

Случайные процессы. Прикладной поток.

Практическое задание 6

Правила:

- Выполненную работу нужно отправить на почту `probability.diht@yandex.ru`, указав тему письма "[СП17] Фамилия Имя - Задание 6". Квадратные скобки обязательны. Вместо Фамилия Имя нужно подставить свои фамилию и имя.
- Прислать нужно ноутбук и его pdf-версию. Названия файлов должны быть такими: `6.N.ipynb` и `6.N.pdf`, где `N` - ваш номер из таблицы с оценками.
- При проверке могут быть запущены функции, которые отвечают за генерацию траекторий винеровского процесса.

1. Генерация винеровского процесса

Генерировать траектории винеровского процесса можно двумя способами:

1. На отрезке $[0, 1]$ траектория генерируется с помощью функций Шаудера. Описание данного метода было рассказано на лекции. Его можно найти так же в книге *А.В. Булинский, А.Н. Ширяев - Теория случайных процессов*.
2. На отрезке $[0, \pi]$ траекторию можно с помощью следующей формулы

$$W_t = \frac{\xi_0 t}{\sqrt{\pi}} + \sqrt{\frac{2}{\pi}} \sum_{k=1}^{+\infty} \frac{\sin(kt)}{k} \xi_k,$$

где $\{\xi_n\}$ --- независимые стандартные нормальные случайные величины.

Траектория для \mathbb{R}_+ генерируется с помощью генерирования отдельных траекторий для отрезков длины 1 или π (в зависимости от метода) с последующим непрерывным склеиванием.

Генерацию траекторий одним из этих способов вам нужно реализовать. Ваш вариант можете узнать из файла с распределением.

Напишите класс, который будет моделировать винеровский процесс. Из бесконечной суммы берите первые n слагаемых, где число n соответствует параметру `precision`. Интерфейс должен быть примерно таким (подчеркивания обязательны!):

In [33]:

```
import scipy.stats as sps
import numpy as np
import matplotlib.pyplot as plt
from time import time

%matplotlib inline
```

In [45]:

```

class ParticularBrounProcess:
    def __init__(self, precision=10000):
        #инициируем запуск конструктора
        self.k = np.arange(1, precision, 1)
        self.c0 = np.random.normal(loc=0, scale=1) / np.sqrt(np.pi)
        self.delta = self.c0 * np.pi
        self.c = np.random.normal(loc=0, scale=1, size=precision - 1) / \
            self.k * np.sqrt(2 / np.pi)
        self._calc_few = np.vectorize(self._few)

    def _few(self, w):
        return self.c0 * w + (np.sin(self.k * w) * self.c).sum()

    def __getitem__(self, r):
        return self._calc_few(r)

class WinerProcess:
    def __init__(self, precision=10000):
        self.precision = precision
        # уровни процесса
        self.pip = []
        self.lower_delta = [0]

    def _get_block(self, w):
        # номер блока
        b = int(w / np.pi)
        # сдвиг внутри блока
        period = w - b * np.pi
        # возвращаем значение
        return self.lower_delta[b] + self.pip[b][period]

    def _inc_1(self, count):
        if (len(self.pip) >= count):
            return
        self._inc_2(count, 0)
        self._inc_1(count)

    def _inc_2(self, count, it):
        if (len(self.pip) >= count or it > 200):
            return
        self._inc_3(count, 0)
        self._inc_2(count, it + 1)

    def _inc_3(self, count, it):
        if (len(self.pip) >= count or it > 200):
            return
        # добавляем блок
        self.pip.append(ParticularBrounProcess(self.precision))

        self.lower_delta.append(self.lower_delta[-1] + self.pip[-1].delta)
        self._inc_3(count, it + 1)

    def __getitem__(self, x):
        # если что добавляем число блоков
        self._inc_1(int(max(x) / np.pi) + 1)
        return list(map(lambda w: self._get_block(w), x))

```



- Экземпляр класса должен представлять некоторую траекторию винеровского процесса. Это означает, что один и тот же экземпляр класса для одного и того же момента времени должен возвращать одно и то же значение. Разные экземпляры класса --- разные (п.н.) траектории.
- Метод `__init__` (конструктор) должен запоминать число слагаемых в сумме (`precision`), а также (может быть) генерировать необходимые случайные величины для начального отрезка.
- Метод `__getitem__` должен принимать набор моментов времени и возвращать значения траектории винеровского процесса в эти моменты времени. При необходимости можно сгенерировать новые случайные величины. Используйте то, что запись `x.__getitem__(y)` эквивалентна `x[y]`.
- Для получения полного балла и быстро работающего кода реализация должна содержать не более одного явного цикла (по отрезкам при непосредственной генерации). Вместо всех остальных циклов нужно использовать функции библиотеки `numpy`.
- Внимательно проверьте отсутствие разрывов траектории в точках, кратных π .
- Имена любых вспомогательных методов должны начинаться с одного подчеркивания.
- В реализации желательно комментировать (почти) каждую строку кода. Это даже больше поможет вам, чем нам при проверке.

