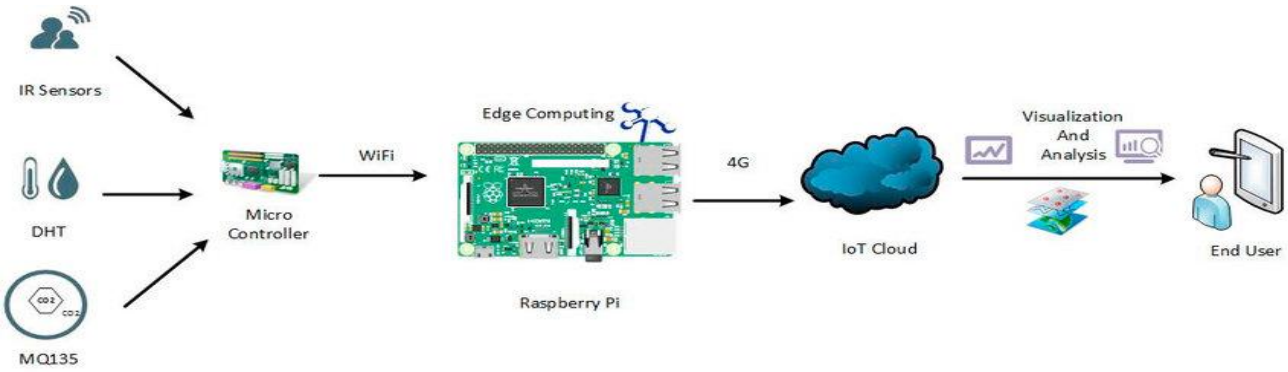
REPUBLIQUE TUNISIENNE
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université de Carthage
Institut National des Sciences Appliquées et de Technologie

RT5 - INSAT **Ouni Sofiane**

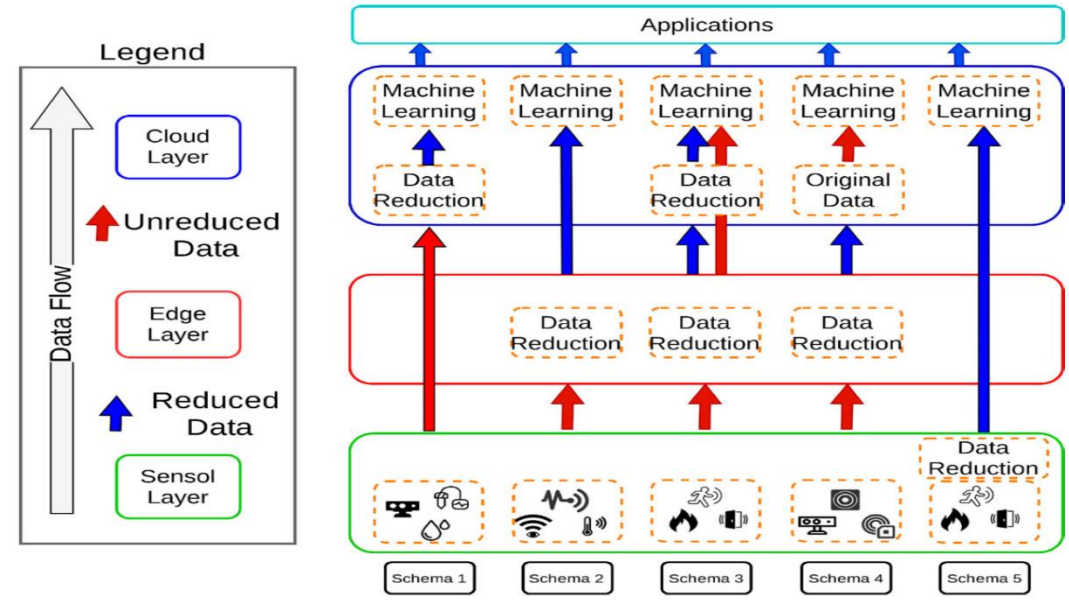
Technologies et Protocols pour l'IoT

TP2 : Edge computing and data filtering for Industrial IoT (IIoT)

