§45. Лабораторная работа № 9
Оптимизация процедуры опроса векторного случайного процесса

Ц е л ь р а б о т ы: изучить процедуру опроса векторного случайного процесса

С о д е р ж а н и е р а б о т ы: построить программный имитатор векторного случайного процесса.

Пример выполнения здания

Сымитировать 6-мерный векторный случайный процесс, корреляционную функцию для этого процесса. Реализовать программно алгоритм, описанный в главе 8 в параграфе 39 “Задача об оптимизации опроса векторного случайного процесса.”

Программный код (язык программирования Python):

```
import random
import itertools
import math
def print_flist(s):
print("[ ", end="")
for i in range(len(s)):
print(f"{s[i]:.20f}; ", end="")
if (i + 1) % 11 == 0:
print()
print("]")
def white_noize():
s = 0
for i in range(12):
s += random.random()
return s - 6
def ud_4_normal_dist(M, D): """ Calculate coefficients
for normal distribution
M - math exception
```

```
D - dispersion"""
return M, D**0.5
def normal_dist_val(u, d):
return white_noize() * d + u
class RandomProcess:
def __init__(self, M: float, D: float, size = 1):
self.func_dist_val = normal_dist_val # функция
плотности
self.ud_4_func_dist = ud_4_normal_dist # функция
получения u,d коэффициентов распределения
self.__data = []
self.__pcoeffs = []
self.math_except = M # мат. ожидание
self.dispersion = D #
дисперсия
self.new_data(size)
def new_data(self, size: int)-> None: # получить
новые данные случайного процесса
if size <= 0 or type(size) != int:
raise ValueError("Bad size for process!!!")
self.__data =
[self.func_dist_val( #
компоненты случайного процесса
*self.ud_4_func_dist(self.math_except, self.dispersion))
for i in range(size)]
if len(self.__data) != len(self.__pcoeffs):
if size > 1:
self.__pcoeffs = RandomProcess.new_pcoeffs(size - 1)
# p-коэффициенты случайного процесса
else:
```

```
self.__pcoeffs = [1]
def size(self)-> int: # количество компонент в процессе
return len(self.__data)
def data(self)-> list:
return self.__data
def value(self): # получить значение случайного
процесса
z = 0
for i in range(len(self.__data)):
z += self.__data[i] * self.__pcoeffs[i]
return z
def __str__(self):
s = "z = "
for i in range(len(self.__data) - 1, -1, -1):
s += f"{abs(self.__data[i])}"
if i != len(self.__data) - 1:
s += f" * {abs(self.__pcoeffs[i])}"
if i != 0:
if self.__data[i - 1] * self.__pcoeffs[i - 1] > 0:
s += " + " else:
s += " - "
return s
@staticmethod
def new_pcoeffs(size): """ Функция аналогия coeffs_equation_by_vieta"""
if size < 1:
raise ValueError('Bad parameters!!!')
eq_roots = [random.randint(2, 100) * (1 if random.randint(0, 1) else -1) for i
in range(size)]
a = [0] * (len(eq_roots) + 1)
a[len(a) - 1] = 1
```

```
a[0] = -1 if size % 2 else 1
for root in eq_roots:
a[0] *= root
a[0] = a[len(a) - 1] / a[0]
sign = -1
for i in range(len(eq_roots) - 1):
inds = [e for e in range(i + 1)]
while inds[0] <= len(eq_roots) - len(inds):
p = 1
for ind in inds:
p *= eq_roots[ind]
a[i + 1] += p
inds[len(inds) - 1] += 1
for k in range(len(inds) - 1, 0, -1):
v = len(eq_roots) - len(inds) + k
if inds[k] > v:
inds[k - 1] += 1
for j in range(k - 1, len(inds) - 1):
inds[j + 1] = inds[j] + 1
a[i + 1] *= a[0] * sign
sign = -sign
return a
def correlation_func(self, t): #
корркеляционная функция
R = 0
for i in range(len(self.__pcoeffs) - t):
R += self.__pcoeffs[i] * self.__pcoeffs[i + t]
return R
def norm_correlation_func(self, t): #
нормированная корредяционная функция
```

```
return self.correlation_func(t) / self.correlation_func(0)
class RandomVectorProcess:
def __init__(self, M, D, size_vector = 1, size_proc = 1):
self.__data = []
self.__new_data(M, D, size_vector, size_proc)
def __new_data(self, M: float, D: float, size_vector: int, size_proc: int):
if size_vector <= 0 or size_proc <= 0:
raise ValueError("Bad size for vector process or vector process's
component!!!")
if len(self.__data) == 0:
self.__data = [RandomProcess(M, D, size_proc) for i in
range(size_vector)]
else:
for i in range(len(self.__data)):
n = self.__data[i].size()
self.__data[i].new_data(n)
def component(self, index: int):
if index < 0 or index >= len(self.__data):
raise IndexError("Index out of range for vector process!!!")
c = self.__data[index]
self.__new_data(c.math_except, c.dispersion, len(self.__data), len(c.data()))
return c
def opt_way_for_get_data(self): """ Функция, которая ищет наиболее
оптимальный способ
опроса векторного случайного процесса """
iw = 0
m = math.inf
miw = iw
permutations = list(itertools.permutations([i for i in
range(len(self.__data))]))
```

```
for way in permutations:
s = 0
for k in range(len(self.__data)):
s += math.log(abs(1 - self.component(way[k]).norm_correlation_func(len(self.__data) - k + 1) ** 2))
s *= -0.5
if s < m:
m = s
miw = iw
iw += 1
return permutations[miw]
def __str__(self):
s = ""
for i in range(len(self.__data)):
s += f"{i}: {self.__data[i]}"
if i != len(self.__data) - 1:
s += "\n"
return s
def size(self):
return len(self.__data)
def main():
print()
vec_proc = RandomVectorProcess(10, 3, 6, 5)
print("Начальное состояние векторного случайного процесса: ")
print(vec_proc)
print()
way = vec_proc.opt_way_for_get_data()
print(f"Оптимальный способ опроса: {way}")
print()
print("Результат опроса:")
```

```
data = [None for i in range(vec_proc.size())]
for ind in way:
data[ind] = vec_proc.component(ind)
for i in range(len(data)):
print(f"{i}: {data[i]}")
main()
```