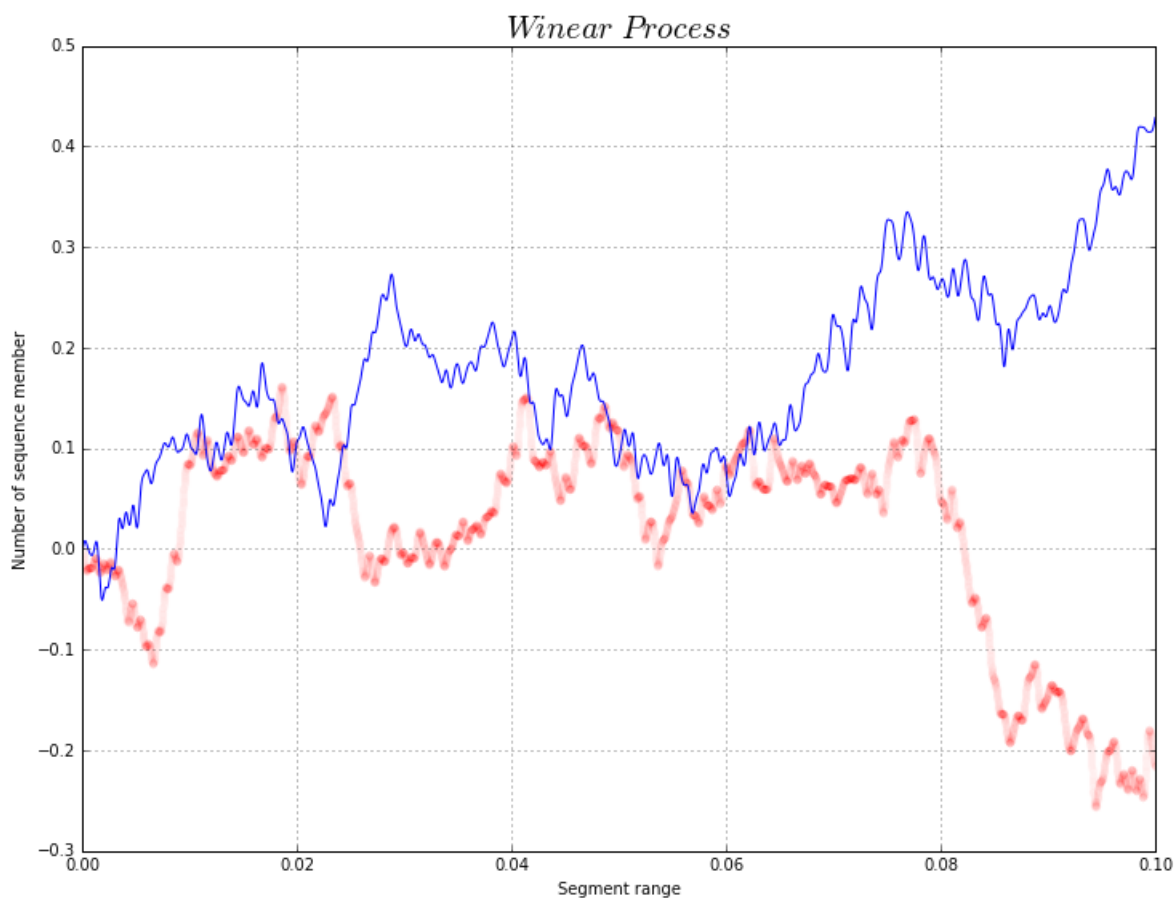
Сгенерируйте траекторию винеровского процесса и постройте ее график. Сгенерируйте еще одну траекторию и постройте график двумерного винеровского процесса. Графики должны быть похожими на графики с семинара. В качестве параметров возьмем