Raspberry as an EDGE computing node



Edge-cloud architecture schema



Edge-cloud architectural schema

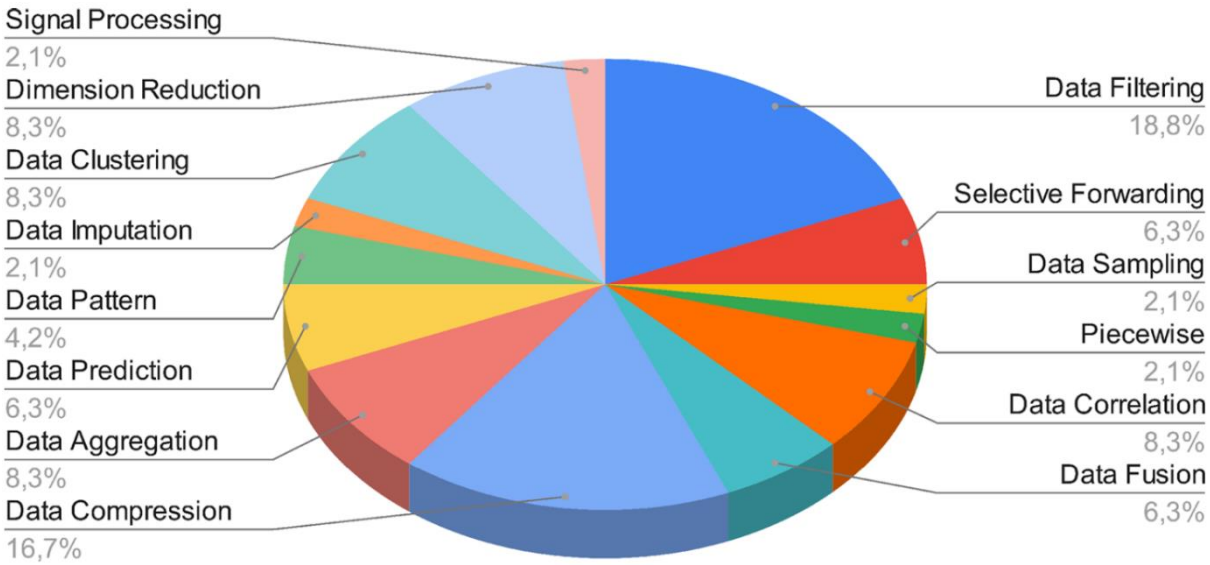
" Internet of Things (IoT) is a technology that connects devices of different types and characteristics through a network. ***The massive quantity*** *of the heterogeneous* ***generated data*** *by the sensors imposes many challenges in making these data available to IoT applications. Data reduction and preprocessing are promising concepts that help to handle these data*

efficiently before storing them. ***Applying data reduction methods at the edge*** *has emerged as an efficient solution. ”*, Pioli et al**. [1]**

[1] Pioli, L., Dorneles, C.F., de Macedo, D.D.J. et al**. An overview of data reduction solutions at the edge of IoT systems: a systematic mapping of the literature**. Computing 104, 1867–1889 (**2022**). https://doi.org/10.1007/s00607-022-01073-6, https://link.springer.com/article/10.1007/s00607-022-01073-6#citeas

I. Background on Data reduction

The following Figure addresses the question: Which **techniques researchers** are applying to perform data **reduction at the edge**?. Throughout this research question, we can understand, in general, which are the most used techniques to perform a data reduction at the edge. These results bring us interesting information once the **hardware used at the edge has less processing power than** the used in **the cloud**. For instance, **data filtering 18.8%** and **data compression 16.7%** were the most used techniques to **reduce data volume at the edge**. [1]



Used technique

II. Data filtering and fusion

The main of **data reduction** approaches are to solve **the problems** of **network bandwidth**, **I/O throughput**, network **energy consumption** and **cloud storage**.

- The work [2] proposed a **data filtering and fusion** in-networking approach to **reduce the IoT sensor data**. The **data filtering** solution is based **on data change detection** and **deviation** and compose the first step of the solution. After filtering, in the second layer, it occurs the **data fusion** is based on a **minimum square error**.

[2] Ismael WM, Gao M, Al-Shargabi AA, Zahary A (2019) An in-networking double-layered data reduction for internet of things (iot). Sensors 19(4):795, https://www.mdpi.com/1424-8220/19/4/795

- In Reference [3], the authors proposed an approach of data reduction based on two phases. In the first phase, the **data are modeled** based on multivariate **normal distribution**. In the second phase, **a Kalman filter** with the same parameters is deployed. The **estimated values** of measured data by end devices are calculated, based on **historical data** and **internal data correlation**. If the predicted **values exceed the prediction range**, the **observations are forwarded** to the cloud.