Варианты заданий к лабораторной работе № 9

Варианты № 1 – № 5.

Построить компьютерный имитатор векторного случайного процесса $\xi = \{\xi_1,...,\xi_6\}$, компоненты ξ_I которого являются стационарными и обратимыми процессами авторегрессии второго порядка такими, что корреляционные функции процессов ξ_1, ξ_2, ξ_3, ξ_4 затухают быстрее, чем корреляционные функции процессов ξ_5 и ξ_6. Найти оптимальную процедуру опроса построенного векторного случайного процесса.

Варианты № 6 – № 10.

Построить компьютерный имитатор векторного случайного процесса $\xi = \{\xi_1,...,\xi_6\}$, компоненты ξ_I которого являются стационарными и обратимыми процессами скользящего среднего второго порядка такими, что корреляционные функции процессов ξ_1, ξ_2, ξ_3, ξ_4 меньше, чем корреляционные функции процессов ξ_5 и ξ_6. Найти оптимальную процедуру опроса построенного векторного случайного процесса.

Варианты № 11 – № 15.

Построить компьютерный имитатор векторного случайного процесса $\xi = \{\xi_1,...,\xi_6\}$, компоненты ξ_I которого являются стационарными и обратимыми процессами авторегрессии второго порядка такими, что корреляционные функции процессов ξ_2, ξ_3, ξ_4, ξ_5 затухают быстрее, чем корреляционные функции процессов ξ_1 и ξ_6. Найти оптимальную процедуру опроса построенного векторного случайного процесса.

Варианты № 16 – № 20.

Построить компьютерный имитатор векторного случайного процесса $\xi = \{\xi_1,...,\xi_6\}$, компоненты ξ_I которого являются стационарными и обратимыми процессами скользящего среднего второго порядка такими, что корреляционные функции процессов ξ_2, ξ_3, ξ_4, ξ_5 меньше, чем корреляционные функции процессов ξ_1 и ξ_6. Найти оптимальную процедуру опроса построенного векторного случайного процесса.

Варианты № 21 – № 25.

Построить компьютерный имитатор векторного случайного процесса $\xi = \{\xi_1,...,\xi_6\}$, компоненты ξ_I которого являются стационарными и обратимыми процессами авторегрессии второго порядка такими, что корреляционные функции процессов ξ_1, ξ_3, ξ_5, ξ_6 затухают быстрее, чем корреляционные функции процессов ξ_2 и ξ_4. Найти оптимальную процедуру опроса построенного векторного случайного процесса.

Варианты № 26 – № 30.

Построить компьютерный имитатор векторного случайного процесса $\xi = \{\xi_1,...,\xi_6\}$, компоненты ξ_I которого являются стационарными и обратимыми процессами авторегрессии второго порядка такими, что корреляционные функции процессов ξ_1, ξ_3, ξ_5, ξ_6 затухают быстрее, чем корреляционные функции процессов ξ_2 и ξ_4. Найти оптимальную процедуру опроса построенного векторного случайного процесса.