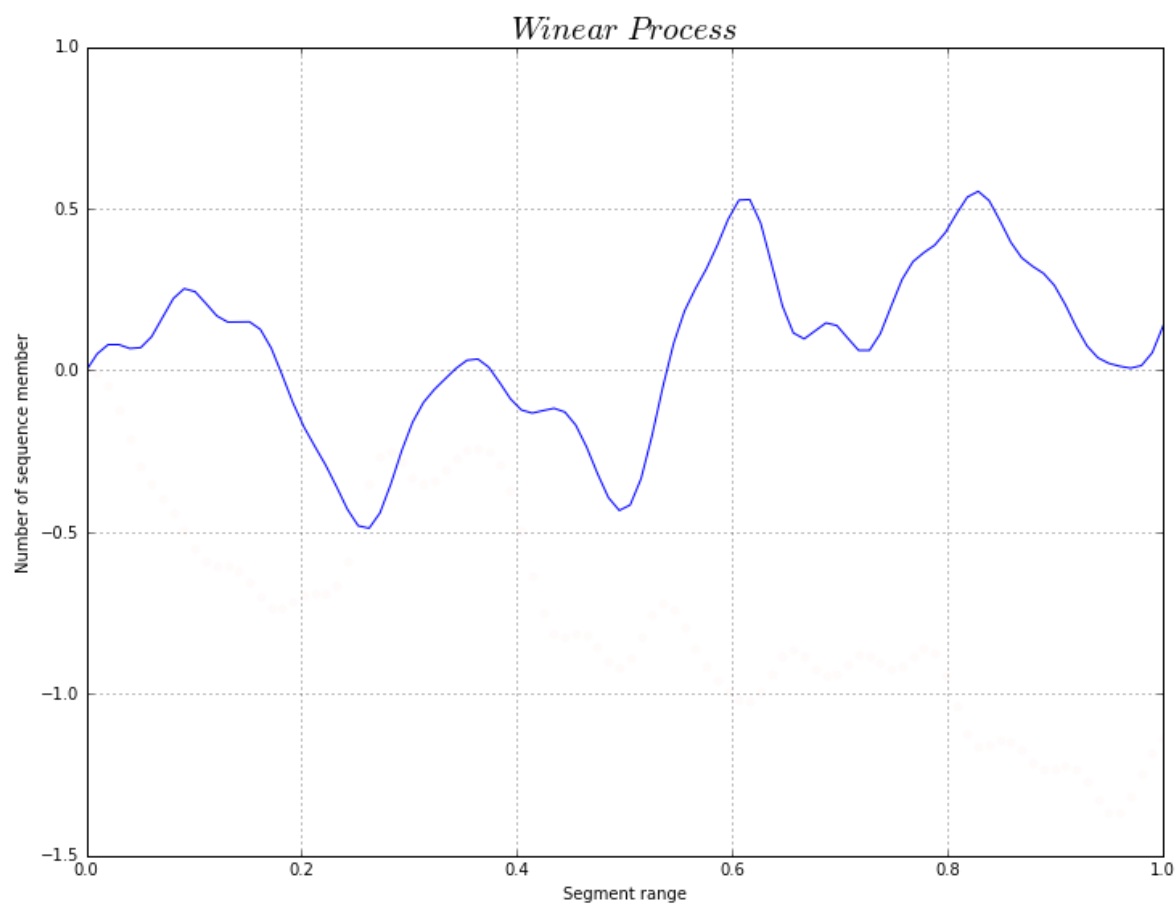
$$n = 1000, 100, \text{length} = 0.1, 1$$

In [58]:

```
def make_one_dimentional_plot(n, length):
    plt.figure(figsize=(12, 9))
    plt.xlabel('Segment range')
    plt.ylabel('Number of sequence member')
    plt.title("$Winear$ $Process$", size=20)
    plt.grid(ls=":")
    # генерим моменты времени
    x = np.linspace(0, length, n)
    # генерим процесс и строим график
    plt.plot(x, WinerProcess(n)[x])
    plt.xlim(0, length)
    plt.scatter(x, WinerProcess(n)[x], alpha=0.01, color = 'r')
    plt.show()
```

```
make_one_dimentional_plot(10000, 0.1)
make_one_dimentional_plot(100, 1)
```