  Kalman filter is known as a recursive algorithm for optimal estimating future states of dynamic systems. It is commonly used as an estimation algorithm and it is also known as an adaptive filter to combine previously estimated values with current measurements.[2]

[3] *Yu, T.; Wang, X.; Shami, A. A Novel Fog Computing Enabled Temporal Data Reduction Scheme in IoT Systems. In Proceedings of the GLOBECOM 2017–2017 IEEE Global Communications Conference, Singapore, 4–8 December 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 1–5.*

- The study [4] presented a two-tier **data reduction framework** that implements a **gradient-based model**.

[4] Li PH,YounHY(2020) Gradient-based adaptivemodeling for iot data transmission reduction. Wireless Netw, 26(8):6175–6188

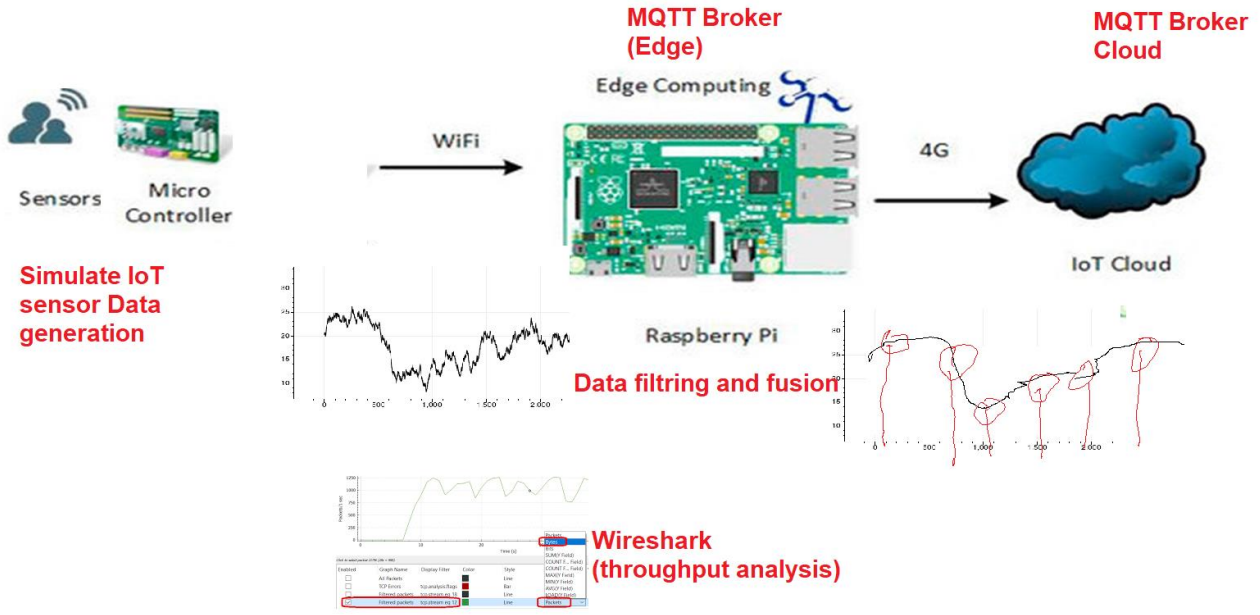
- The usage of **AI (Artificial intelligence)** to support **data preprocessing** seems to be an interesting solution to **increase data quality** when using the edge architecture model. The **ML (Machine Learning)** closer to the data source might **reduce the data transmission**. The authors in [5] proposed an **architecture** that introduces the **ML closer to the data source**. They discuss the three main techniques (e.g., Device and Edge, Edge and Cloud, and Device and Cloud) to perform ML considering the edge computing paradigm.

