a_{nn} единственным образом находятся из (3.10).

In [57]:

```
def get_partial_covariance(y_s, n):
    ro_vector = numpy.array([y_s[i]/y_s[0] for i in range(0, n+1)])  # n+1 elements
    matrix = numpy.array([[ro_vector[abs(j-i)] for i in range(0, n)] for j in range(0, n)])
    answer_vector = numpy.linalg.solve(matrix, ro_vector[1:])
    return answer_vector[n-1]

partial_covariance = [get_partial_covariance(covariance, n) for n in range(1, N-1)]
#print(partial_covariance)
```

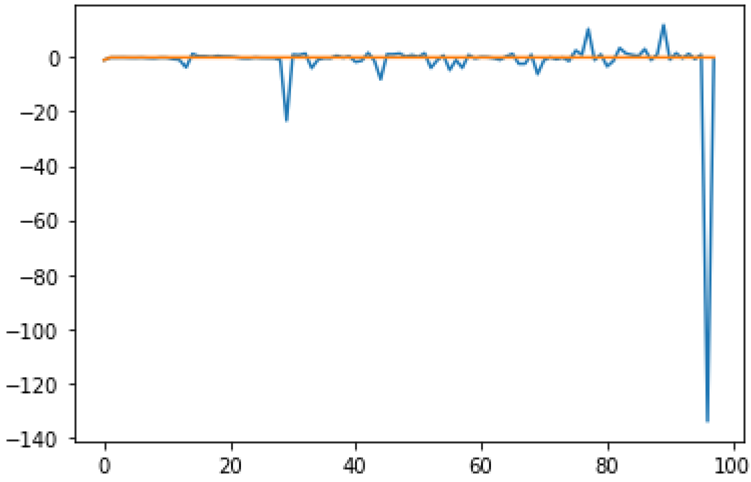
1. Оценка частной ковариационной функции. Изобразим также график частной ковариацинной функции (оранжевым цветом) и ее оценку (синим цветом).

In [71]:

```
partial_covariance_est = [get_partial_covariance(covariance_est, n) for n in range(1, N
-1)]
#print(partial_covariance_est)
plt.plot(partial_covariance_est)
plt.plot(partial_covariance)
```

Out[71]:

```
[<matplotlib.lines.Line2D at 0x15c37a869e8>]
```



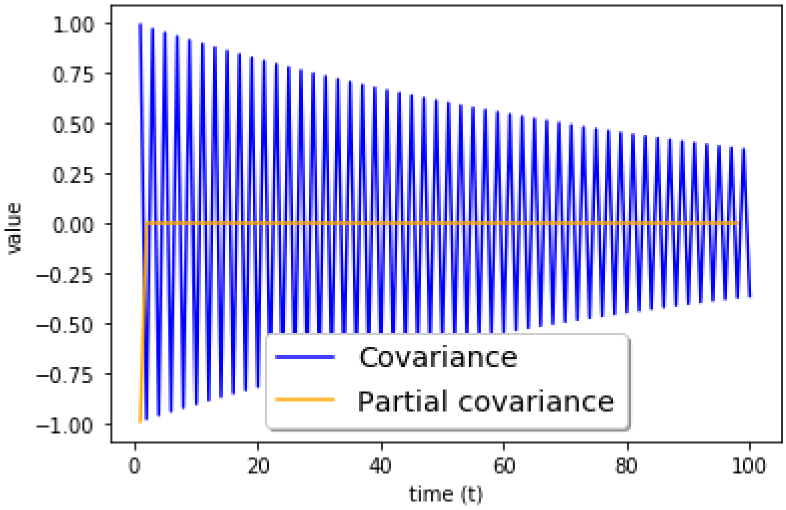
1. Графики ковариационной и частной ковариационной функции.

In [72]:

```
def generate_x_axe(y):
    return [x+1 for x in range(0, len(y))]

fig, ax = plt.subplots()
ax.plot(generate_x_axe(covariance), covariance, label='Covariance', color='blue')
ax.set_xlabel('time (t)')
ax.set_ylabel('value')

ax.plot(generate_x_axe(partial_covariance), partial_covariance, label='Partial covarian
ce', color='orange')
legend = ax.legend(loc='lower center', shadow=True, fontsize='x-large')
plt.show()
```



Приложение А. Листинг программы.

In [ ]:

```
from statistics import mean
import numpy
import matplotlib.pyplot as plt


def get_partial_covariance(y_s, n):
    ro_vector = numpy.array([y_s[i]/y_s[0] for i in range(0, n+1)])  # n+1 elements
    matrix = numpy.array([[ro_vector[abs(j-i)] for i in range(0, n)] for j in range(0, 
n)])
    answer_vector = numpy.linalg.solve(matrix, ro_vector[1:])
    return answer_vector[n-1]


def generate_x_axe(y):
    return [x+1 for x in range(0, len(y))]


if __name__ == '__main__':
    N = 100
    p = 1
    a = 0.99
    white_noise = numpy.random.normal(0, 1, size=100)  # N elements in the white_noise
    data = [0]
    for i in range(1, N):  # N elements in the data
        data.append(white_noise[i] - a*data[i-1])
    # print(data)
    s_max = N #int(N/3)
    sigma_2 = mean([x*x for x in white_noise])
    covariance = [sigma_2]
    for i in range(1, s_max):  # N/3 elements in the covariance vector
        covariance.append(-a*covariance[i-1])
    #print(covariance)

    covariance_est = []
    for s in range(0, s_max):
        covariance_est.append((1/(N-s))*(sum([data[t]*data[t-s] for t in range(s+1, N
)])))
    #print(covariance_est)

    partial_covariance = [get_partial_covariance(covariance, n) for n in range(1, N-1)]
    # print(partial_covariance)
    partial_covariance_est = [get_partial_covariance(covariance_est, n) for n in range(
1, N-1)]
    # print(partial_covariance_est)

    fig, ax = plt.subplots()
    ax.plot(generate_x_axe(covariance), covariance, label='Covariance', color='blue')
    ax.set_xlabel('time (t)')
    ax.set_ylabel('value')

    ax.plot(generate_x_axe(partial_covariance), partial_covariance, label='Partial cova
riance', color='orange')
    legend = ax.legend(loc='lower center', shadow=True, fontsize='x-large')

    plt.show()
```

In [ ]:

```

```