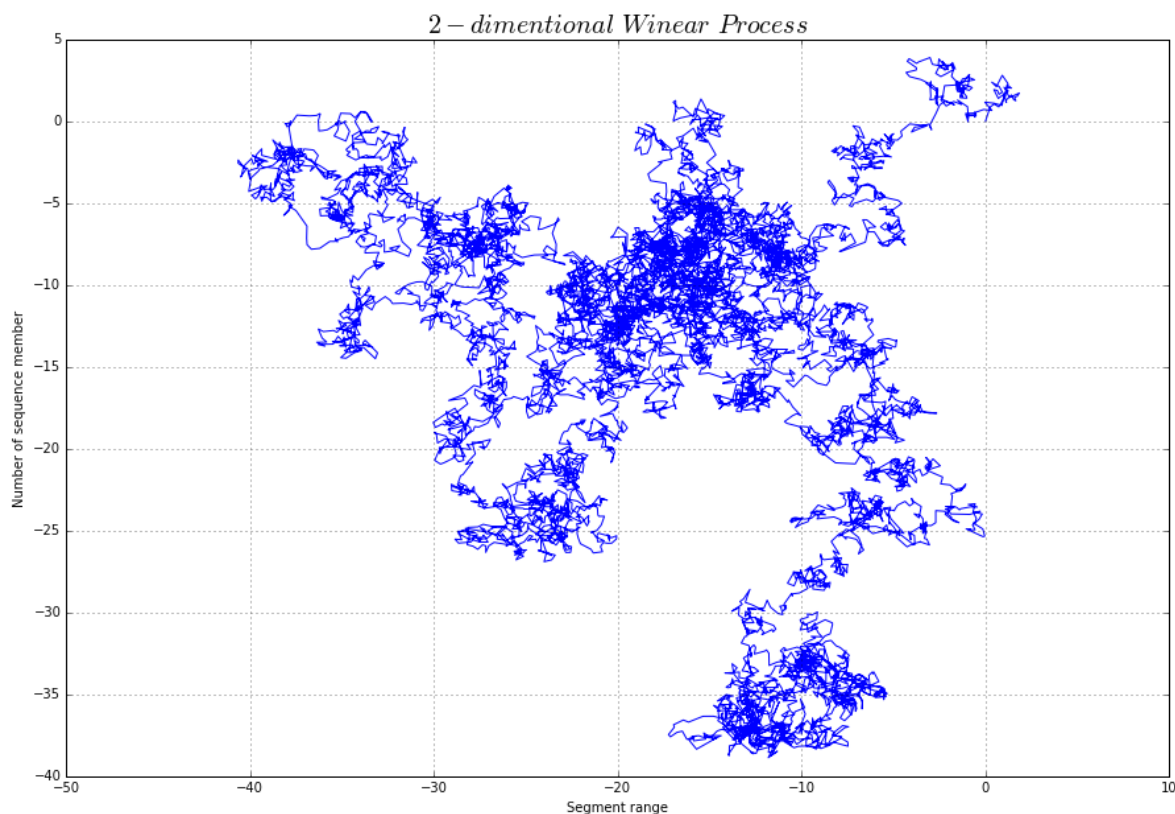




Получили Винеровский одномерный процесс. Построим теперь график для **двумерного процесса**:

In [5]:

```
def make_two_dimentional_plot(n, length):
    plt.figure(figsize=(15, 10))
    plt.xlabel('Segment range')
    plt.ylabel('Number of sequence member')
    plt.title("$2-dimentional$ $Winear$ $Process$", size=20)
    plt.grid(ls=":")
    # генерим моменты времени
    x = np.linspace(0, length, n)
    y = np.linspace(0, length, n)
    # генерим процесс и строим график
    plt.plot(WinerProcess(n)[x], WinerProcess(n)[x])
    #plt.scatter(x, WinerProcess(n)[x])
    plt.show()
make_two_dimentional_plot(10000, 1000)
```



Допустим, для исследования свойств траекторий винеровского процесса нам нужно сгенерировать траекторию с хорошей точностью до достаточно большого значения t . Какие проблемы могут возникнуть при использовании реализованного класса? Для этого попробуйте запустить следующий код.

In [47]:

```
#Wt = WinerProccess()
#t = np.linspace(0, 10 ** 7, 10 ** 5)
#values = Wt[t]
```

закомментировала ячейку выше, чтобы случайно не запустить ее еще раз :(На этой ячейке мой компьютер виснет и отказывается работать. Предполагаю, что основных причин тут две:


1. Ограничение на глубину рекурсии. Эту проблему можно обойти, модифицировав код. В интерпретаторе **Python** по умолчанию установлено ограничение глубины рекурсии равное 1000, но этот численный параметр можно переустановить.

2. Ограничение оперативной памяти. Прикинем, сколько здесь её потребуется: $I = 10^7 \cdot \frac{10^4}{\pi}$, считая все данные однотипными, получим примерно $I_1 = 0.37$ **терабайт!**, при оперативной памяти конкретно моего компьютера всего **4Гб**.

Для избавления от таких проблем реализуйте следующую функцию:

In [48]:

```
def winer_proccess_path(end_time, step, precision=10000):
    # Моменты времени, в которые нужно вычислить значения
    times = np.arange(0, end_time, step)
    my_times = times * (10000 / end_time)
    # Сюда запишите значения траектории в моменты времени times
    values = WinerProcess(precision)[my_times] * np.sqrt(end_time / 10000)
    return times, values
```



Для получения полного балла и быстро работающего кода реализация должна содержать не более одного явного цикла (по отрезкам при непосредственной генерации). Вместо всех остальных циклов нужно использовать функции библиотеки `numpy`. Внутри этой функции можно реализовать вспомогательную функцию.