[5] HafeezT, Lina X, McardleGavin (2021) **Edge intelligence for data handling** and **predictive** maintenance in iiot. IEEE Access 9:49355–49371

III. LAB on Data filtering and fusion for MQTT raspberry Edge computing

Le but du LAB ou du TP est de réaliser une maquette simple pour interpréter les traitements de filtrage et de fusion des données au niveau Edge. La Maquette est formée par :

- Dans Votre PC, une application de simulation de génération des données physique d'un capteur, ces données sont ensuite publiées (publish) vers le broker MQTT dans le Edge Raspberry
- Dans le Edge Raspberry (machine virtuelle ou carte raspberry), une application va recevoir les données puis appliquer un filtre et la fusion de ces données pour les réduire. Puis, les données réduites sont envoyées vers le cloud simulé par votre PC. Il doit à son tour avoir un broker pour recevoir les données réduites.
- Nous utilisons Wireshark dans le PC (plus rapide) pour analyser et faire le graphique du débit (throughput) du trafic généré des données par le capteur simulé. Puis faire aussi l'affichage du trafic réduit envoyé vers le cloud. Cela permettra de visualiser le gain en bande passante.

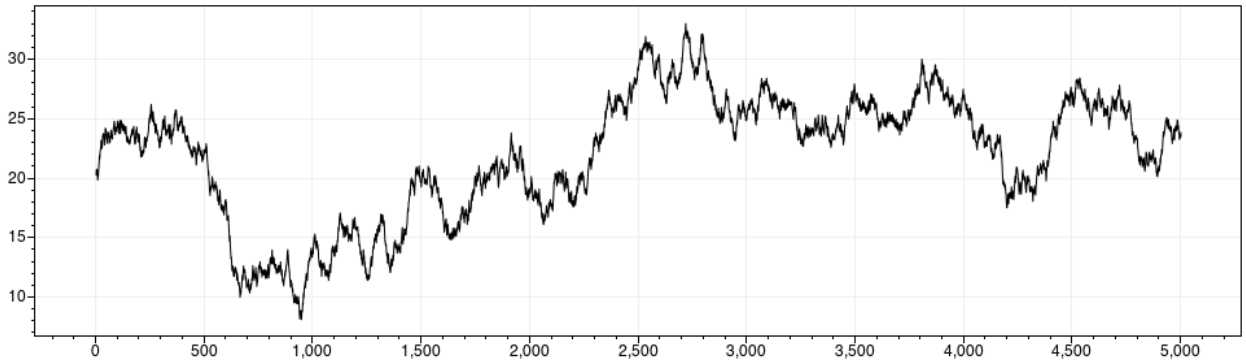


Afin de pourvoir finaliser ce TP nous procédons par étape dans les questions suivante :

Etape 1 : Génération de trafic MQTT simulé de capteur IoT et publication vers le Edge

Réaliser une application python qui permet de générer aléatoirement des valeurs de données continues et puis de les publier périodiquement vers le broker dans le Edge (Raspberry). Pour réaliser la génération simulée avec des valeurs relativement réalistes comme d'un capteur physique (température, pression,….). Le simulateur permet de gérer des valeurs aléatoires mais respectant une certaine variation.

1.1. Il faut commencer à tester le code type de génération de données industrielles réalisé dans le line : https://bitperfect.at/en/blog/simulation-von-sensordaten , le code en CS est mis à la fin dans les annexes. Il faut convertir le code en python et faire l'affichage graphique du Data set (liste) généré.



Un exemple de code python pour affichage graphe de dataset :

```
importing the required module
import matplotlib.pyplot as plt

# x axis values
x = [1,2,3]
# corresponding y axis values
y = [2,4,1]

# plotting the points
plt.plot(x, y)

# naming the x axis
plt.xlabel('x - axis')
# naming the y axis
plt.ylabel('y - axis')

# giving a title to my graph
plt.title('My first graph!')

# function to show the plot
plt.show()
```

1.2. Créer une application python qui publie avec MQTT les données du dateset et ce périodiquement. Il Faut vérifier la bonne réception de ces données par le broker (Raspberry). Vous pouvez utiliser la ligne de commande mosquitto_sub pour visualiser textuellement ces données.

1.3. Utiliser wireshark pour visualiser le débit de ce trafic publié (published topic). Il faut utiliser le filtre sur MQTT pour ne visualiser que le trafic MQTT.

Wireshark (throughput analysis)

1.4. Nous voudrons maintenant que la publication ne soit pas uniformément périodique car cela est trop parfait. Pour cela, nous supposons de publications dans des instants aléatoires mais avec une valeur moyenne de publications (lamdda λ). Nous allons simuler des arrivées ou des publications Poissonniennes. Pour vous faciliter la tâche voici un code démarrage à utiliser. Ce code permet de publier en MQTT selon lambda qui est dans le code l=10 dans l'exemple, c'est-à-dire 10 arrivées ou publications par secondes. Une fois vous avez testé le code, il faut l'adapter pour qu'il transmet les données simulées du Dataset.


```
import paho.mqtt.client as mqtt
import time
import math
import random

def next_time_interval (l):
   n=random.random()*1.0
   inter_event_time = -math.log(1.0 - n)/l
   return (inter_event_time)

# l lambda number of messages per second (poissonian arrival model)
def Generation_pub_messages_oneTopic (client, l, NbreMessages):
   global Time
   l=10
   NbreMessages=20
   i=0
   while i<NbreMessages :
    tt= next_time_interval (l)
    print("Publishing Time:",Time)
    time.sleep(tt) # wait
    Time+=tt
    m="m"+str(i)
    client.publish("top/top1",m)
```

```
    i=i+1

########################################
global Time

broker_address="127.0.0.1"
print("Start message publication....")
client = mqtt.Client("Pub")
client.connect(broker_address)
print ("start periodic publish")
# l : lambda arrivals per seconds

Time=0
l=10
NbreMessages=100
Generation_pub_messages_oneTopic (client, l, NbreMessages)
```

1.5. Utiliser wireshark pour visualiser le débit de ce trafic publié (published topic) .Comparer avec le résultat précédant.

Etape 2 : Création de l'application de filtrage, réduction simplifié et de publication vers le cloud

Après avoir réalisé le code de publication vers le Edge broker, il faut créer une application qui reçoit ces données et puis fait le filtrage et la transmission de données réduites. Pour simplifier, ici la réduction

2.1. Pour la réduction des données, nous allons maintenant utiliser le principe d'échantillonnage des données pour n'envoyer les données que dans des intervalles de temps précis. Créer une application python qui reçoit les données du Edge broker les ajoute dans une liste, puis à chaque période on envoi ou publie vers le broker du cloud (simulé par votre PC) la moyenne des données de cette dernière liste. La liste est vidée à chaque période. filter in wireshark by topic

2.2. Utiliser Wireshark pour afficher le débit reçu par le cloud. Comparer ce débit avec celui soumis au début par le capteur.

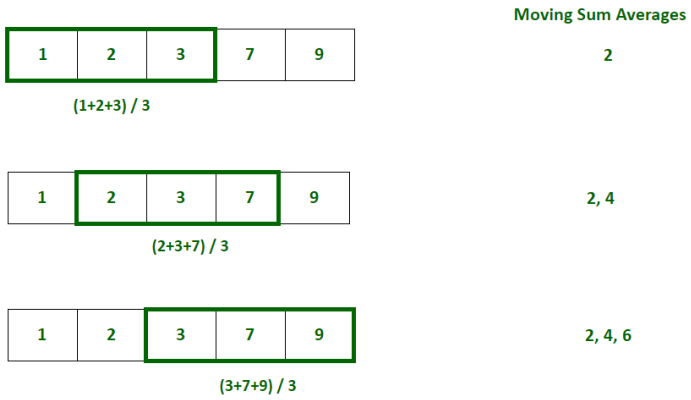
Etape 3 : Création de l'application de filtrage, réduction et de publication vers le cloud

Dans cette partie nous voudrons améliorer le filtrage et la fusion pour avoir des valeurs plus réalistes.

3.1. Créer une autre application qui permet de lire les données publiées vers le cloud et d'afficher le graphique. Comparer la courbe de variation des données cotés capteur avec la courbe de variation des données dans le cloud. Vous allez remarquer qu'il y a des imprécisions dans le calcul de la moyenne.

3.2. Pour résoudre le problème précédant, modifier dans votre code précédant le calcul de la moyenne selon le principe de la fenêtre glissante (Moving Averages). Ce principe est détaillé dans le lien suivant : https://www.geeksforgeeks.org/how-to-calculate-moving-averages-in-python/

Exemple de Moving average :



L'équation de la moyenne selon la fenêtre glissante sera :

$$\bar{x}_i = \frac{1}{2M+1} \sum_{j=-M}^{j=M} x[i+j]$$

Où le point de données moyen est calculé à partir de la moyenne des points contenus dans la fenêtre glissante définie comme +/- M et centrée autour de x_i.