2. Исследования

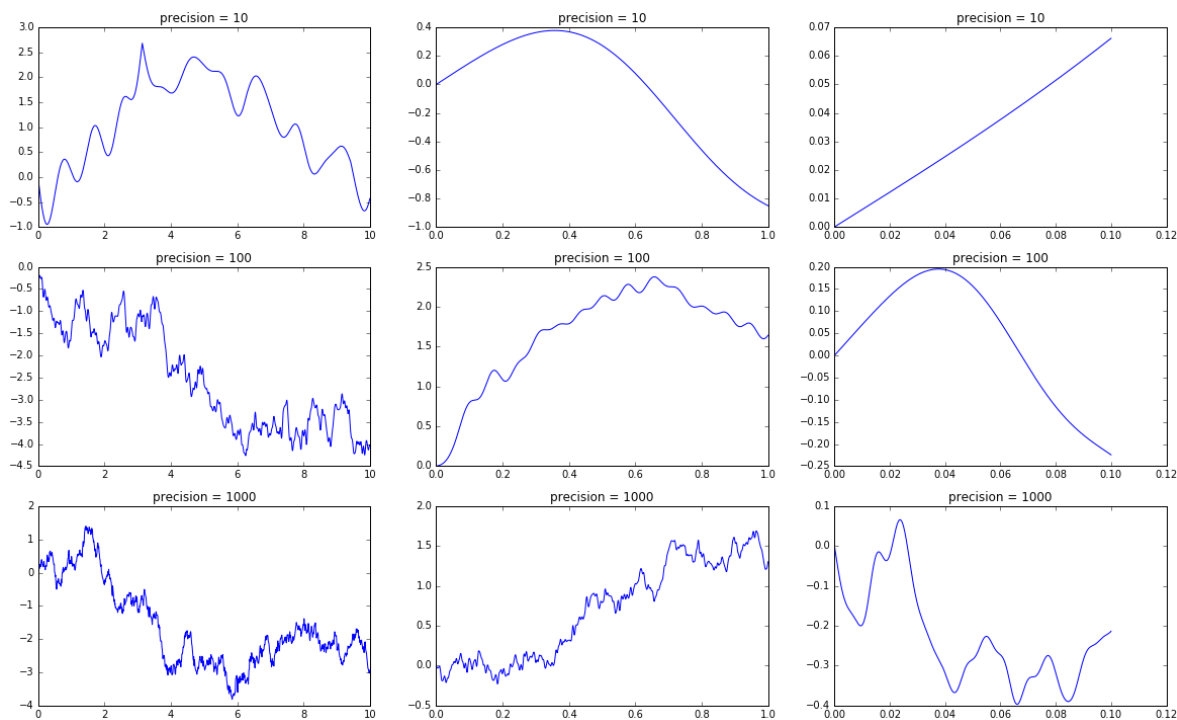
Следующая часть работы делается в паре.

Для каждого из двух способов генерация траектории винеровского процесса постройте таблицу 3×3 из графиков траекторий винеровского процесса. По вертикали изменяйте количество n используемых слагаемых в сумме ($n = 10; 100; 1000$), по горизонтали --- длину отрезка, на котором генерируется винеровский процесс (использовать отрезки $[0, 10]$, $[0, 1]$, $[0, 0.1]$). Обратите внимание, что от размера сетки зависит только точность отображения функции на графике, а не сама функция, поэтому сетку нужно выбирать достаточно мелкой.

In [49]:

```
plt.figure(figsize=(20, 12))
for i, precision in enumerate([10, 100, 1000]):
    for j, max_time in enumerate([10, 1, 0.1]):
        t = np.linspace(0, max_time, 1000)
        values = WinerProcess(precision=precision)[t]

        plt.subplot(3, 3, i * 3 + j + 1)
        plt.plot(t, values)
        plt.title('precision = %d' % precision)
plt.show()
```



Какие выводы можно сделать про каждый способ генерации?

Вывод (по моему методу):

Наиболее похожий на Винервский процесс график и вообще то, что было на лекции, -- левый нижний, то есть тот, где больше всего членов суммы и тот, где расстояния между генерациями самые маленькие. В противоположной ситуации он вовсе вырождается почти в прямую

Постройте графики полученных траекторий для каждого способа? Отличаются ли траектории визуально?

Какие можно сделать выводы из сравнения двух способов генерации?

Следующая часть работы делается индивидуально.

1. Сгенерируйте 100 траекторий винеровского процесса с достаточно хорошей точностью и нарисуйте их на одном графике? Что можно сказать про поведение траекторий?
2. Нарисуйте график двумерного винеровского процесса (см. презентацию с семинара).