Code :

```
# Program to calculate moving average
arr = [1, 2, 3, 7, 9]
window_size = 3

i = 0
# Initialize an empty list to store moving averages
moving_averages = []

# Loop through the array to consider
# every window of size 3
while i < len(arr) - window_size + 1:

    # Store elements from i to i+window_size
    # in list to get the current window
    window = arr[i : i + window_size]

    # Calculate the average of current window
    window_average = round(sum(window) / window_size, 2)
```

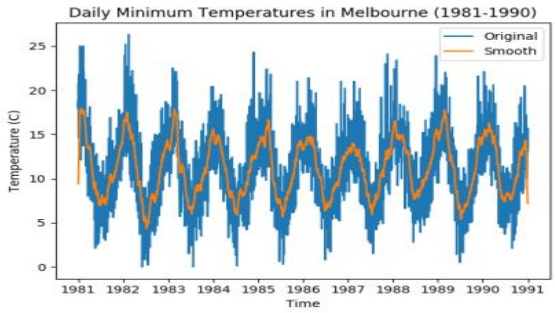
```
    # Store the average of current
    # window in moving average list
    moving_averages.append(window_average)

    # Shift window to right by one position
    i += 1

print(moving_averages)
```

Autre référence : https://waterprogramming.wordpress.com/2018/09/04/implementation-of-the-moving-average-filter-using-convolution/

3.3. Vérifiez les deux courbes de données : originalement créé par le capteur et celle réduite. Vous allez voir la concordance des deux courbes comme le montre cette figure :



Annexes

Generate Normal Distributed Consecutive Values

https://bitperfect.at/en/blog/simulation-von-sensordaten

`BasicSimulatorClass.cs`

```
public class Simulator
{
   private readonly Random _random;
   private readonly float _mean;
   private readonly float _standardDeviation;
   private readonly float _stepSizeFactor;
   // _value is of type double to reduce necessity of casting to float
   private double _value;

   public Simulator(int seed, float mean, float standardDeviation)
   {
      _random = new Random(seed);
      _mean = mean;
      _standardDeviation = Math.Abs(standardDeviation);
```

```
        // we define a _stepSizeFactor that is used when calculating the
        // next value
        _stepSizeFactor = _standardDeviation / 10;
        // we set a starting _value which is not exactly _mean (it could be
        // but my personal preference is to not have each data set start on
        // the same value)
        _value = _mean - _random.NextDouble();
    }
}

private static readonly List<int> Factors = new(){-1, 1};

public double CalculateNextValue()
{
    // first calculate how much the value will be changed
    double valueChange = _random.NextDouble() * _stepSizeFactor;
    // second decide if the value is increased or decreased
    int factor = Factors[DecideFactor()];

    // apply valueChange and factor to _value and return
    _value += valueChange * factor;
    return _value;
}

private int DecideFactor()
{
    // the distance from the _mean
    double distance;
    int continueDirection;
    int changeDirection;

    // depending on if the current value is smaller or bigger than the mean
    // the direction changes are flipped: 0 means a factor of -1 is applied
    // 1 means a factor of 1 is applied
    if (_value > _mean)
    {
        distance = _value - _mean;
        continueDirection = 1;
        changeDirection = 0;
    }
    else
    {
        distance = _mean - _value;
```

```
        continueDirection = 0;
        changeDirection = 1;
    }

    // the chance is calculated by taking half of the _standardDeviation
    // and subtracting the distance divided by 50. This is done because
    // chance with a distance of zero would mean a 50/50 chance for the
    // randomValue to be higher or lower.
    // The division by 50 was found by empiric testing different values
    double chance = (_standardDeviation / 2) - (distance / 50);
    double randomValue = _random.NextDouble() * _standardDeviation;

    // if the random value is smaller than the chance we continue in the
    // current direction if not we change the direction.
    return randomValue < chance ? continueDirection : changeDirection;
}
```

TestBasicSimulator.cs Copy to clipboard

```
List<double> dataSet = new List<double>();
Simulator sim = new Simulator(seed: 12345, mean: 20, standardDeviation: 5);

for(int i = 0; i < 100000; i++)
{
    dataSet.Add(sim.CalculateNextValue);
}
```