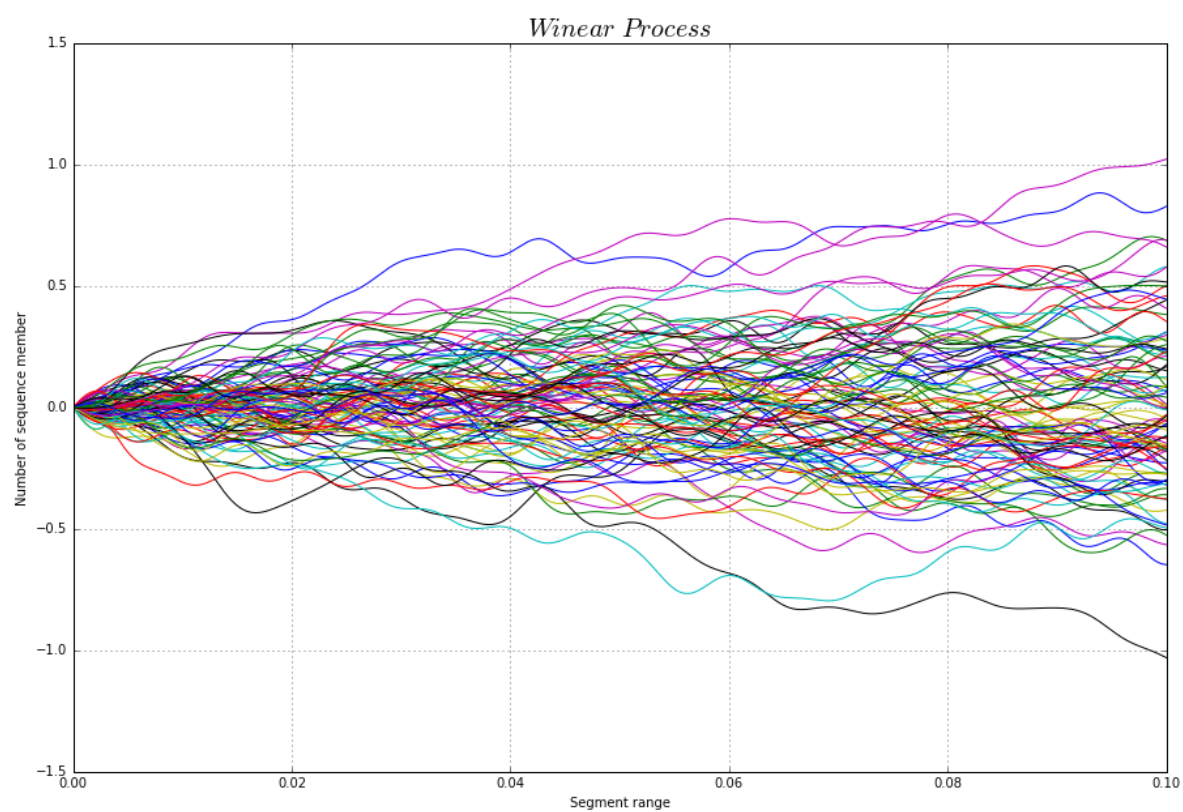
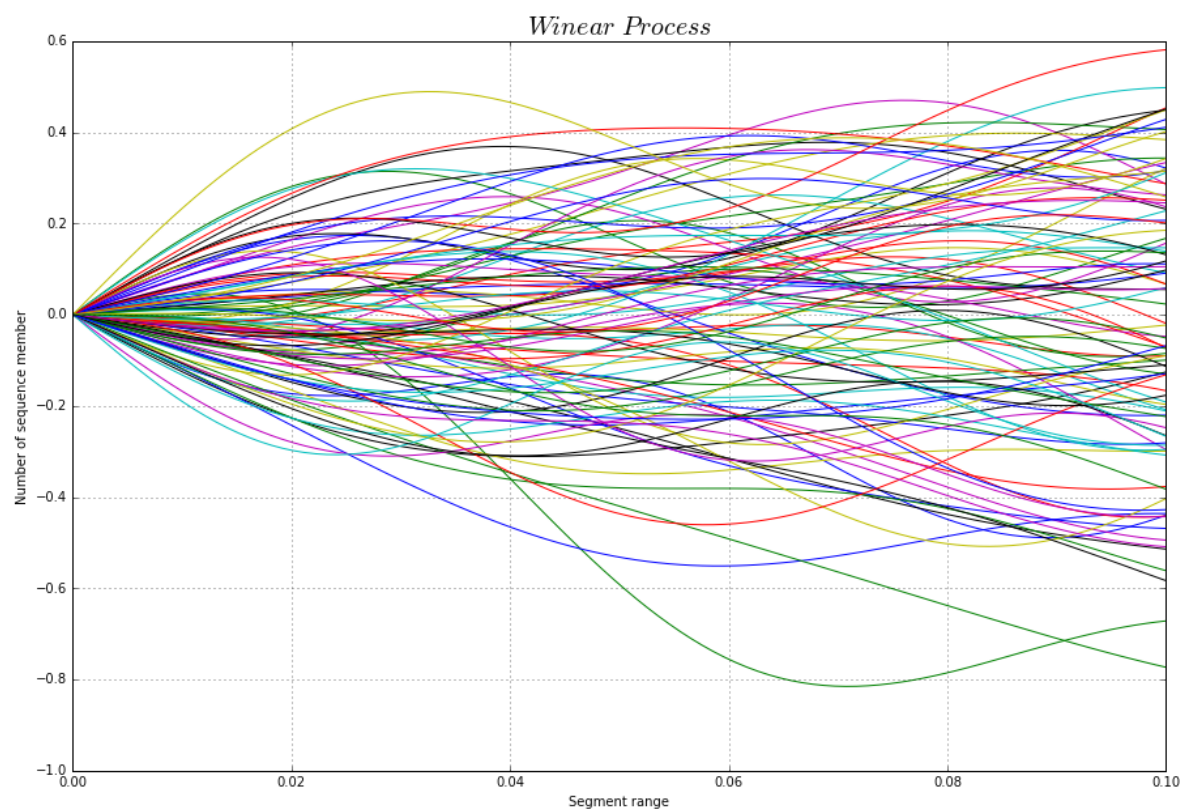
Сгенерируем 100 траекторий винеровского процесса и нарисуем их на одном графике:

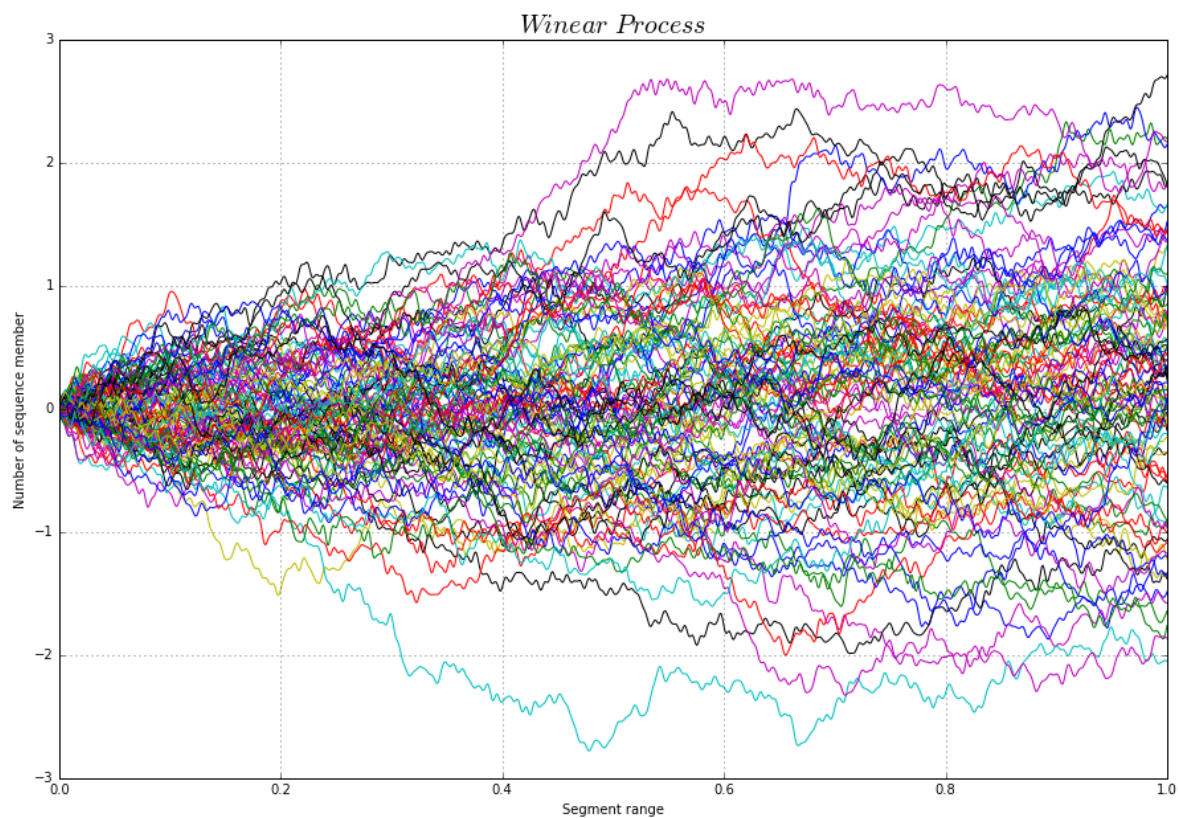
In [50]:

```
length = 0.1
n = 100
plt.figure(figsize=(15, 10))
plt.xlabel('Segment range')
plt.ylabel('Number of sequence member')
plt.title("$Winear$ $Process$",size=20)
plt.grid(ls=":")
for i in range(100):
    x = np.linspace(0, length ,n)
    plt.plot(x, WinerProcess(n)[x])
plt.xlim(0, length)
plt.show()

length = 0.1
n = 1000
plt.figure(figsize=(15, 10))
plt.xlabel('Segment range')
plt.ylabel('Number of sequence member')
plt.title("$Winear$ $Process$",size=20)
plt.grid(ls=":")
for i in range(100):
    x = np.linspace(0, length ,n)
    plt.plot(x, WinerProcess(n)[x])
plt.xlim(0, length)
plt.show()

length = 1
n = 1000
plt.figure(figsize=(15, 10))
plt.xlabel('Segment range')
plt.ylabel('Number of sequence member')
plt.title("$Winear$ $Process$",size=20)
plt.grid(ls=":")
for i in range(100):
    x = np.linspace(0, length ,n)
    plt.plot(x, WinerProcess(n)[x])
plt.xlim(0, length)
plt.show()
```





Итак, как мы видим, наши предыдущие выкладки подтверждаются на больших выборках -- чем больше число слагаемых, тем больше траектории напоминают Винервские процессы и тем быстрее расходятся.

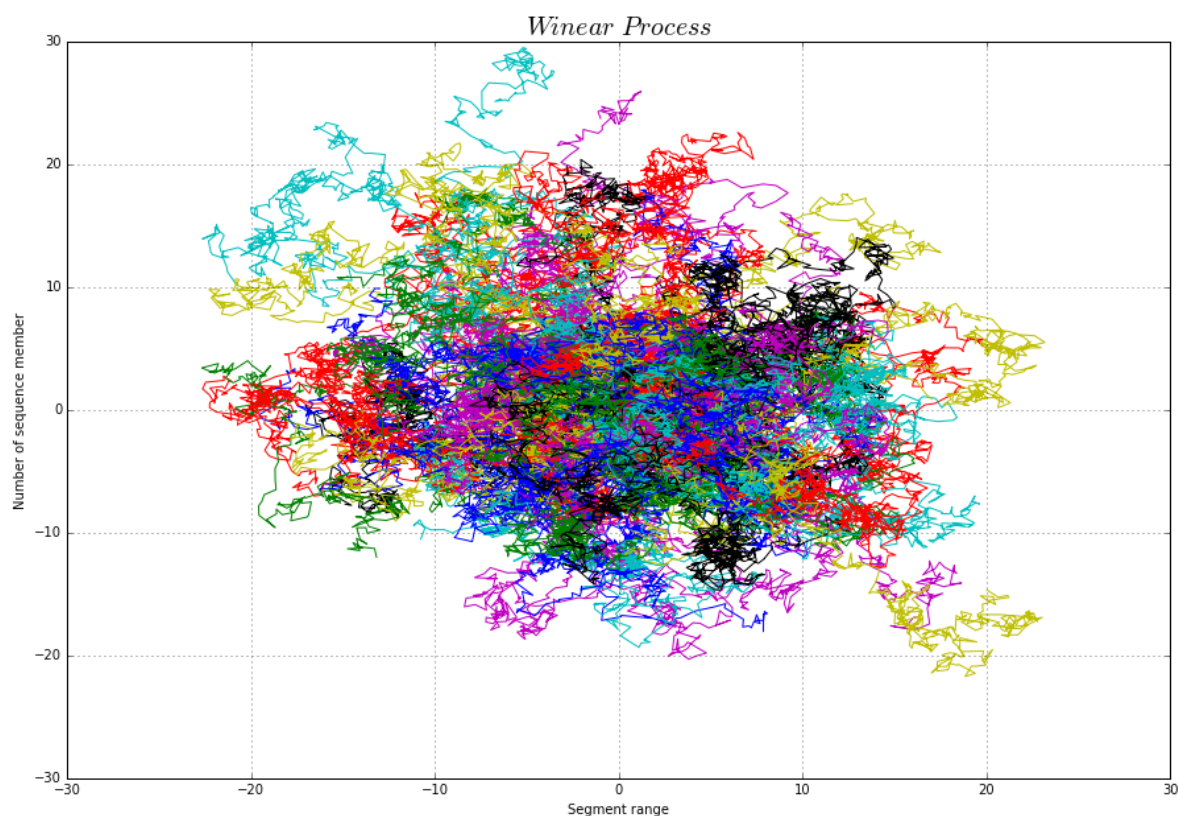
А теперь сделаем то же самое для двумерного процесса:

In [61]:

```

length = 100
n = 500
plt.figure(figsize=(15, 10))
plt.xlabel('Segment range')
plt.ylabel('Number of sequence member')
plt.title("$Winear$ $Process$",size=20)
plt.grid(ls=":")
for i in range(100):
    x = np.linspace(0, length ,n)
    y = np.linspace(0, length ,n)
    # генерим процесс и строим график
    plt.plot(WinerProcess(n)[x],WinerProcess(n)[x] )
plt.show()

```



In []:

