Universidad Politécnica
de Madrid

Escuela Técnica Superior de

Ingenieros Informáticos

Máster Universitario en Ciencia de Datos

Trabajo Fin de Máster

Time Series Random Generator

Author: Jorge Martín Lasaosa

Supervisors: Aurora Pérez and Juan Pedro Caraça-Valente

Madrid, June 2020

Trabajo Fin de Máster

Máster Universitario en Ciencia de Datos

Título: Time Series Random Generator

June 2020

Autor: Jorge Martín Lasaosa

Tutores:

Aurora Péréz y Juan Pedro Caraça-Valente
Departamento de lenguajes y sistemas informáticos e ingeniería de software
ETSI Informáticos
Universidad Politécnica de Madrid

Abstract

A time series is an ordered sequence of values. This data type appears in many real-life domains, as the result of measuring continuously any kind of variable. For example, in medicine, seismology, finances…

Lots of projects have been carried out in the field of time series prediction or analysis. There are many attempts to predict the stock market in financial companies. Also, in medicine the patient cardiograms are analysed using data mining techniques for finding anomalies, etc.

Due to its many applications, this is a field studied for many years. However, it has acquired a new dimension with the arrival of COVID-19, and many projects have been started trying to help in the current situation.

For this master's thesis, the objective is twofold. First, a time series random generator must be designed and developed. Given some parameters, it can generate synthetic random time series. These parameters are related to time series properties such as trend, periodicity, the appearance of events like peaks or valleys...

Secondly, another application will be designed and developed. It will use data mining techniques to group the different synthetic generated time series.

These techniques will consist of performing Fourier transformations on the time series, and using the coefficients obtained as inputs for a clustering algorithm. Thus, it will be important as a proper data mining tool but also as a means to prove the correct functioning of the time series random generator and its many functionalities.

Resumen

Una serie temporal es una secuencia ordenada de valores. Este tipo de datos aparecen en muchos campos de la vida real, como resultado de una medición continua de cualquier tipo de variable. Por ejemplo, en medicina, sismología, finanzas…

Se han llevado a cabo una gran cantidad de proyectos en el ámbito de la predicción o análisis de series temporales. Mientras en el ámbito de las finanzas, por ejemplo, se ha intentado predecir el comportamiento del mercado de valores, en medicina se analizan los cardiogramas de un paciente mediante técnicas de minería de datos en busca de anomalías.

Debido a estas y a otras muchas aplicaciones, este es un campo estudiado desde hace muchos años. Sin embargo, ha adquirido una nueva dimensión con la llegada del COVID-19, y se han lanzado muchos proyectos que intentan ayudar en la situación actual.

Para este trabajo de fin de máster, la finalidad es doble. En primer lugar, hay que diseñar y desarrollar una aplicación software que, dada una serie de parámetros, sea capaz de generar series temporales sintéticas. Estos parámetros referencian propiedades de las series temporales como la tendencia, la periodicidad, la aparición de eventos como picos o valles…

En segundo lugar, se procederá con el diseño y desarrollo de una segunda que, mediante técnicas de minería de datos, agrupe las diferentes series temporales sintéticas generadas.

Estas técnicas consistirán en realizar transformaciones de Fourier sobre las series temporales, y utilizar los coeficientes obtenidos como entradas para un algoritmo de agrupación. La aplicación desarrollada no sólo servirá como una herramienta de minería de datos sino también como una forma de probar el correcto funcionamiento del generador aleatorio de series temporales y sus muchas funcionalidades.

Contents

1 Introduction

This thesis is motivated by a need on the part of supervisors to create synthetic time series and it is divided into three main sections:

- State of the art
- Development
- Results and conclusions

In the first one, the theory behind the different algorithms or tools used during the development of the project will be explained. This section will try to give a theoretical approach which is highly based on bibliographic references.

In the development section, the applications mentioned before will be described. First, a time series random generator must be designed and developed. Given some parameters, it can generate synthetic random time series. These parameters are related to time series properties which can be tuned:

- Number of values, initial standard deviation and mean.
- Trend
- Periodicity
- Appearance of events like peaks or valleys
- Seed

Secondly, another application will be designed and developed. It will use data mining techniques to group the different synthetic generated time series.

These techniques will consist of performing Fourier transformations on the time series, and using the coefficients obtained as inputs for a clustering algorithm. Thus, it will be important as a proper data mining tool but also as a means to prove the correct functioning of the time series random generator and its many functionalities.

For each run of the time series generation tool, every time series need to be different from each other because of the random component. Nevertheless, they need to share a common base that identify them as a unique group. This factor will be checked at the end by the cluster algorithm.

Considering the requests made by the supervisors, both applications will be written in Python. As a personal decision, both projects will be developed using PyCharm Integrated Development Environment.

An iterative and incremental approach is going to be applied for the software tools development. The basic idea is to develop a system through repeated cycles (iterative) and in smaller portions at a time (incremental).

In each increment, a slice of functionality is delivered from the requirements to the deployment, and follows the next phases:

- Requirements
- Analysis & Design
- Implementation
- Application interface (only for the time series generator)
- Test
- Activity diagram (only for the time series generator)
- Validation

At the end of each increment, the PyInstaller package would be used to generate a ready-to-use executable. This package builds smaller executables thanks to transparent compression, it is fully multi-platform, and use the OS support to load the dynamic libraries, thus ensuring full compatibility.[1]

After the development section, the last one will summarize the results and conclusions obtained.

[1] PyInstaller contributors. (2020, February 14). PyInstaller. Retrieved from https://www.pyinstaller.org/

2 State of the art

2.1 Random number generation

Random number generators can be hardware random-number generators (HRNG), which generate true random numbers, or pseudo-random number generators (PRNG), which generate numbers that look random, but are deterministic.

Since the generation is done programmatically, this section focuses on the pseudo-random number generators. These are not truly random, because are completely determined by an initial value, called the PRNG's seed.

The standard algorithms used during the second part of the 20th century for PRNGs comprised linear congruential generators (LCG). The linear congruential method[2] generates the desired sequence of random numbers X_n by setting

$$X_{n+1} = (aX_n + c) \, mod \, m \, , n \geq 0$$

where:

• m	is the modulus;	$0 < m$
• a	is the multiplier;	$0 < a < m$
• c	is the increment;	$0 \leq c < m$
• X_0	is the starting value;	$0 \leq X_0 < m$

Nevertheless, the quality of LCGs was inadequate, but better methods were unavailable. These algorithms exhibited artifacts like:

- The periods for some seed states were shorter than expected.
- For generations with many values, there was a lack of uniformity of distribution.
- Correlation of successive values.

The progress came with the introduction of techniques based on linear recurrences. The Mersenne Twister PRNG was published in 1998 for generating uniform pseudorandom numbers. For a choice of parameters, the algorithm provides a super astronomical period of $2^{19937} - 1$. At that time, it passed several stringent statistical tests, and its speed was comparable to other modern generators. Its merits were due to the efficient algorithms that are unique to polynomial calculations over the two-element field.[3]

This was the default generator in the Python language starting from version 2.3.

[2] Knuth, D. (1997). The Linear Congruential Method. In Seminumerical Algorithms. The Art of Computer Programming (3 ed., p. 10).

[3] Matsumoto, M., & Nishimura, T. (1998). Mersenne Twister. ACM Transactions on Modeling and Computer Simulation, 8, 3-30.

Later in 2003, George Marsaglia introduced the family of xorshift generators based on a linear recurrence. A xorshift random number generator produces a sequence of 232−1 integers x, or a sequence of 264−1 pairs x, y, or a sequence of 296−1 triples x, y, z, etc., by means of repeated use of a simple computer construction: exclusive-or (xor) a computer word with a shifted version of itself.

Combining such xorshift operations for various shifts and arguments provides extremely fast and simple PRNGs that seem to do very well on tests of randomness. To give an idea of the power and effectiveness of xorshift operations, a C procedure which implements this RNG can be very fast, over 200 million/second, and the resulting random integers pass all the tests of randomness that have been applied to them.[4]

In 2006, the WELL family of generators was developed. In some ways improves on the quality of the Mersenne Twister which has a too-large state space and a very slow recovery from state spaces with many zeros.[5]

The last great change is the permuted congruential generator (PCG), which is a PRNG algorithm developed in 2014. It is both extremely practical and statically good. It has several important properties, including solid mathematical foundations, good time and space performance…

The key idea is to pass the output of a fast well-understood "medium quality" random number generator to an efficient permutation function (aka hash function) that enhances the quality of the output. The algorithm can be applied at variety of bit sizes, including 64 and 128 bits which provide periods of 2^{64} and 2^{128}.[6]

A PCG differs from a classical LCG in three ways[7]:

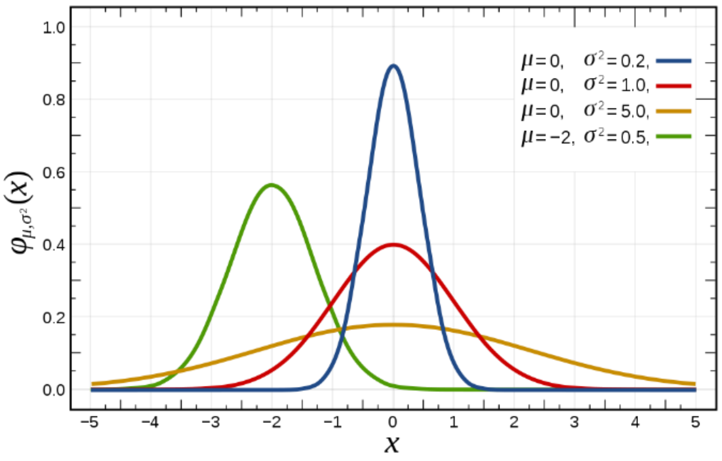
- The LCG modulus and state are larger, usually twice the size of the desired output.
- It uses a power-of-2 modulus, which results in a particularly efficient implementation with a full period generator and unbiased output bits.
- The state is not output directly, but rather the most significant bits of the state are used to select a bitwise rotation or shift which is applied to the state to produce the output.

[4] Marsaglia, G. (2003). Xorshift RNG's. Journal of Statistical Software.
[5] Panneton, F., L'Ecuyer, P., & Matsumoto, M. (2006, March). Improved Long-Period Generators Based on Linear Recurrences. ACM Transactions on Mathematical Software, 32, 1-16.
[6] O'Neill, M. E. (2014). PCG: A Family of Simple Fast Space-Efficient Statistically.
[7] Wikipedia contributors. (2020, June 12). Permuted Congruential Generator. Retrieved from Wikipedia, The Free Encyclopedia: https://en.wikipedia.org/wiki/Permuted_congruential_generator

2.2 Generating random normal samples

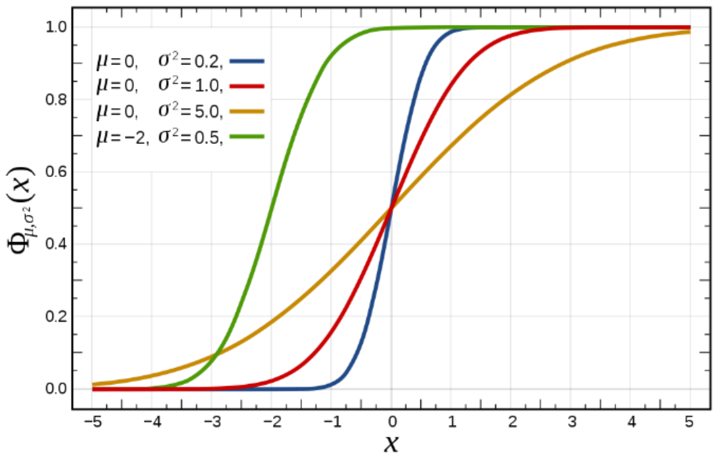
The first goal of this project is about generating random samples with a specific mean and a standard deviation. For this purpose, the normal distribution is going to be used.

It is necessary to state that this distribution $N(\mu, \sigma^2)$ parameterized by its mean (μ) and standard deviation (σ) has a probability density function ($f_N(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$) and a cumulative density function ($F_N(x) = \int_{-\infty}^{x} f_N(t)dt$)



[8]

Figure 1: Probability Density Function (PDF)



[9]

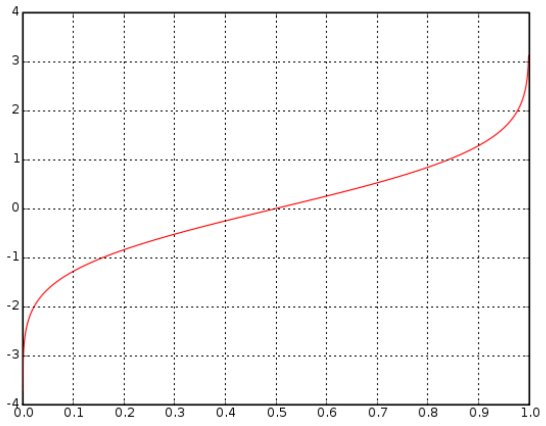
Figure 2: Cumulative Distribution Function (CDF)

[8] Wikimedia Commons contributors. (2018, June 17). Normal Distribution PDF. Retrieved from Wikimedia Commons, the free media repository: https://commons.wikimedia.org/w/index.php?title=File:Normal_Distribution_PDF.svg&oldid=306777443

[9] Wikimedia Commons contributors. (2017, October 15). Normal Distribution CDF. Retrieved from Wikimedia Commons, the free media repository: https://commons.wikimedia.org/w/index.php?title=File:Normal_Distribution_CDF.svg&oldid=263015008

The algorithms commented below are ones of many other that generate normally distributed values. This is possible since a $N(\mu, \sigma^2)$ can be generated as $x = \mu + \sigma Z$, where Z is standard normal. All these algorithms rely on the availability of a random number generator U capable of producing uniform random variates.

The most straightforward method is based on the probability integral transform property: if U is distributed uniformly on $(0,1)$, then $\phi^{-1}(U)$ will have the standard normal distribution. The drawback of this method is that it relies on calculation of the probit function ϕ^{-1}, which cannot be done analytically. For a better understanding, ϕ^{-1} would be the inverse of the cumulative density function (CDF)[10]:



11

Figure 3: Probit function

The central limit theorem establishes that, in some situations, when independent random variables are added, their properly normalized sum tends toward a normal distribution.

If $U = X_1, X_2, \ldots, X_n$ is a random sample of size n taken from a population (either finite or infinite) with mean μ and finite variance σ^2 and if $\bar{X} = \frac{X_1 + X_2 + \cdots + X_n}{n}$ is the sample mean, the limiting form of the distribution of

$$Z \approx \frac{\bar{X} - \mu}{\frac{\sigma}{\sqrt{n}}} = \frac{\sqrt{n}(\bar{X} - \mu)}{\sigma}$$

as $n \to \infty$ is the standard normal distribution.[12]

[10] Wikipedia contributors. (2020, May 29). Wikipedia, The Free Encyclopedia. Retrieved from https://en.wikipedia.org/w/index.php?title=Normal_distribution&oldid=959529084

[11] Wikimedia Commons contributors. (2017, November 26). Probit plot. Retrieved from Wikimedia Commons, the free media repository: https://commons.wikimedia.org/w/index.php?title=File:Probit_plot.png&oldid=269252925

[12] C. Montgomery, D., & C. Runger, G. (2014). Applied Statistics and Probability for Engineers (6 ed.). Wiley.

The Box–Muller transform allows to sample a pair of normally distributed variables using a source of uniformly distributed variables.

Let U_1, U_2 be independent random variables from the same rectangular density function on the interval $(0,1)$. Consider the random variables:

$$X = \sqrt{-2\, ln\, U_1}\, cos(2\pi U_2)$$

$$Y = \sqrt{-2\, ln\, U_1}\, sin(2\pi U_2)$$

Then (X, Y) will be a pair of independent random variables from the same normal distribution with mean zero, and unit variance.[13]

The Marsaglia polar method is a modification of the Box–Muller method which can be improved quite a bit by generating X, Y according to:

$$X = U_1 \sqrt{\frac{-2\, ln\, (U_1^2 + U_2^2)}{(U_1^2 + U_2^2)}}$$

$$Y = U_2 \sqrt{\frac{-2\, ln\, (U_1^2 + U_2^2)}{(U_1^2 + U_2^2)}}$$

Where U_1, U_2 are uniform on $[-1,1]$, conditioned by $U_1^2 + U_2^2 < 1$. Again, X and Y are independent, standard normal random variables.[14]

[13] Box, G., & E. Muller, M. (1958). A Note on the Generation of Random Normal Deviates

[14] Marsaglia, G., & Bray, T. (1964). A Convenient Method for Generating Normal Variables. SIAM Review, 6(3), 260-264.

The Ratio method[15] is a rejection method. To generate a random variable X with density $f(x)$ (PDF), let $C_f = \{(u,v): 0 \leq u \leq f^{1/2}(v\,/\,u)\}$. Then if (U,V) is a pair of random variables distributed uniformly over C_f, $x = V\,/\,U$ has the desired density f. The joint density of $(X,Y) = T(U,V)$ is

$$g(x,y) = \begin{cases} 2y, 0 \leq y \leq f^{1/2}(x) \\ 0, otherwise. \end{cases}$$

and hence the marginal density of X is:

$$\int_0^{f^{1/2}} 2y\,dy = f(x)$$

The ziggurat algorithm[16] is computationally much faster than the Marsaglia polar method and the Box–Muller transform. The gaussian distribution that the ziggurat algorithm chooses from is made up of n equal-area regions; $n-1$ rectangles that cover the bulk of the distribution, on top of a non-rectangular base that includes the tail of the distribution. This can be seen on Figure 4.

Given a monotone decreasing probability density function $f(x)$, defined for all $x \geq 0$, the base of the ziggurat is defined as all points inside the distribution and below $y_1 = f(x_1)$. This consists of a rectangular region from $(0,0)$ to (x_1, y_1), and the (typically infinite) tail of the distribution, where $x > x_1$ (and $y < y_1$).

This layer (call it layer 0) has area A. On top of this, add a rectangular layer of width x_1 and height $A\,/\,x_1$, so it also has area A. The top of this layer is at height $y_2 = y_1 + A\,/\,x_1$, and intersects the density function at a point (x_2, y_2), where $y_2 = f(x_2)$. This layer includes every point in the density function between y_1 and y_2, but (unlike the base layer) also includes points such as (x_1, y_2), which are not in the desired distribution.

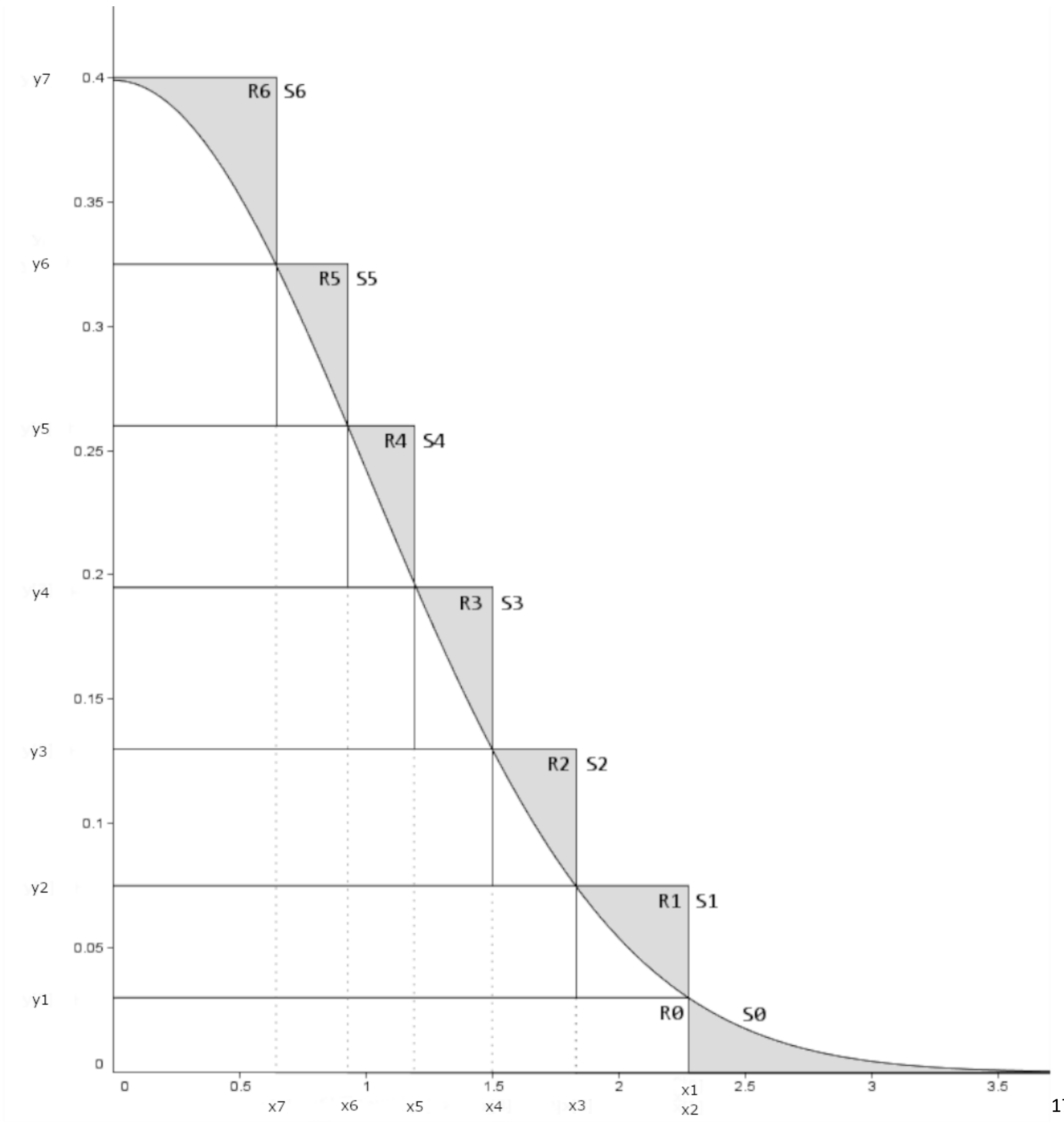
Further layers are then stacked on top. To use a precomputed table of size n, one chooses x_1 such that $x_n = 0$, meaning that the top box, layer $n-1$, reaches the distribution's peak at $(0, f(0))$ exactly.

Ignoring for a moment the problem of layer 0, and given uniform random variables U_1 and $U_2 \in [0,1)$, the ziggurat algorithm for a normal distribution can be described as:

- Let $x = -\,ln(U_1)\,/\,x_1$
- Let $y = -\,ln(U_2)$
- If $2y > x^2$, return $x + x_1$
- Otherwise, go back to step 1

[15] Kinderman, A., & Monahan, J. (1977). Computer Generation of Random Variables Using the Ration of Uniform Deviates. In ACM Transactions on Mathematical Software (pp. 257-260)

[16] Wikipedia contributors. (2020, May 2013). Wikipedia, The Free Encyclopedia. Retrieved from https://en.wikipedia.org/w/index.php?title=Ziggurat_algorithm&oldid=956443636



[17]

Figure 4: Gaussian distribution divided in rectangles by Ziggurat algorithm

[17] Colin. (2011, September). The Ziggurat Algorithm for Random Gaussian Sampling. Retrieved from Heliosphan: https://heliosphan.org/zigguratalgorithm/zigguratalgorithm.html

2.3 Fourier analysis

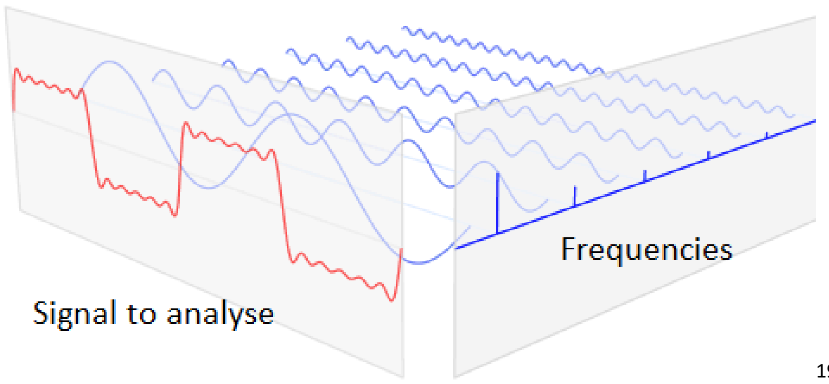
A Fourier series can be created only with the fundamental frequency, its amplitude, and the amplitudes of its harmonics. A discrete sum is sufficient:

$$F(t) = \sum_{n=-\infty}^{\infty} a_n \cdot cos(2\pi_{nv_0} t) + b_n \cdot sin(2\pi n v_0 t)$$

where v_0 is the fundamental frequency. Sines as well as cosines are required because the harmonics are not necessarily in phase with the fundamental or with each other. Furthermore, the sum is taken from $-\infty$ to ∞ for the sake of mathematical symmetry.

This process of building a waveform by adding together a fundamental frequency and overtones or harmonics of various amplitudes is called Fourier synthesis. [18]

The alternative process – of extracting from the signal the various frequencies and amplitudes that are present – is called Fourier analysis and is much more important in its practical and physical applications.



[19]

Figure 5: Fourier analysis of a signal

Whether $F(t)$ is periodic or not, a complete description of $F(t)$ can be given using sines and cosines. So, it can be transformed. If $F(t)$ is not periodic, it requires all frequencies to be present if it is to be synthesized. A non-periodic function can be considered as a limiting case of a periodic one, where the period tends to infinity and, consequently, the fundamental frequency tends to zero.

This transformation can be done thanks to the discrete Fourier transform, which states that given two sets of numbers $[a_n]$ and $[A_m]$, each set having N elements, can be mutually transformed:

$$A(m) = \frac{1}{N} \sum_0^{N-1} a(n) \cdot e^{2\pi i \frac{nm}{N}}; \;\; a(n) = \sum_0^{N-1} A(m) \cdot e^{-2\pi i \frac{nm}{N}}$$

[18] James, J. (2011). Fourier Series. In J. James, A Student's Guide to Fourier Transforms (3 ed., pp. 2-3). Cambridge.
[19] Pgfplots contributors. (n.d.). Fourier Transform. Retrieved from Pgfplots: http://pgfplots.net/media/tikz/examples/PNG/fourier-transform.png

Back in the days, many ingenious devices were invented for performing Fourier transforms, mechanically, electrically, acoustically, and optically. These are now part of history since the invention of the Fast Fourier Transform algorithm (FFT)[20], which uses symmetries in the calculated terms. The symmetry is highest when the size of the sample is a power of 2, and the transform is therefore most efficient for these sizes.

The fast Fourier transform is a computationally efficient algorithm for computing the DFT. It requires fewer multiplications than a more straightforward programming implementation of the DFT and its relative advantage in this respect increases with the length of the sample sequences involved.

The FFT makes use of the periodic nature of twiddle factors and of symmetries in the structure of the DFT expression. Various, slightly different, versions of the FFT can be derived from the DFT. For example:

- Decimation-in-time (DIT: radix-2 and radix-4)
- Decimation-in-frequency (DIF)

These versions of the FFT differ in the exact form of the intermediate computations that make them up. However, ignoring rounding errors, they both produce the same results. [21]

Along the different FFT algorithms that exist, the Cooley-Tukey algorithm is the most used one. This is a divide and conquer algorithm that recursively breaks down a DFT of any composite size $N = N_1 N_2$ into many smaller DFTs of sizes N_1 and N_2, along with $O(N)$ multiplications by complex roots of unity traditionally called twiddle factors.

A radix-2 decimation-in-time (DIT) FFT is the simplest and most common form of the Cooley-Tukey algorithm, although highly optimized Cooley-Tukey implementations typically use other forms of the algorithm as described below.[22]

[20] James, J. (2011). Discrete and digital Fourier transforms. In J. F. James, A Student's Guide to Fourier Transforms (3 ed., pp. 127-129). Cambridge.
[21] Reay, D. (2012). Fast Fourier Transform. In D. Reay, Digital Signal Processing with the OMAP-L138 eXperimenter (pp. 212-223). New Jersey: John Wiley & Sons, Inc.
[22] Wikipedia contributors. (2020, July 9). Cooley-Tukey FFT algorithm. Retrieved from Wikipedia, The Free Encyclopedia: https://en.wikipedia.org/wiki/Cooley%E2%80%93Tukey_FFT_algorithm

2.4 Partitional clustering algorithms: k-Means

A partitional clustering algorithm receives as input a data set and divide the different instances in partitions or groups regarding to the similarity between the instances.

The k-Means is probably the most famous partitional clustering algorithm, which divide the different instances in k partitions or groups. This k must be known a priori. The standard algorithm (the most used) is referred as Lloyd's algorithm or as "naïve k-means" and the different steps are shown below[23]:

1. Initialization. Set K means $\{m^{(k)}\}$ to random values
2. Assignment. Each data instance n is assigned to the nearest mean. We denote our guess for the cluster $k^{(n)}$ that the point $x^{(n)}$ belongs to by $\hat{k}^{(n)}$.

   $$\hat{k}^{(n)} = argmin\ \{d(m^{(k)}, x^{(n)})\}$$

   If tie happens, $\hat{k}^{(n)}$ is set to the smallest of the winning $\{k\}$
3. Update step. The model parameters, the means, are adjusted to match the sample means of the data points that they are responsible for:

   $$m^{(k)} = \frac{\sum_n r_k^{(n)} x^{(n)}}{R^{(k)}}$$

   Where $R^{(k)}$ is the total responsibility of mean k,

   $$R^{(k)} = \sum_n r_k^{(n)}$$

   If $R^{(k)} = 0$, then we leave the mean $m^{(k)}$ where it is.
4. Repeat the assignment step and update step until the assignments do not change.

This algorithm is acceptable when clusters are compact and well separated. Nevertheless, it performs poorly to find subsets within other bigger sets. It is sensitive to scale, noise, and outliers.

Although this is the standard version, it has many others that usually change the initialization or update steps. Some of them are shown next.

[23] MacKay, D. (2003). Chapter 20. An example Inference Task: Clustering. In D. MacKay, Information Theory, Inference and Learning Algorithms (pp. 284-292). Cambridge University Press.

2.4.1 First variation: k-medians

The k-medoids approach is a first attempt to get more robust clustering algorithms. It consists in considering criteria based on least norms instead of least squared norms, so that the cluster centres are the spatial medians, also called geometric or L_1-medians, of the elements belonging to each cluster.[24]

2.4.2 Second variation: Partitioning Around Medoids

The k-medoids or partitioning around medoids (PAM) algorithm searches for k "representative" objects, called medoids, which minimize the average dissimilarity of all objects of the data set to the nearest medoid.

In this model the representative object of a cluster is the object for which the average dissimilarity to all the objects of the cluster is minimal. This is the medoid of the cluster and because of this the model is called the k-medoid model.

The difference with the k-means is that in this method, each cluster is characterized by a centrally located object (given in the data set), instead of a centroid.[25]

It works as follows:

1. Initialize: greedily select k of the n data points as the medoids to minimize the cost.
2. Associate each data point to the closest medoid.
3. While the cost of the configuration decreases:
    1. For each medoid m, and for each non-medoid data point o:
        1. Consider the swap of m and o, and compute the cost change.
        2. If the cost change is the current best, remember this m and o combination.
    2. Perform the best swap m_{best} and o_{best}, if it decreases the cost function. Otherwise, the algorithm execution finishes.

It is more robust to noise and outliers as compared to k-means because it minimizes a sum of pairwise dissimilarities instead of a sum of squared Euclidean distances.[26]

[24] Cardot, H., Peggy, C., & Jean-Marie, M. (June de 2012). A fast and recursive algorithm for clustering large datasets with k-medians. Computational Statistics & Data Analysis, 56(6), 1434-1449.
[25] Rousseeuw, P., & Kaufman, L. (1987). Clustering by means of Medoids. In Statistical Data Analysis Based on the L1-Norm and Related Methods (pp. 405-416). Brussels.
[26] Wikipedia contributors. (2020, June 12). k-medoids. Retrieved from Wikipedia, The Free Encyclopedia: https://en.wikipedia.org/wiki/K-medoids

2.4.3 Third variation: Fuzzy C-Means Clustering (FCM)

The FCM algorithm is one of the most widely used fuzzy clustering algorithms. In non-fuzzy or hard clustering algorithms as k-means, data is divided into crisp clusters, where each data point belongs to exactly one cluster. In fuzzy clustering or soft clustering, the data points can belong to more than one cluster, and associated with each of the points are membership grades which indicate the degree to which the data points belong to the different clusters.[27]

It attempts to partition a finite collection of elements $X = \{x_1, x_2, \dots, x_n\}$ into a collection of c fuzzy clusters with respect to some given criterion.

Given a finite set of data, the algorithm returns a list of c clusters $C = \{c_1, c_2, \dots, c_c\}$ and a partition matrix $W = w_{ij} \in [0,1], i = 1, \dots, n,\ j = 1, \dots, c$ where w_{ij} tells the degree to which the element x_i belongs to cluster c_j.

The FCM aims to minimize:

$$\underset{C}{argmin} \sum_{i=1}^{n} \sum_{j=1}^{c} w_{ij}^{m} \left\|x_i - c_j\right\|^2,$$

where:

$$w_{ij} = \frac{1}{\sum_{k-1}^{c} \left(\frac{\left\|x_i - c_j\right\|}{\left\|x_i - c_k\right\|}\right)^{\frac{2}{m-1}}}$$

Although it differs from the k-means objective function by the addition of membership values and the fuzzifier, it has the same problem as k-means:

- The minimum is a local minimum.

Furthermore, the results depend on the initial choice of weights.[28]

[27] Wolfram developers. (2004). Fuzzy Logic. Retrieved from Wolfram: https://reference.wolfram.com/legacy/applications/fuzzylogic/Manual/12.html
[28] Wikipedia contributors. (2020, March 8). Fuzzy Clustering. Retrieved from Wikipedia, The Free Encyclopedia: https://en.wikipedia.org/wiki/Fuzzy_clustering#Algorithm

3 Development

As it is mentioned in the introduction, the development is divided in two different parts. Both will design and develop a ready-to-use application following an iterative and incremental methodology.

3.1 Random time series generator

The first goal of this thesis would be to build a software tool for randomly generating a set of time series. The tool must allow to specify the desired time series behaviour by setting different parameters. The output file would be a CSV containing the time series set, stored either by rows or by columns.

This section describes the process of development of the random time series generator.

3.1.1 First Increment: Generate random samples

The main goal in this early step would be to build a self-contained application for Windows. The software tool will not require network connections or external dependencies to meet its functional requirements.

Given some parameters, a "comma-separated values" file (CSV) is created with the sequences generated. These sequences would contain random values with a specific mean and a standard deviation.

3.1.1.1 Requirements

Functional requirements

- The application would receive the following parameters from the user.
  - n_v = Number of values for each generated time series
  - n_t = Number of generated time series
  - μ_T = Mean of the time series' different means
  - σ_μ = Standard deviation of the means
  - σ_t = Standard deviation of each time series

  The output would be n_t time series. The different means of the time series $(\mu_{t1}, \mu_{t2}, \dots, \mu_{tn_t})$ would have a μ_T mean and a σ_μ standard deviation. For each time series and its different mean, there would be n_v random values with a μ_{tx} mean and a σ_t standard deviation.
- Giving the location and the file name as parameters, the user would save the results in a CSV file.

Non-functional requirements

- The application should be suitable to any type of computer running a Windows operating system.
- The application's language must be English.

3.1.1.2 Analysis and design

3.1.1.2.1 Graphical User Interface

Tkinter is the tool selected to build a graphical user interface. It is the standard Python interface to the Tk GUI toolkit. Both Tk and Tkinter are available on most Unix platforms, as well as on Windows systems.[29]

It is necessary to state that Tkinter is a library which provides a Python functional interface to a graphic library based on Tcl/Tk. Tcl is a programming language and Tk is a widget toolkit and both are free and open source. Nowadays, Tkinter is Python's de facto standard GUI.

For this increment it would be necessary to create a window with the following widgets:

- Labels, which displays text or images, typically that the user will just view but not otherwise interact with. Labels are used for different purposes as identifying controls or other parts of the user interface, providing textual feedback or results…[30]
- Entries, which presents the user with a single line text field that they can use to type in a string value.[31]
- Button, which is very much designed for the user to interact with, and press to perform some action. Like labels, they can display text or images, but also have a whole range of new options used to control their behaviour.[32]

3.1.1.2.2 Generating random numbers with a specific mean and a standard deviation

NumPy is the package selected to generate the random number. The chosen function has two parts:

- For the random number generation, it uses a 128-bit implementation of O'Neill's permutation congruential generator. The specific member of the PCG family that they use is PCG XSL RR 128/64.[33]

[29] Tkinter contributors. (2020, June 04). Tkinter. Retrieved from python.org: https://docs.python.org/3/library/tkinter.html
[30] Roseman, M. (2019). Label. In M. Roseman, Modern tkinter for busy developers (2 ed., p. 28). Late Afternoon Press.
[31] Roseman, M. (2019). Entry. En M. Roseman, Modern tkinter for busy developers (2 ed., pág. 34). Late Afternoon Press.
[32] Roseman, M. (2019). Button. In M. Roseman, Modern tkinter for busy developers (2 ed., p. 31). Late Afternoon Press.
[33] Numpy contributors. (2020, June 4). Permuted Congruential Generator. Retrieved from Numpy.org: https://numpy.org/devdocs/reference/random/bit_generators/pcg64.html#numpy.random.PCG64

- Once the random numbers are generated, a normal generator use 256-step Ziggurat methods that are 2-10 times faster than NumPy's Box-Muller or inverse CDF implementations.[34]

The results given by this function are close to the desired mean and standard deviation. When generating big samples, there is a minor error. However, to achieve the exact standard deviation and mean, these transformations over the data ($samples$) are needed:

- Subtract the sample mean from every number in the sample.

$$zero\_mean\_samples = samples - mean(samples)$$

- Scale the samples so that the standard deviation is precise

$$scaled\_samples = zero\_mean\_samples * \frac{desired\_std}{std(zero\_mean\_samples)}$$

- Add the desired mean.

$$final\_samples = scaled\_samples + mean$$

3.1.1.3 Implementation

The validation of the entries given by the user play an important role during the development. Once the run button is triggered, the different parameters are validated. For example, to validate the integers this function is used:

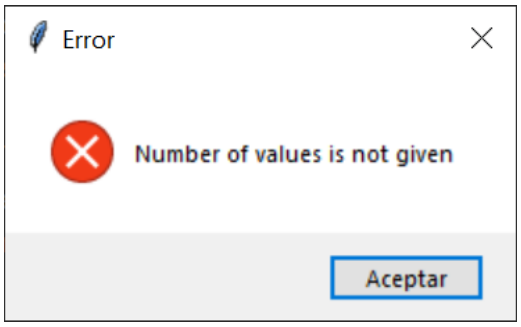
```
def check_integer(entry_widget, error_text, parameter_name):
    # Getting the value from the widget
    value = entry_widget.get()
    new_value = value

    # Checking if the value is not None (null)
    if value:
        # Using a function to test if it is an integer
        if not is_positive_unsigned_integer(value):
            error_text = error_text + parameter_name + " is not an integer" +
"\n"
        else:
            new_value = int(value)

    else:
        error_text = error_text + parameter_name + " is not given" + "\n"
    return new_value, error_text
```

[34] Numpy contributors. (2020, June 4). Random sampling. What's new or different? Retrieved from Numpy.org: https://numpy.org/devdocs/reference/random/index.html#what-s-new-or-different

If some of the entries contain a value which is not correct, a message box arises in the application with the $error\_text$ information. Besides, program execution is cancelled. A message box widget provides several functions that you can use to display an appropriate message.

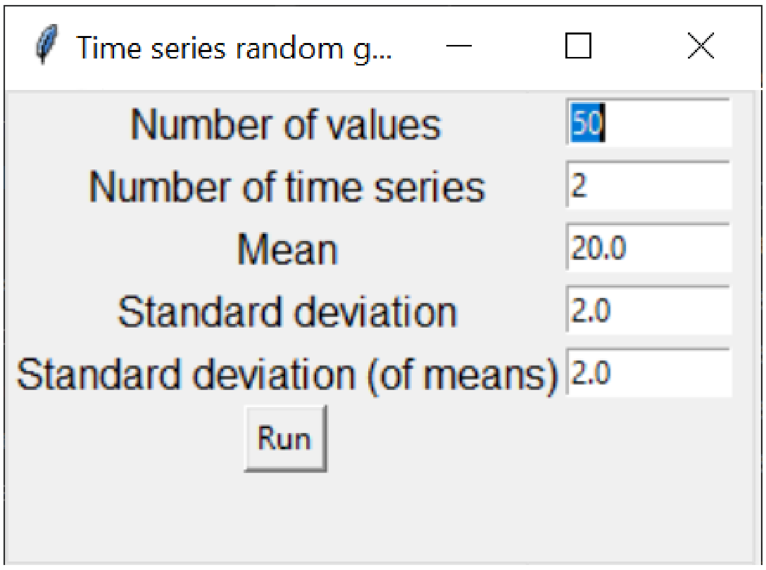

Error

Number of values is not given

Figure 6: Message box when the number of values is not given

3.1.1.4 Application interface

The final application interface is shown in Figure 7. The testing will be done by the supervisors, and the parts that would need to be changed, dropped, or added will be reviewed during the validation process.


Time series random g...

Number of values	50
Number of time series	2
Mean	20.0
Standard deviation	2.0
Standard deviation (of means)	2.0

Figure 7: First increment result of the application

3.1.1.5 Test

In order to check the time series generator, a unit testing is implemented into the code. This is done through unittest library, which allows individual units of source code to be tested and determines whether they are fit for use.

The test will check the sequence generator. It only accepts an error whose value is less than 10^{-5}. The code can be seen below:

```python
def test_generate_time_series(self):
    samples = generate_time_series(5.0, 3.0, 20, only_seed=True)
    self.assertEqual(5.0, round(np.mean(samples), 5))
    self.assertEqual(3.0, round(np.std(samples), 5))
    self.assertEqual(20, len(samples))
```

With the parameter configuration shown in Figure 7, the resulted time series is shown in Figure 8:

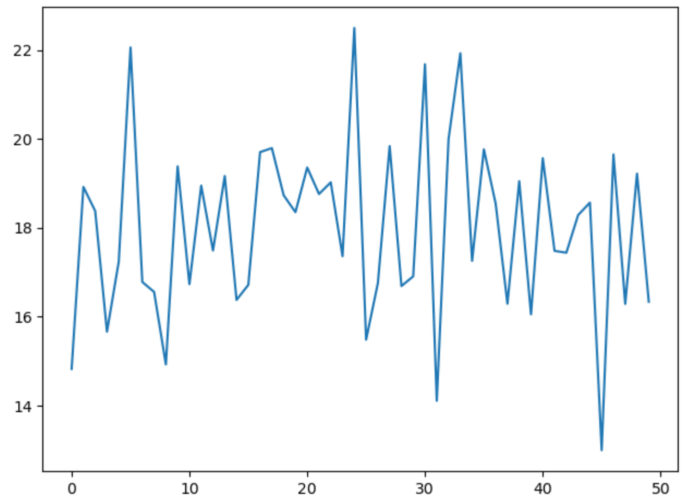


Figure 8: Execution result of the first increment.

3.1.1.6 Activity diagram

Once the button run is clicked, the process triggered is represented by the activity diagram shown in Figure 9.

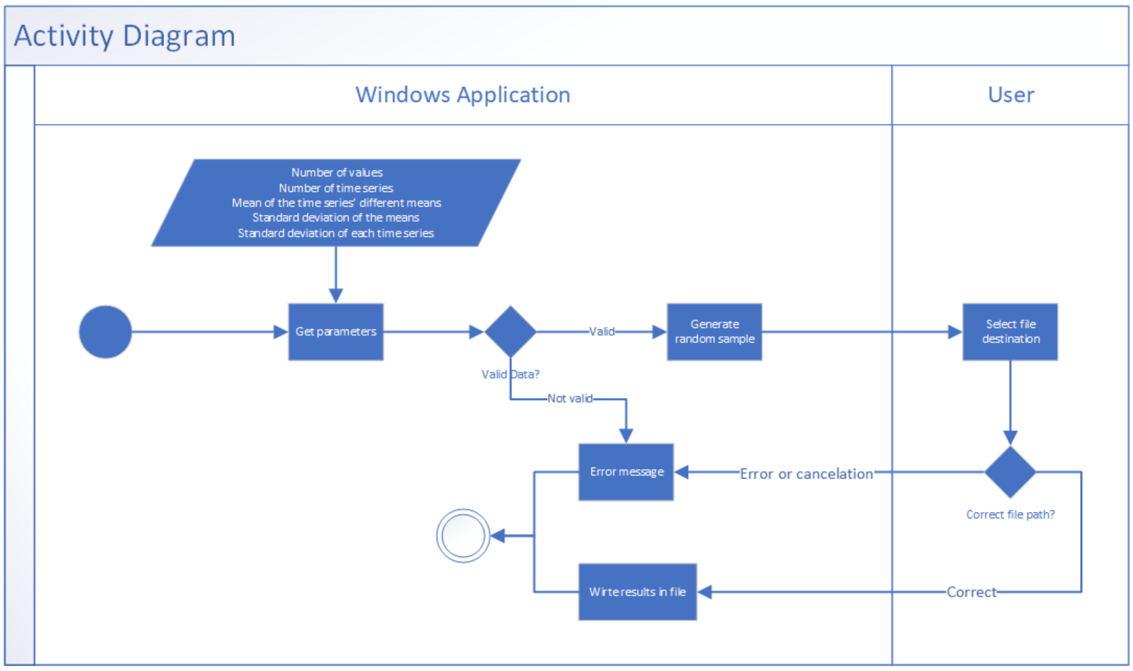


Figure 9: Activity diagram - First increment

3.1.1.7 Validation

The general purpose of the increment is validated. Nevertheless, there are some things related with this functionality that must be improved:

The user should be able to select some features of the CSV file generated:

- CSV separator between values
- Decimal separator of the values
- Statistics for each generated time series (if selected)
- Boolean parameter that specifies if the time series are displayed in rows or in columns.

3.1.2 Second Increment: Trend

There are two main objectives in which this second increment is focused on. The first one would be to cover the different CSV properties proposed by the supervisors in the last increment validation.

Secondly, it is mandatory to offer the possibility of adding an increasing or decreasing trend to the random time series.

3.1.2.1 Requirements

Functional requirements

- There are two inputs for generating the trend:
    - $trend_t$ = Type of the trend. It can be increasing, decreasing or none.
    - $trend_i$ = Intensity of the trend. It would be a float number which represents the percentage over the mean of the sequence.

  Given these parameters, the trend is generated and added to the random time series from the first increment.

- For the CSV part, the user could change the following settings:
    - rc = Boolean that specifies if the series is displayed in rows or in columns.
    - csv_s = Type of CSV separator. The options are the comma, the semicolon, and the vertical slash.
    - csv_{ds} = Type of decimal separator. The options are the comma or the dot.
    - $stats$ = Boolean that specifies if statistics about the generated time series are also shown at the end of the CSV file.

  The random time series generator outputs a CSV file containing all the time series. Depending on the rc Boolean value, the generated time series would be saved as new rows or columns of the file.

  Furthermore, the CSV file would have csv_s as a separator between values, csv_{ds} as a decimal separator of the values and, if stats Boolean is true, then it would have information about the mean and the standard deviation at the end of each time series.

Non-functional requirements

- The settings part must be in a different window than the main part.

3.1.2.2 Analysis and design

3.1.2.2.1 Graphical User Interface

To separate the CSV settings from the main part of the application, a Tkinter widget called notebook is going to be used. This widget manages a collection of windows and displays one at a time. Each child window is associated with a tab, which the user may select to change the currently displayed window.

Due to this new implementation, there two main tabs:

- Main: contains the parameters defined on the first increment
- Settings: contains more tabs to be used. Although there are only a bunch of parameters right now, new ones are expected in the next increments.

Inside the Settings tab, there will be another two:

- The first one called CSV with the parameters related to file creation (separator, decimal separator, statistics and displaying mode).
- Secondly, a Trend tab with the parameters to create the trend (trend type and intensity)

For the settings parameters, a new widget would be implemented:

- Radiobutton**,** which lets you choose between several mutually exclusive choices. It is not limited to just two choices and is always used together in a set.[35]

3.1.2.2.2 Generating the trend

The NumPy package is the one selected to generate the trend. There is a function called *linspace* that returns evenly spaced numbers over a specified interval. The function has the following form:

$$numpy.linspace(start, stop, number\_of\_values)$$

So, for each time series x generated, if our trend type parameter $trend_t$=increasing, then:

$$trend = numpy.linspace(0, trend_i, n_v)$$

Else, if our trend type parameter $trend_t$=decreasing, then:

$$trend = numpy.linspace(trend_i, 0, n_v)$$

This trend is added to the first increment final sample.

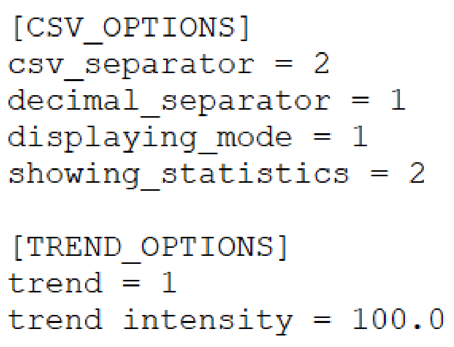
[35] Roseman, M. (2019). Radiobutton. In M. Roseman, Modern tkinter for busy developers (2 ed., p. 34). Late Afternoon Press.

3.1.2.3 Implementation

All the parameters inside the settings can be saved. This is done through an informal standard for configuration files: INI. INI files are text files composed by sections, properties, and values.

Thus, once a save button is clicked on any settings tab, the different parameters are saved in a file named "conf.ini" which is saved in the project root. The package used for managing these files is configparser, which implements a basic configuration language which provides a structure similar to what is found in Microsoft Windows INI files.[36]

The final file looks like Figure 10:



```
[CSV_OPTIONS]
csv_separator = 2
decimal_separator = 1
displaying_mode = 1
showing_statistics = 2

[TREND_OPTIONS]
trend = 1
trend intensity = 100.0
```

Figure 10: File for saving the parameters (conf.ini)

The different parts mentioned before are:

- *Section*: CSV_OPTIONS, TREND_OPTIONS
- *Properties*: csv_separator, decimal_separator, displaying_mode…
- *Values*: 2, 1, 1, 2…

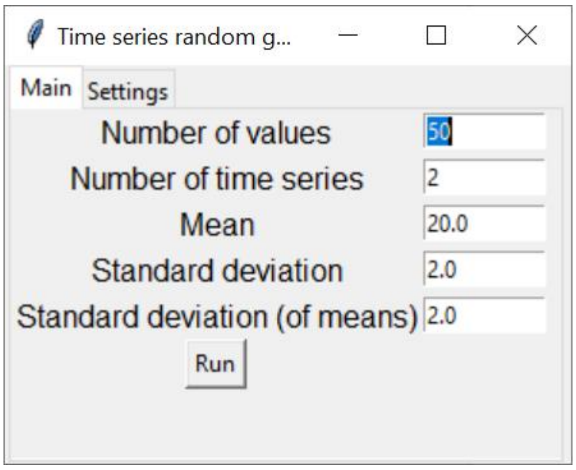
Once the application starts running, it uploads every parameter from the file and update the different widgets. Each time a save button is clicked every parameter is saved into the file.

For this second increment, the main tab parameters are not saved, and default parameters will be shown when the application starts.

[36] Python contributors. (2020). Configparser. Obtenido de Python.org: https://docs.python.org/3/library/configparser.html

3.1.2.4 Application interface

The updated application is represented by Figures 11, 12 and 13. The testing will be done by the supervisors, and in the validation part we will review which parts need to be changed, dropped, or added.



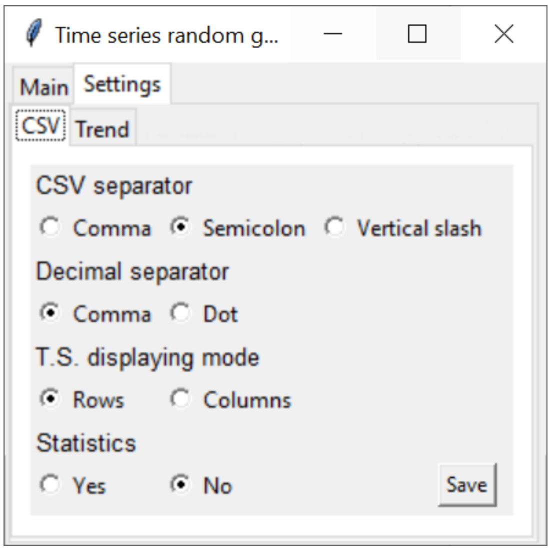
Time series random g...

Main | Settings

Number of values: 50
Number of time series: 2
Mean: 20.0
Standard deviation: 2.0
Standard deviation (of means): 2.0

Run

Figure 11: Main tab of the second increment.



Time series random g...

Main | Settings

CSV | Trend

CSV separator
Comma [✓ Semicolon] Vertical slash

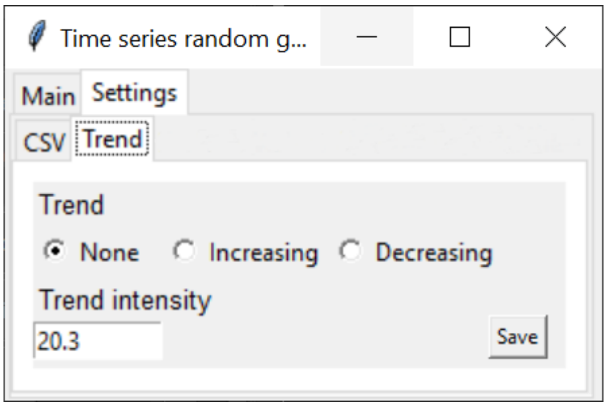
Decimal separator
[✓ Comma] Dot

T.S. displaying mode
[✓ Rows] Columns

Statistics
Yes [✓ No]

Save

Figure 13: Settings-CSV tab of the second increment.



Time series random g...

Main | Settings

CSV | Trend

Trend
[✓ None] Increasing Decreasing

Trend intensity
20.3

Save

Figure 12: Settings-Trend tab of the second increment.

3.1.2.5 Test

A generated time series with the parameters of Figure 11 can be generated with the Trend value as None or Increasing. Taking into account that the Increasing trend has a specified intensity of 20.3, both possibilities are shown in Figure 14.

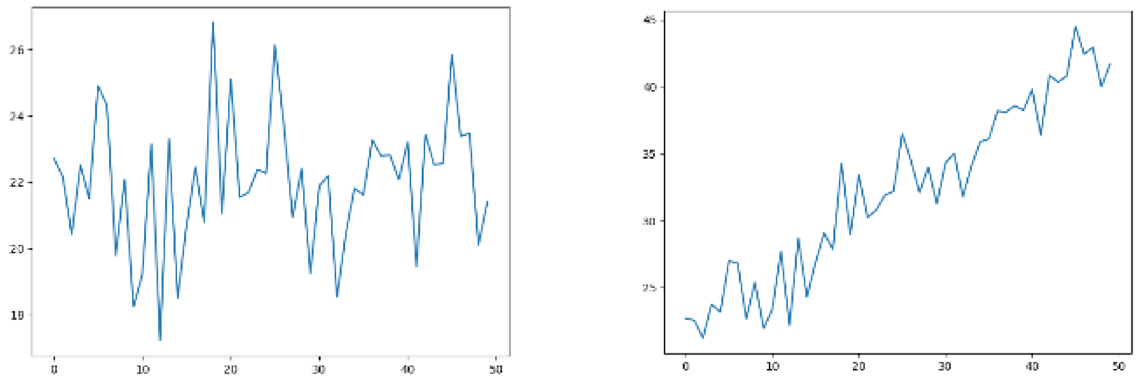


Figure 14: Random sequence with Trend value as None (left) or Increasing (right)

3.1.2.6 Activity diagram

Once the button run is clicked, the process triggered is represented by the activity diagram shown in Figure 15. The light blue nodes and arrows represent the steps developed in this increment.

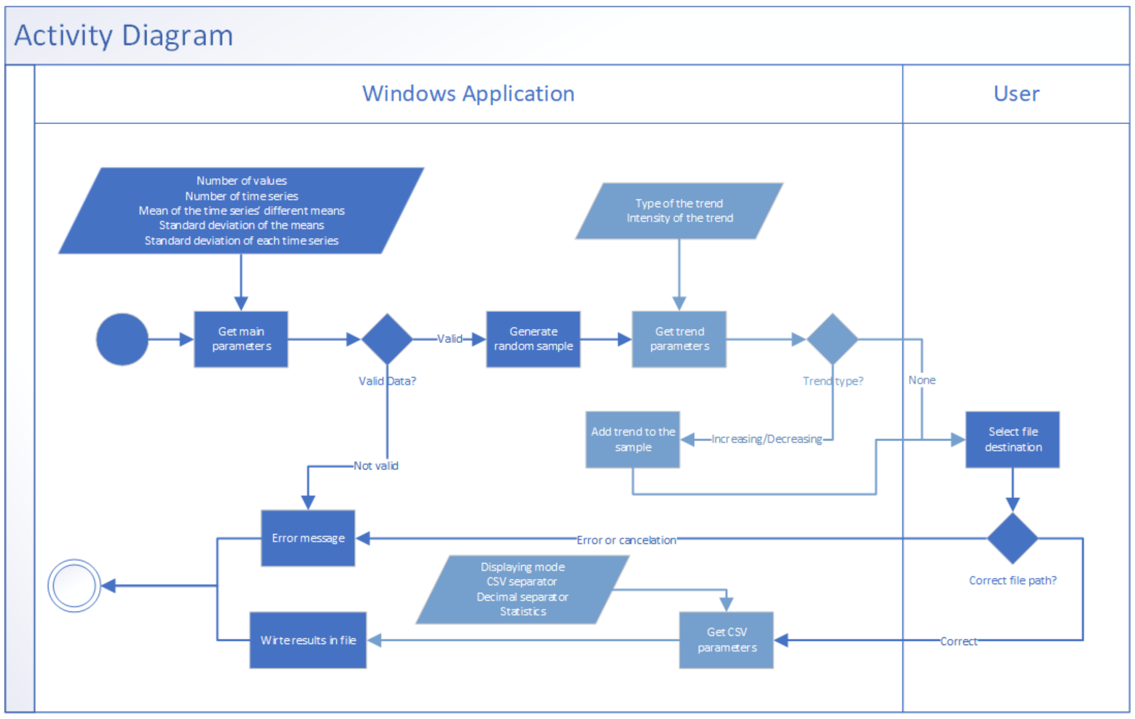


Figure 15: Activity diagram - Second increment

3.1.2.7 Validation

The general purpose of the increment is validated with the supervisors. However, there is a concern about the randomness of the trend generated. Thus, there will be two new parameters for the trend in the next increment:

- $trend_r$= If the trend includes randomness by itself.
- $trend_{sd}$= Standard deviation of the trend.

Besides, a recommendation about how to manage the settings arise during the conversation. Save button tends to be tedious to use, it is easy to forget if you clicked it or not before the values generation. And there is not something that shows if the parameters are or not saved.

3.1.3 Third Increment: Peaks and valleys

In this step there are three different parts. The first and second ones are about adding the different trend properties proposed by the supervisors in the last increment validation, and to implement a different approach of using/saving the parameters.

The third one consists of implementing spontaneous appearances of peaks and valleys during the time series.

3.1.3.1 Requirements

Functional requirements

- There are two new inputs for generating the trend:
    - $trend_r$= Boolean that determines if the trend includes randomness.
    - $trend_{sd}$= Standard deviation of the trend.

  If $trend_r$ is true, a random array of mean zero and standard deviation $trend_{sd}$ is generated and added to the trend.

- For the peak and valley appearances, the user could change the following settings:
    - pv_t = Type of event that can appear in the time series. It can be None, Peak, Valley or Both.
    - pv_m = Magnitude of the peak or valley. It is specified as a percentage over the time series mean.
    - pv_p = Probability of occurrence in percentage.
    - pv_r = Boolean that determines if the peak or valley has recovery.
    - pv_{rt} = Duration of the recovery in time units.
    - pv_{rp} = Position of the peak or valley in the recovery. The positions allowed are Beginning, Middle, End or All of them.

  Given the parameters, if pv_t is not None, for each generated time series tx and for each of its values, there is a probability $pv_p/100$ of occurring a peak or valley. With recovery pv_r being False, $\mu_{tx} * pv_m$ is added ($pv_t = Peak$) or subtracted ($pv_t = Valley$) to/from all the next values of the sequence.

  On the other hand, if pv_r is True, different calculations are done to produce evenly spaced numbers that reach the peak or valley in the different situations given by pv_{rp}.

  For pv_t = Both or pv_{rp} = All, an algorithm draws the different options with the same probabilities.

Non-functional requirements

- Save button for settings tabs should disappear.

3.1.3.2 Analysis and design

3.1.3.2.1 Graphical User Interface

A Peak & Valley tab is added inside the Settings one. The widgets used are already mentioned on the increments before.

Following the supervisors advise, save buttons are deleted, and an automatic checking of the entries is implemented to avoid typing errors.

3.1.3.3 Implementation

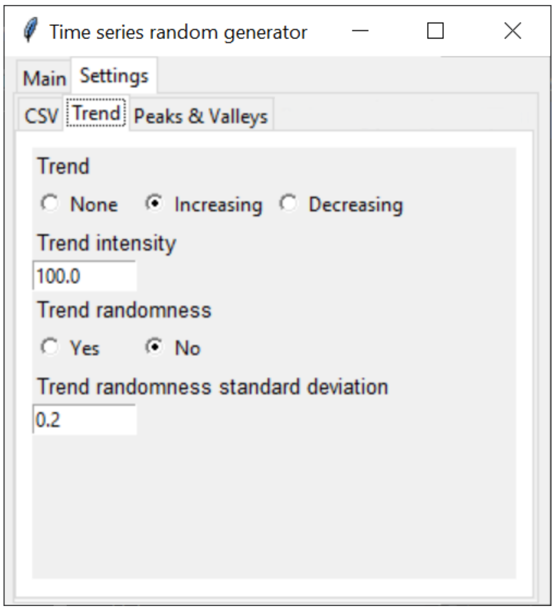
To avoid “Save” buttons, an event listener is implemented in every entry. As soon as a key is pressed, the following method is triggered (depending on if the expected result is an integer or a float, only line 6 would change):

```
1.  def validate_int(value_if_allowed,widget_name):
2.      # We do the cast
3.      try:
4.          int(value_if_allowed) | float(value_if_allowed)
5.          # Checking which widget has done the change and changing the value to
    the casted one.
6.          if "entry_number_of_values" in widget_name:
7.              MainFrame.number_of_values = int(value_if_allowed)
8.          if "entry_number_of_time_series" in widget_name:
9.              MainFrame.number_of_time_series = int(value_if_allowed)
10.         return True
11.     # If the cast can not be done
12.     except ValueError:
13.         # We need to allow empty values
14.         if value_if_allowed == '':
15.             return True
16.         else:
17.             return False
```

Thus, if the new value after pressing any key is not validated, then the entry text would not change. If it changes, the value is automatically saved in the configuration file (conf.ini).

3.1.3.4 Application interface

The updated tabs are the ones shown in Figures 16, 17. The other tabs are unchanged.



Time series random generator

Main | Settings

CSV | Trend | Peaks & Valleys

Trend

None [✓ Increasing] Decreasing

Trend intensity

100.0

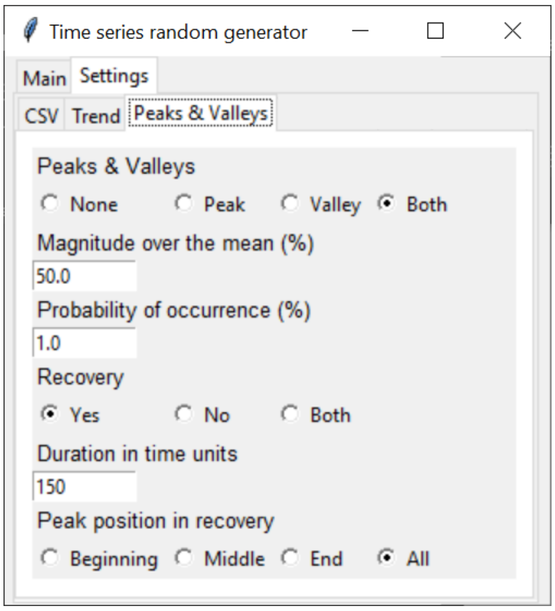
Trend randomness

Yes [✓ No]

Trend randomness standard deviation

0.2

Figure 16: Settings-Trend tab of the third increment



Time series random generator

Main | Settings

CSV | Trend | Peaks & Valleys

Peaks & Valleys

None Peak Valley [✓ Both]

Magnitude over the mean (%)

50.0

Probability of occurrence (%)

1.0

Recovery

[✓ Yes] No Both

Duration in time units

150

Peak position in recovery

Beginning Middle End [✓ All]

Figure 17: Settings-Peaks & Valleys tab of the third increment

3.1.3.5 Test

For this test, the application is run with Figure 11 parameters, without trend but with peaks and valleys specified by the parameters of Figure 17. The results are shown in Figures 18, 19 and 20. The position of the peak or valley (beginning, middle or end) refers to the position of the event in the recovery.

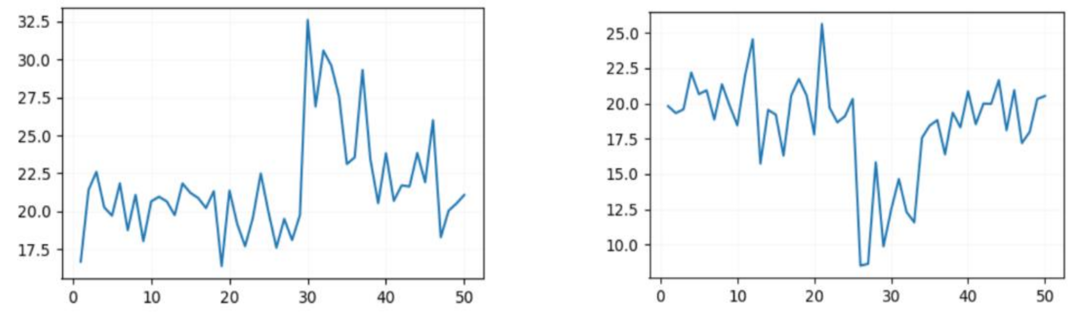


Figure 18: Time series with peak (left) and valley (right) in beginning position of the recovery.

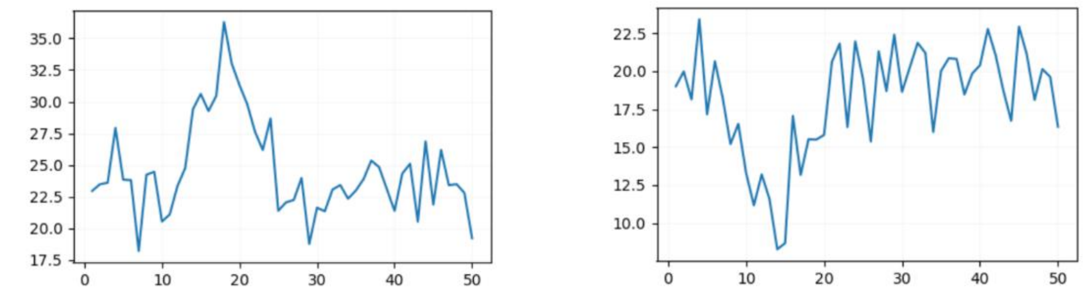


Figure 19: Time series with peak (left) and valley (right) in middle position of the recovery.

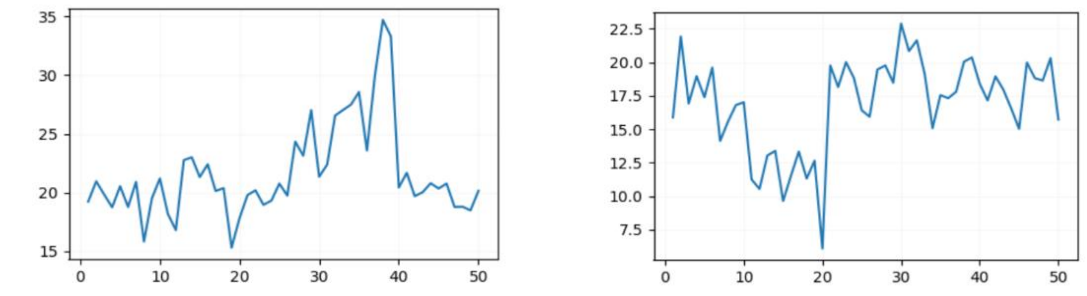


Figure 20: Time series with peak (left) and valley (right) in end position of the recovery.

If there is not recovery, after a peak or a valley the time series values do not return to the same magnitudes as before the peak or the valley happened, as it is shown in Figure 21.

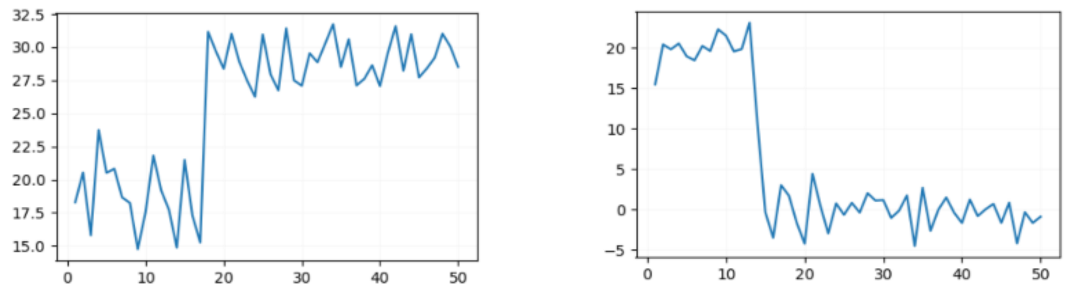


Figure 21: Time series with peak (left) and valley (right) without recovery.

3.1.3.6 Activity diagram

Once the button run is clicked, the process triggered is represented by the activity diagram shown in Figure 22. The light blue nodes and arrows represent the steps developed in this increment.

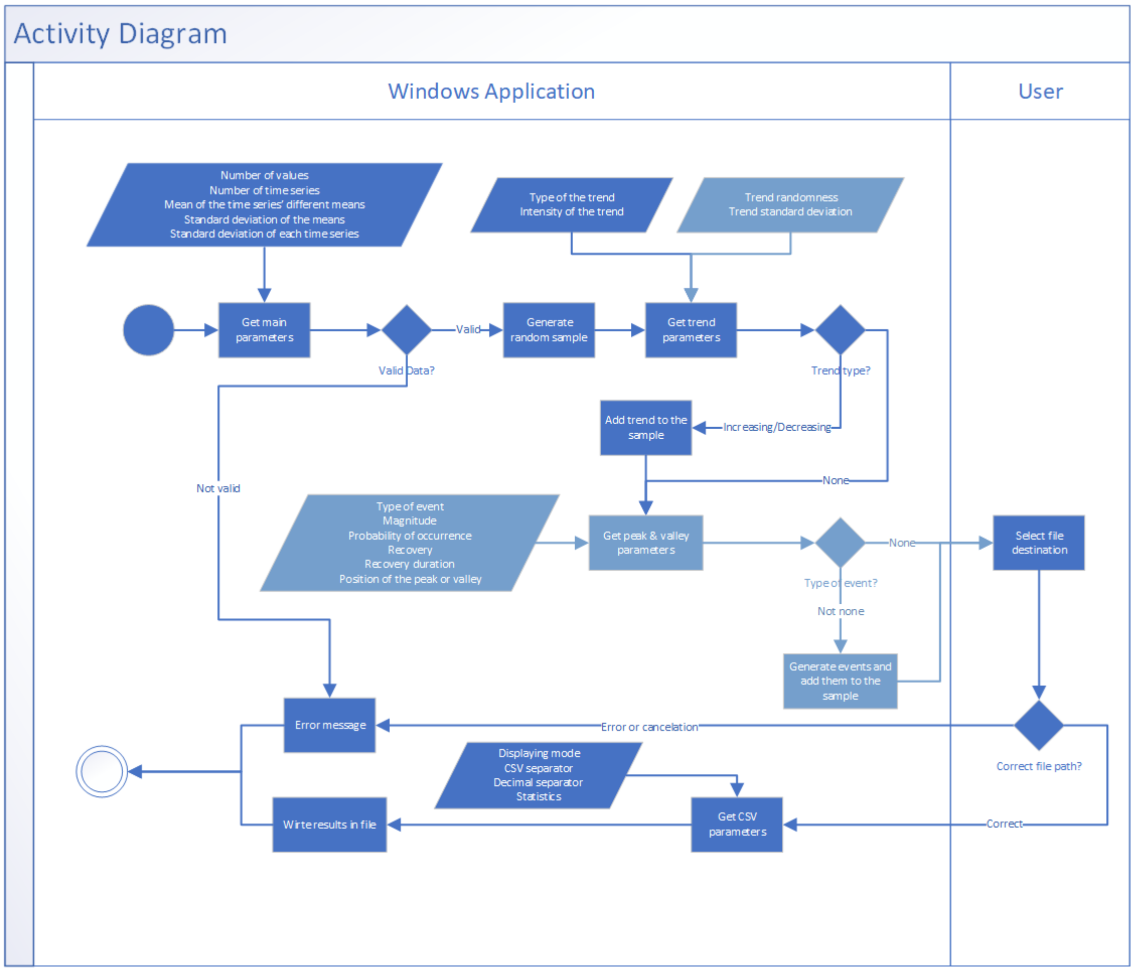


Figure 22: Activity diagram - Third increment

3.1.3.7 Validation

The general purpose of the increment is validated. However, the process for saving the parameters is still not satisfactory. For the next increment, there should be three new functionalities:

- *Save actual configuration as default*: save the parameters in the default configuration file (conf.ini)
- *Export configuration to file*: save the parameters in a configuration file specified by the user.
- *Import parameters from configuration file*: upload the parameters from a configuration file specified by the user.

3.1.4 Fourth Increment: Periodicity

The main goal of this increment is to offer the user the possibility of adding periodicity to the time series. Furthermore, the requests made by the supervisors in the increment before will be cover by adding a new way of saving, importing, and exporting the parameters of the configuration files.

3.1.4.1 Requirements

Functional requirements

- For the periodicity, the user could change the following settings:
  - pe_t = Type of periodicity function added to the sequence. It can be None, Sine or Cosine.
  - pe_a = Amplitude of the function.
  - pe_p = Period of the function generated.

  Given the parameters, if $pe_t = Sine$:

  - Using the function $f(x) = pe_{t^*} sin\left(\frac{2*\pi}{pe_p}\right) * x$ with $x \in [0, n_v - 1]$ and adding the result to the sequence generated.

  If $pe_t = Cosine$:

  - Using the function $f(x) = pe_{t^*} cos\left(\frac{2*\pi}{pe_p}\right) * x$ with $x \in [0, n_v - 1]$ and adding the result to the sequence generated.
- Giving the location and the file name as parameters, the user will be able to import or export the parameters in use from/to a configuration file.

3.1.4.2 Analysis and design

3.1.4.2.1 Graphical User Interface

A Config file tab is added inside the Settings. There are three buttons that corresponds to the functionalities mentioned in the last validation:

- *Save actual configuration as default*: save the parameters in the default configuration file (conf.ini)
- *Export configuration to file*: save the parameters in a configuration file specified by the user.
- *Import parameters from configuration file*: upload the parameters from a configuration file specified by the user.

For export and import options, there is a file dialog triggered which looks like Figure 23:

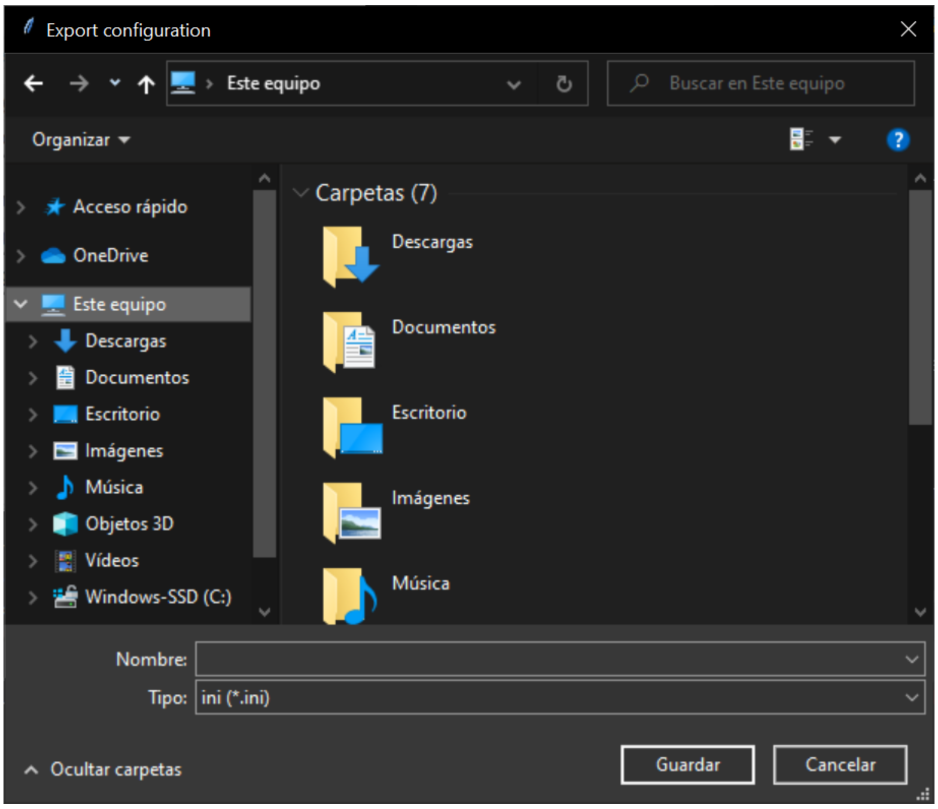


Figure 23: File dialog for export configuration.

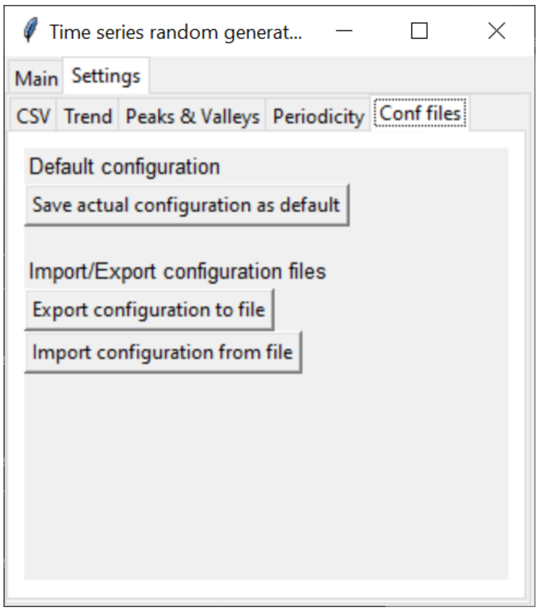
Besides, a periodicity tab is added to Settings. The widgets used are already mentioned on the increments before.

3.1.4.3 Implementation

There were not drawbacks in the implementation of the functionalities mentioned.

3.1.4.4 Application interface

The updated tabs are represented by Figures 24 and 25:



Time series random generat...

Main | Settings

CSV | Trend | Peaks & Valleys | Periodicity | Conf files

Default configuration

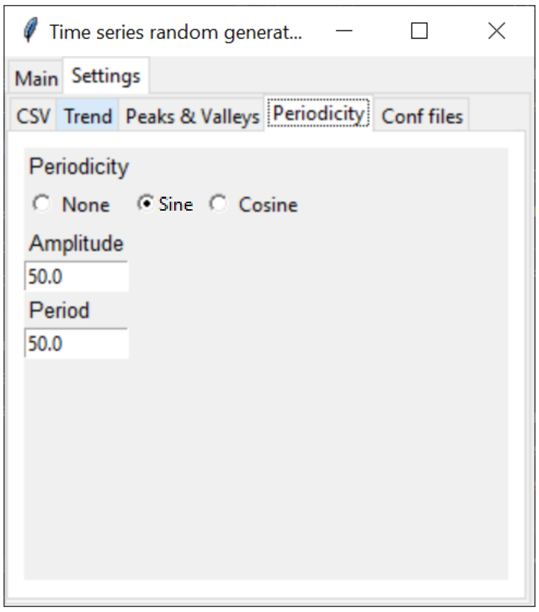
Save actual configuration as default

Import/Export configuration files

Export configuration to file

Import configuration from file

Figure 25: Settings-Conf files tab of the fourth increment



Time series random generat...

Main | Settings

CSV | Trend | Peaks & Valleys | Periodicity | Conf files

Periodicity

None [✓ Sine] Cosine

Amplitude

50.0

Period

50.0

Figure 24: Settings-Periodicity tab of the fourth increment

3.1.4.5 Test

An example of time series is shown in Figure 26. It is obtained with periodicity parameters defined in Figure 24 and with the parameters of Figure 11, without trend, peaks, or valleys.

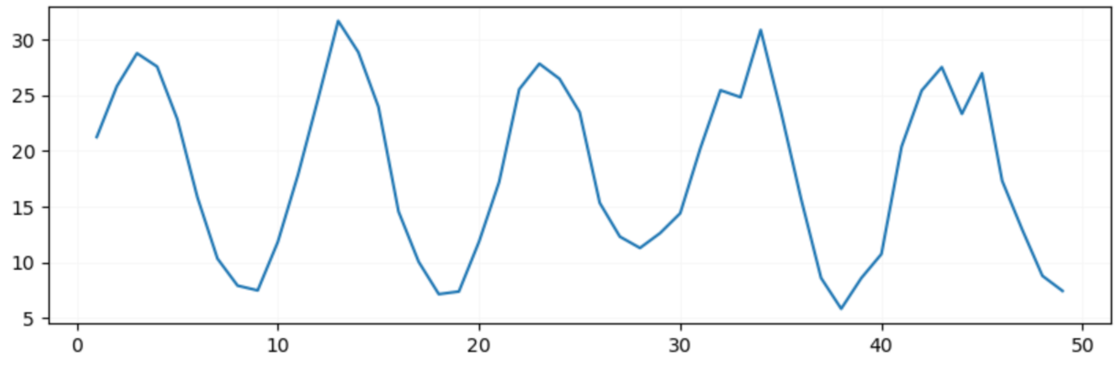


Figure 26: Time series with periodicity.

3.1.4.6 Activity diagram

Once the button run is clicked, the process triggered is represented by the activity diagram shown in Figure 27. The light blue nodes and arrows represent the steps developed in this increment.

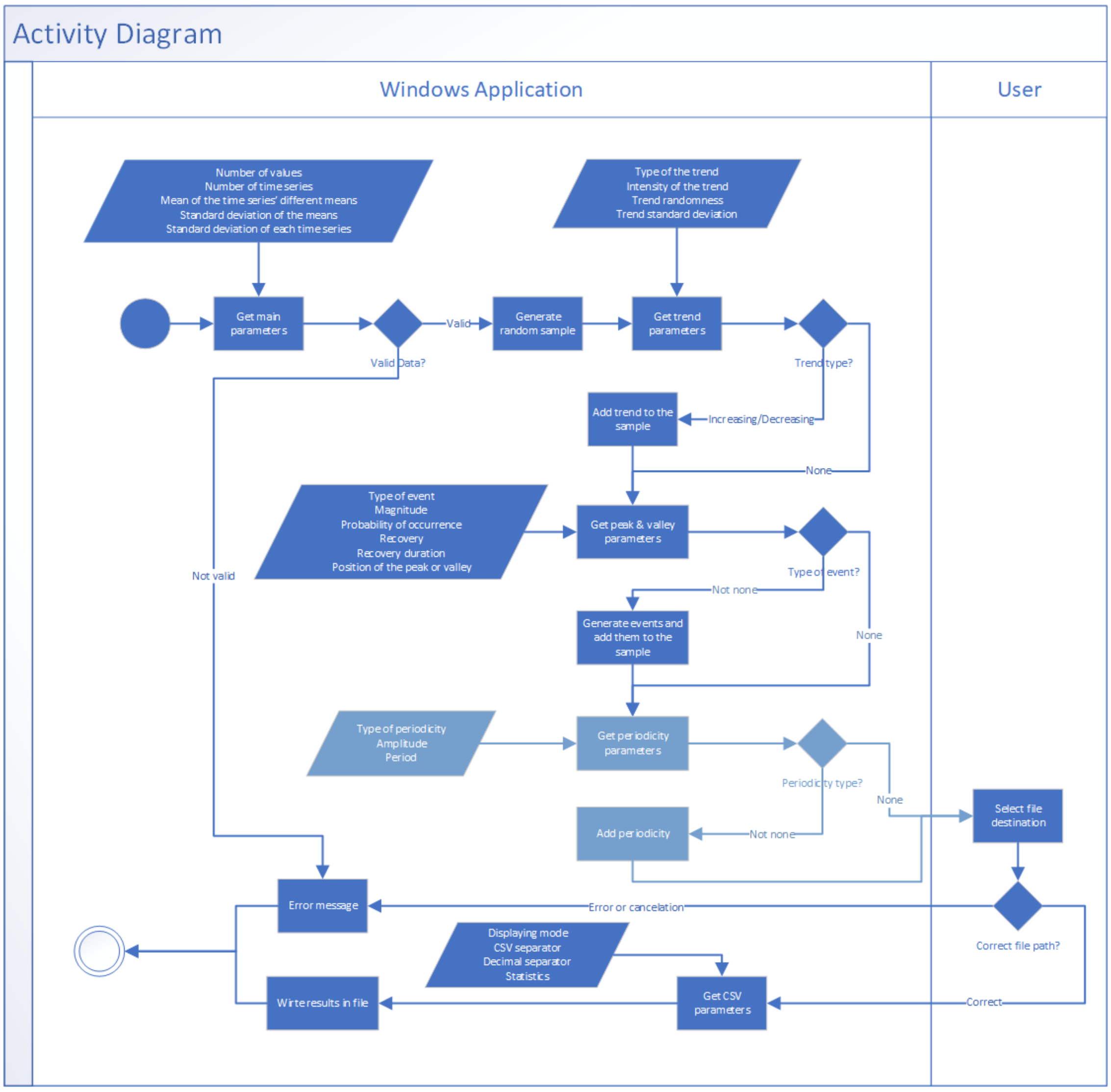


Figure 27: Activity diagram - Fourth increment

3.1.4.7 Validation

The general purpose of the increment was successfully validated by the supervisors.

3.1.5 Fifth Increment: Seed

The main goal of this increment is to offer the user the possibility of adding an external seed saved in a CSV file. Thus, given a seed, the application would be able to generate copies with different properties as peaks, valleys, trend, and random changes over the original.

3.1.5.1 Requirements

Functional requirements

- Giving the location and the file name as parameters, the user will be able to import the seed to add. Besides, the user can select if the seed is added or not to the output file.

3.1.5.2 Analysis and design

3.1.5.2.1 Graphical User Interface

There will be another tab in Settings called Seed. The widgets used have been explained before and can be seen below:

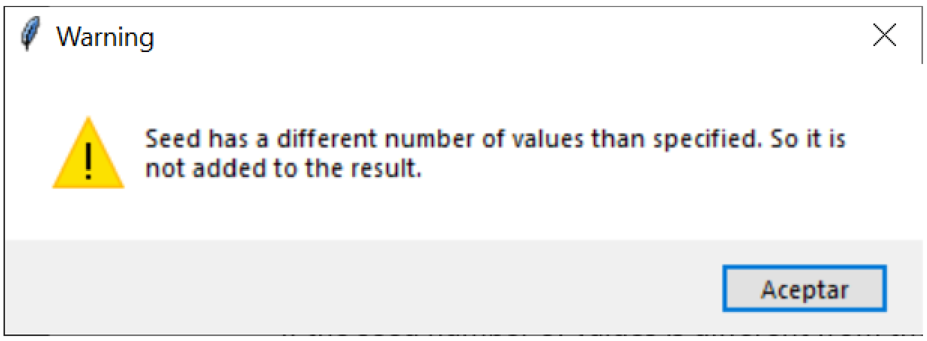
- File dialog for asking the file name and location of the seed.
- A radio button that offers the possibility of adding or not the seed to the output file.

3.1.5.3 Implementation

There are some restrictions when adding a seed file:

- The file must have the extension CSV.
- The CSV separator and decimal separator inside the seed file must be the same as the ones selected in Settings>>CSV tab.
- Only the numbers of the first line will be read. For this reason, the seed must be stored in a row (not in a column)
- The number of values in the file must be the same as the specified in the main tab (number of values parameter)

If the seed number of values is different from the generated ones, the seed will not be used in the generation process. Besides, a warning dialog shown in Figure 28 will arise.

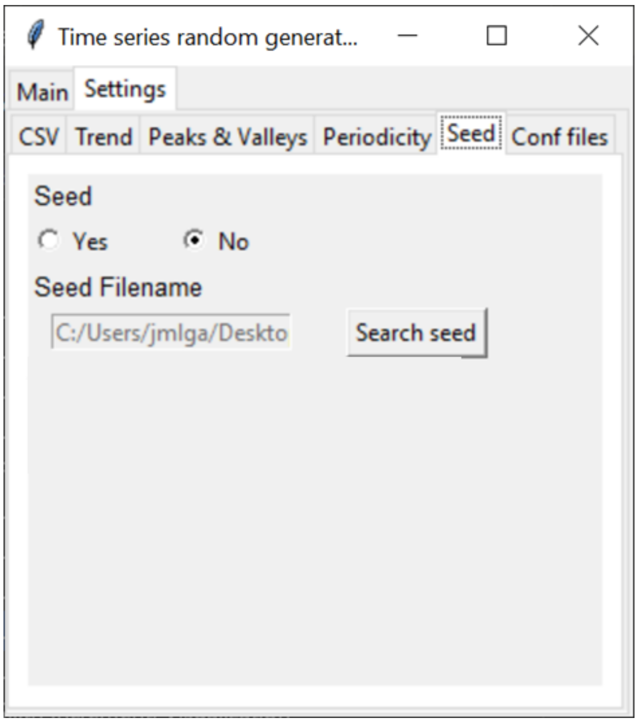


Seed has a different number of values than specified. So it is not added to the result.

Figure 28: Warning dialog when there is a difference in the number of values.

3.1.5.4 Application interface

The updated tab is shown in Figure 29.


Time series random generat...

Main | Settings

CSV | Trend | Peaks & Valleys | Periodicity | Seed | Conf files

Seed

Yes [✓ No]

Seed Filename

C:/Users/jmlga/Deskto

Figure 29: Settings-Seed tab of the fifth increment

3.1.5.5 Test

An example of a seed that could be use is shown in Figure 30.

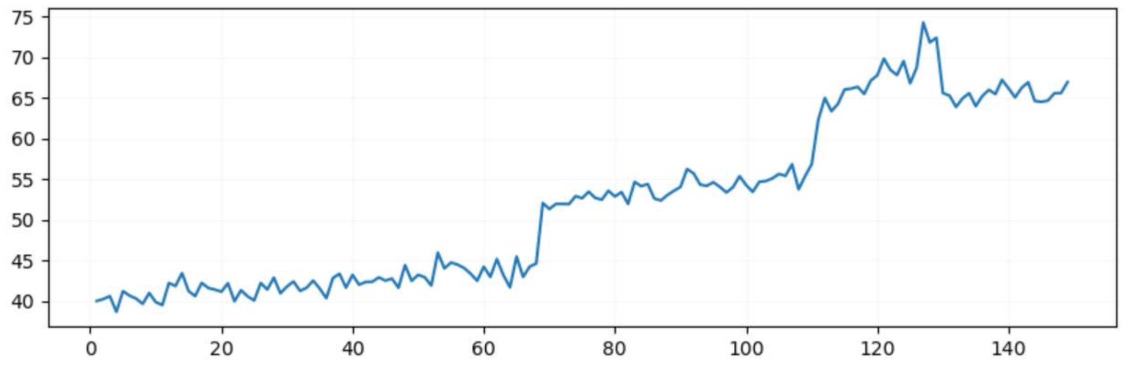


Figure 30: Seed for test in fifth increment.

The parameters used for this test are:

- Zero mean, zero standard deviation and 150 values. No trend, peaks, or valleys.
- Sine periodicity (Amplitude = 5, Period = 20)

With the parameters mentioned before and the seed of Figure 30 selected, the generated time series looks like Figure 31.

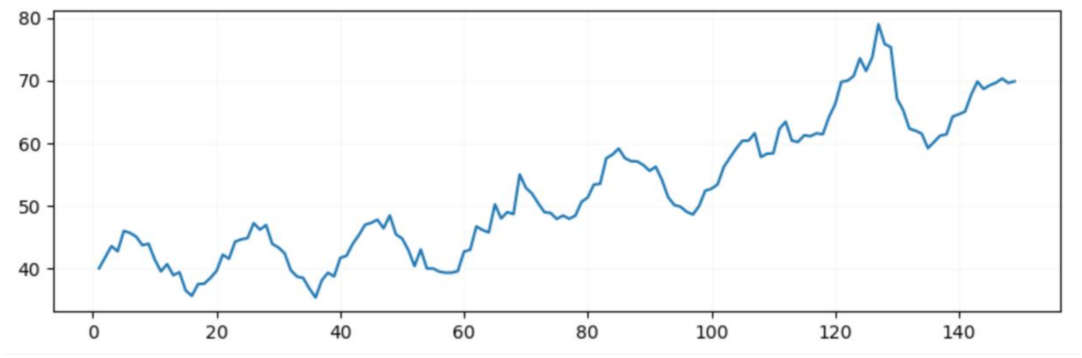


Figure 31: Adding a sine wave to the seed created before.

3.1.5.6 Activity diagram

Once the button run is clicked, the process triggered is represented by the activity diagram shown in Figure 32. The light blue nodes and arrows represent the steps developed in this increment.

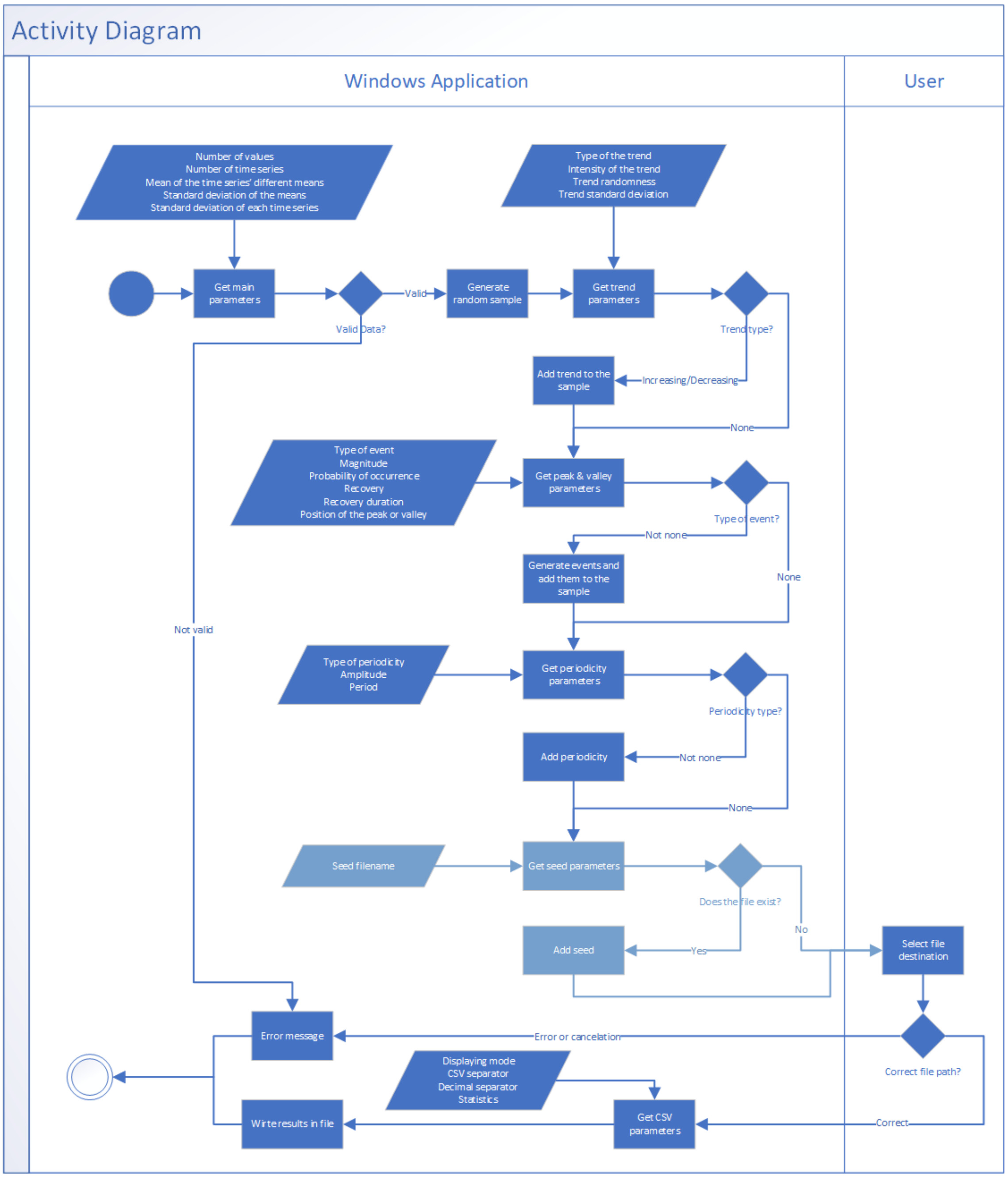


Figure 32: Activity diagram - Fifth increment

3.1.5.7 Validation

The general purpose of the increment is validated. However, due to the restrictions on the seed file selection, the supervisors propose to show some statistics of the seed file once it is chosen in the file dialog. The statistics proposed are:

- Number of values
- Mean
- Standard deviation

3.1.6 Sixth Increment: Visualization

The last increment in this application seeks, as a goal, the visualization of the generated time series. The user will be able to navigate over different graphs which represent the generated time series and to focus on specific parts of them.

3.1.6.1 Requirements

Functional requirements

- Once the Run button is clicked in the main tab and the calculations have been done, the visualization window arises.
- Two buttons (Previous/Next) allow the user to move to another time series in the visualization window.
- If the mouse hovers over the graph and the mouse scroll wheel is used, it would zoom in/out the graph.
- If the mouse left-click to pan and drag the graph, it would change the visible part of the graph.

3.1.6.2 Analysis and design

3.1.6.2.1 Graphical User Interface

The visualization interface is bigger than the ones used in the increments before. Because of this, the new part will be displayed in a different window. This new window will have the visualization part made with matplotlib**.**

Matplotlib is a library for making 2D plots of arrays in Python. It can be used in a Pythonic, object-oriented way. Although Matplotlib is written primarily in pure Python, it makes heavy use of NumPy and other extension code to provide good performance even for large arrays.[37]

To accomplish the panning and dragging requirement, the matplotlib navigation toolbar[38] is used. Nevertheless, every tool except the panning one is deleted from the toolbar. The implementation of the mouse scroll wheel is done programmatically and is explained in the implementation part.

[37] Hunter, J. (2008). Matplotlib. Retrieved from https://matplotlib.org/users/history.html
[38] Matplotlib contributors. (n.d.). Matplotlib: Interctive Navigation. Retrieved from https://matplotlib.org/3.2.1/users/navigation_toolbar.html

3.1.6.3 Implementation

To implement the mouse wheel scrolling functionality, there is a scroll listener in the graph which triggers the next code:

```
def zoom_fun(event):
    # get the current x and y limits
    cur_xlim = get_xlim()
    cur_ylim = get_ylim()

    # get event x location
    xdata = event.xdata
    # get event y location
    ydata = event.ydata

    if event.button == 'down':
        # deal with zoom in
        scale_factor = 1 / 1.1
    elif event.button == 'up':
        # deal with zoom out
        scale_factor = 1.1
    else:
        # deal with something that should never happen
        scale_factor = 1
        print(event.button)

    # set new limits
    set_xlim([xdata - (xdata - cur_xlim[0]) / scale_factor,
                 xdata + (cur_xlim[1] - xdata) / scale_factor])
    set_ylim([ydata - (ydata - cur_ylim[0]) / scale_factor,
                 ydata + (cur_ylim[1] - ydata) / scale_factor])
```

Thus, once the scrolling is done, the axes limits are redefined to zoom in or out in the graph.

3.1.6.4 Application interface

The visualization window looks like Figure 33.

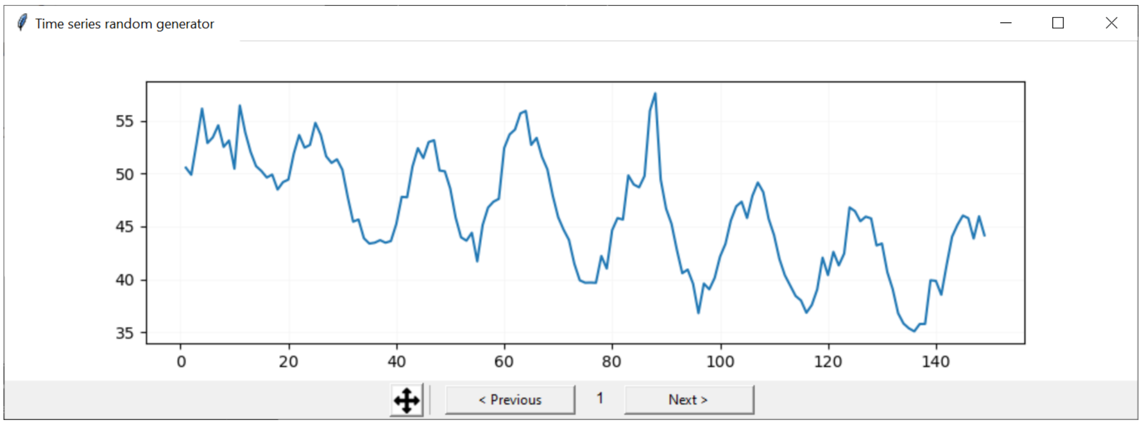


Figure 33: Visualization window for the last increment.

For this increment, there is not a test section.

3.1.6.5 Activity diagram

Once the button run is clicked, the process triggered is represented by the activity diagram shown in Figure 34.

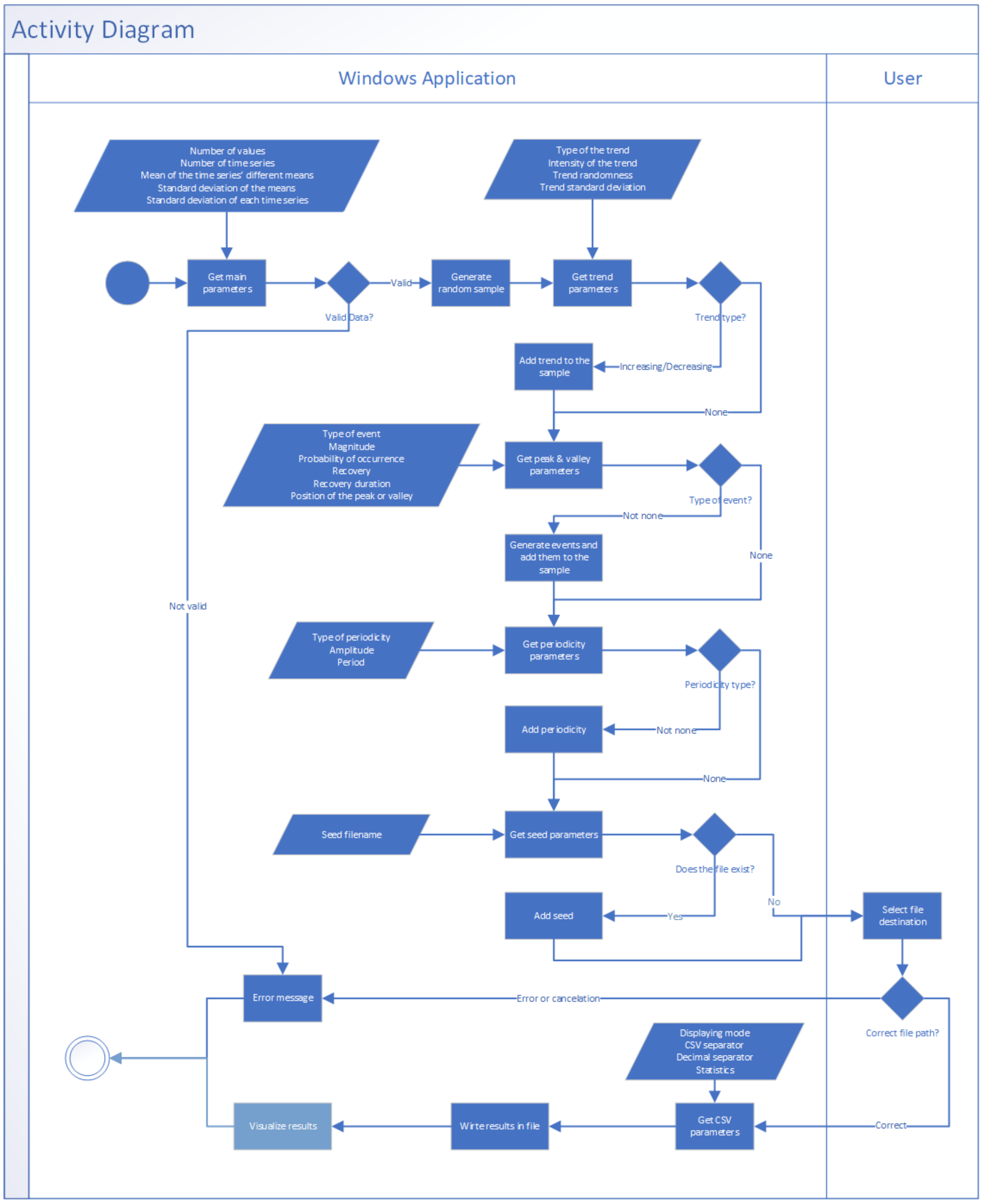


Figure 34: Activity diagram – Sixth increment

3.1.6.6 Validation

The general purpose of the increment is validated. Nevertheless, after an exhaustive use of the application done by the supervisors, there are two things that must change:

- The distribution of the frames is not satisfactory. From now on, there will be three main tabs:
    - CSV
    - Settings
    - Configuration fields

  Settings tab will contain the following tabs:
    - Main
    - Trend
    - Peaks & Valleys
    - Periodicity
    - Seed
- A “Restore default configuration” button is needed in configuration files.
- The periodicity tab needs to duplicate its widgets to provide a second periodicity addition.

It is not necessary to develop a whole increment for solving the problems showed before. The final application will be shown in the results section.

3.2 Data mining application

The second objective of this thesis would be to build another software tool for applying data mining techniques. Thus, the results given by the first application would be grouped. It will check the correct functioning of the time series random generator and its many functionalities.

This section describes the process of development of the data mining application.

3.2.1 First Increment: Discrete Fourier transform and visualization

The main objective in this first increment is to create another self-contained application for Windows which allow the user to import some time series files to visualize them along with a Fourier synthesis.

3.2.1.1 Requirements

Functional requirements

- The application would receive the following parameters from the user.
    - fp = File path
    - c = Number of Fourier coefficients used

  Given these parameters, each time series of the file selected is transformed (DFT) with the FFT algorithm. Only c coefficients from the ones given as a result are used to generate the inverse discrete Fourier transform. Nevertheless, each coefficient has two parts: the real and imaginary part. Thus, if $c = 4$ then there are 8 coefficients (4 real parts and 4 imaginary parts)

  Both the original time series and the inverse generated are shown in a new window.

Non-functional requirements

- The application should be suitable to any type of computer running a Windows operating system.
- The application's language must be English.
- The application must implement a system to import different files at the same time and manage these imported files.

3.2.1.2 Analysis and design

3.2.1.2.1 Graphical User Interface

As in the other application, Tkinter is the tool selected to build a graphical user interface. For this increment it would be necessary to create a window with the following widgets:

- Labels, entries, and buttons. All of them are mentioned and explained before in Section 3.1.1.2.1.
- Listbox: displays a list of single-line text items, usually lengthy, and allows the user to browse through the list.[39]

3.2.1.2.2 Discrete Fourier Transform (DFT) and its inverse

NumPy is the package selected to apply the discrete Fourier transform. It provides a function for computing the one-dimensional discrete Fourier Transform (DFT) of a real-valued array by means of an efficient algorithm called the Fast Fourier Transform (FFT).[40]

The result of applying this function is an array of complex values whose real and imaginary components are the Fourier coefficients. Once this result is obtained, only the number of coefficients given by the user are used. And with these components, the inverse DFT reconstructs the original time series.

3.2.1.3 Implementation

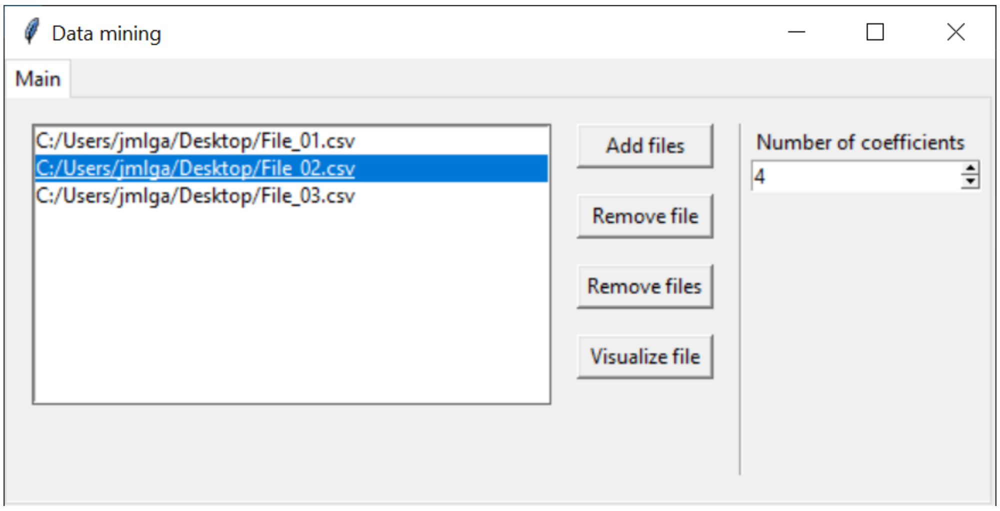
There were not drawbacks in the implementation of the mentioned functionalities.

[39] Roseman, M. (2019). Listbox. In M. Roseman, Modern tkinter for busy developers (2 ed., pp. 44-45). Late Afternoon Press.

[40] NumPy contributors. (2020, June 16). numpy.fft.rfft. Retrieved from NumPy: https://numpy.org/devdocs/reference/generated/numpy.fft.rfft.html#numpy.fft.rfft

3.2.1.4 Application interface

The application interface is shown in Figure 35. The testing will be done by the supervisors, and the parts that would need to be changed, dropped, or added will be reviewed during the validation process.



C:/Users/jmlga/Desktop/File_01.csv
C:/Users/jmlga/Desktop/File_02.csv
C:/Users/jmlga/Desktop/File_03.csv

Number of coefficients
4

Figure 35: Data mining application in the first increment.

3.2.1.5 Test

Once the "Visualize file" button is clicked, the results with eight and sixteen coefficients for a previously generated time series would be as Figure 36 and 37.

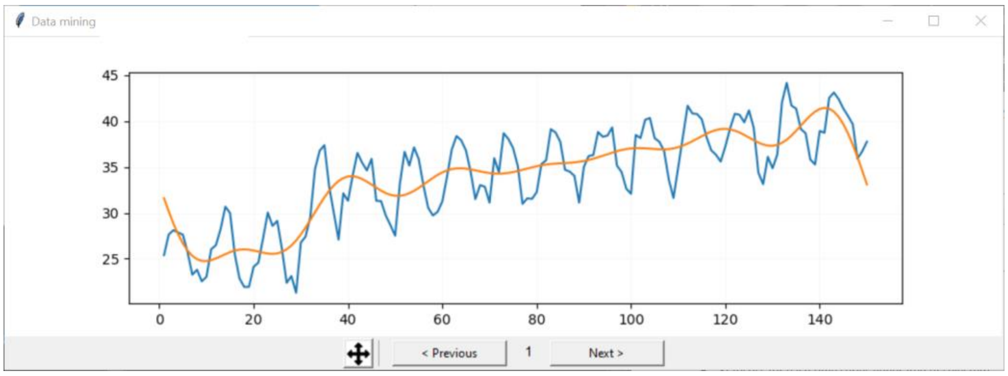


Figure 36: Time series visualization along the Fourier Transform inverse with eight coefficients

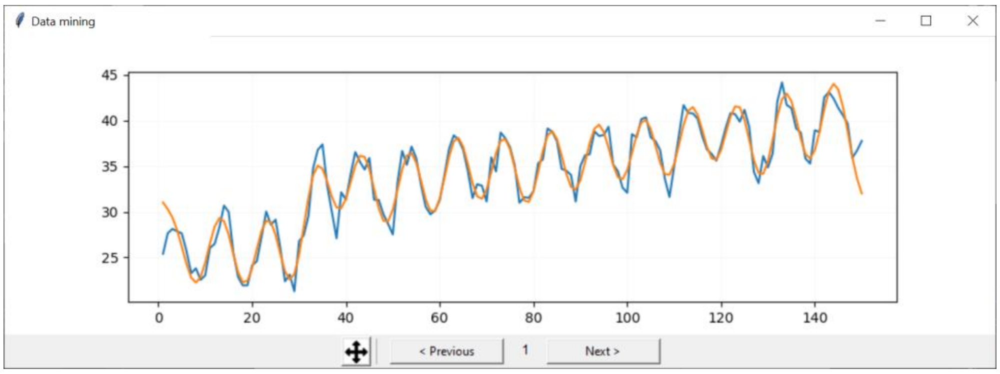


Figure 37: Time series visualization along the Fourier Transform inverse with sixteen coefficients

3.2.1.6 Validation

The general purpose of the increment was successfully validated by the supervisors.

3.2.2 Second Increment: Partitional clustering on Fourier coefficients

The new functionality developed in this increment applies a partitional clustering with the Fourier coefficients as inputs. Having different time series files, each of them usually containing a different type of time series, the main goal is to check if the clusters found by the algorithm, when applied to the whole set of time series, match the different files.

3.2.2.1 Requirements

Functional requirements

- The application would receive the following parameters from the user.
    - fp = Files paths
    - c = Number of Fourier coefficients used
    - k = Number of clusters

  Given these parameters, each time series of the files included in the listbox is transformed (DFT) with the FFT algorithm. Only c coefficients from the ones given as a result are used as inputs for the k-Means algorithm with k clusters. The results will be written in CSV.

3.2.2.2 Analysis and design

3.2.2.2.1 K-Means

The scikit-learn package provides a k-Means algorithm which tries to separate samples in n groups of equal variances, minimizing a criterion known as the inertia. The algorithm divides a set of N samples X into K disjoint clusters C, each described by the mean μ_j of the samples in the cluster. The inertia mentioned before is:

$$\sum_{i=0}^{n} \min_{\mu_j \epsilon C}\left(\left\|x_i - \mu_j\right\|^2\right)$$

It can be recognized as a measure of how internally coherent clusters are[41]. It has some drawbacks in different situations. For example, it performs poorly with elongated clusters, or manifolds with irregular shapes.

However, the different scenarios are not going to be studied due to the huge amount of different time series types (or behaviours) the generator tool can build.

3.2.2.3 Implementation

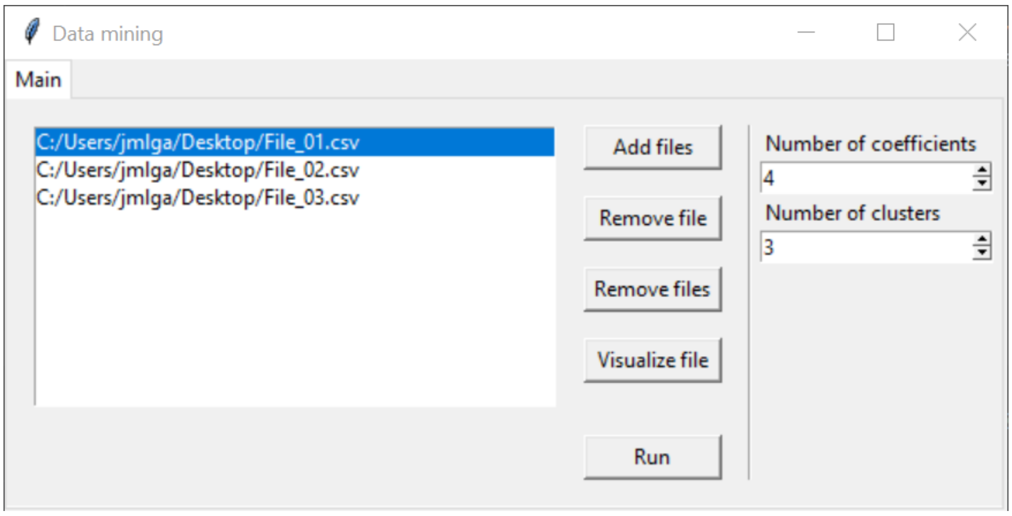
There were not drawbacks in the implementation of the mentioned functionality

.

[41] Scikit-learn developers. (2017). Clustering - KMeans. Retrieved from Scikit-learn: https://scikit-learn.org/stable/modules/clustering.html#k-means

3.2.2.4 Application interface

The updated application interface can be seen in Figure 38.



C:/Users/jmlga/Desktop/File_01.csv
C:/Users/jmlga/Desktop/File_02.csv
C:/Users/jmlga/Desktop/File_03.csv

Number of coefficients: 4
Number of clusters: 3

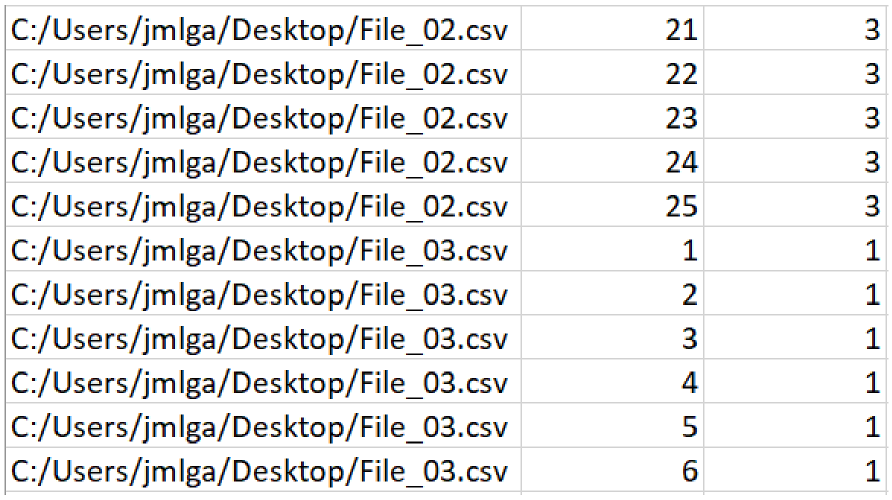
Figure 38: Data mining application in the second increment.

3.2.2.5 Test

Once the “Run” button is clicked, all the operations are done. First, all the input files are merged in one file containing the whole set of time series. Then, Fourier transform is applied, and the time series are transformed into a new representation using the desired coefficients number. The clustering algorithm is applied to these data and a file dialog arises to save the result in a CSV file, with one row per time series. The output file will contain the clustering results stored in three columns as follows:

- *File:* Name of time series source file
- *Time series:* Index of the time series inside the source file.
- *Cluster:* Cluster in which the time series is grouped.

An example of a clustering output file is shown in Figure 39:



C:/Users/jmlga/Desktop/File_02.csv	21	3
C:/Users/jmlga/Desktop/File_02.csv	22	3
C:/Users/jmlga/Desktop/File_02.csv	23	3
C:/Users/jmlga/Desktop/File_02.csv	24	3
C:/Users/jmlga/Desktop/File_02.csv	25	3
C:/Users/jmlga/Desktop/File_03.csv	1	1
C:/Users/jmlga/Desktop/File_03.csv	2	1
C:/Users/jmlga/Desktop/File_03.csv	3	1
C:/Users/jmlga/Desktop/File_03.csv	4	1
C:/Users/jmlga/Desktop/File_03.csv	5	1
C:/Users/jmlga/Desktop/File_03.csv	6	1

Figure 39: Output example

The output can be understood as follows:

- Time series #21 from C:/Users/jmlga/Desktop/File_01.csv file has been assigned to cluster #3.
- Time series #6 from C:/Users/jmlga/Desktop/File_03.csv file has been assigned to cluster #1
- …

The Figure 39 shows how all the five time series of the File_02.csv file are assigned to one cluster, while all those of the File_03.csv file are assigned to a different one.

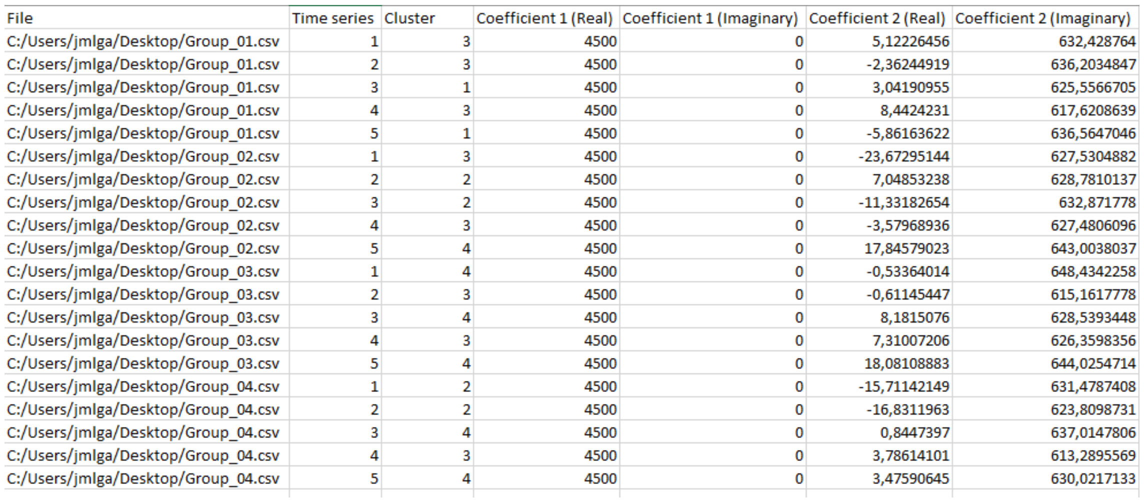
Taking into account that these two files have been generated with the previously created time series random generation tool, applying two different configurations, the fact that the clusters found by the algorithm coincide with the File_02 File_03 sets confirms that the generator works correctly, being able to generate series of different types (or different behaviour).

3.2.2.6 Validation

The general purpose of the increment is validated, but there are some little changes or tests proposed by the supervisors. Since this is the last increment, both the changes and the tests are done in this section.

There is one thing that must be changed. When the CSV file is written, the coefficients used by the Fourier transform must be appended. The new appearance of the CSV file can be seen in Figure 40 and 41.

Furthermore, the supervisors propose a test for checking that the data mining tool is working independently from the files selected as inputs. The idea is to generate three different files with five time series each one. All of them need to be generated with the same input parameters. Thus, if the algorithm works independently from the files then it would not cluster the time series depending on its file. The result of this experiment can be seen in Figure 40:



File	Time series	Cluster	Coefficient 1 (Real)	Coefficient 1 (Imaginary)	Coefficient 2 (Real)	Coefficient 2 (Imaginary)
C:/Users/jmlga/Desktop/Group_01.csv	1	3	4500	0	5,12226456	632,428764
C:/Users/jmlga/Desktop/Group_01.csv	2	3	4500	0	-2,36244919	636,2034847
C:/Users/jmlga/Desktop/Group_01.csv	3	1	4500	0	3,04190955	625,5566705
C:/Users/jmlga/Desktop/Group_01.csv	4	3	4500	0	8,4424231	617,6208639
C:/Users/jmlga/Desktop/Group_01.csv	5	1	4500	0	-5,86163622	636,5647046
C:/Users/jmlga/Desktop/Group_02.csv	1	3	4500	0	-23,67295144	627,5304882
C:/Users/jmlga/Desktop/Group_02.csv	2	2	4500	0	7,04853238	628,7810137
C:/Users/jmlga/Desktop/Group_02.csv	3	2	4500	0	-11,33182654	632,871778
C:/Users/jmlga/Desktop/Group_02.csv	4	3	4500	0	-3,57968936	627,4806096
C:/Users/jmlga/Desktop/Group_02.csv	5	4	4500	0	17,84579023	643,0038037
C:/Users/jmlga/Desktop/Group_03.csv	1	4	4500	0	-0,53364014	648,4342258
C:/Users/jmlga/Desktop/Group_03.csv	2	3	4500	0	-0,61145447	615,1617778
C:/Users/jmlga/Desktop/Group_03.csv	3	4	4500	0	8,1815076	628,5393448
C:/Users/jmlga/Desktop/Group_03.csv	4	3	4500	0	7,31007206	626,3598356
C:/Users/jmlga/Desktop/Group_03.csv	5	4	4500	0	18,08108883	644,0254714
C:/Users/jmlga/Desktop/Group_04.csv	1	2	4500	0	-15,71142149	631,4787408
C:/Users/jmlga/Desktop/Group_04.csv	2	2	4500	0	-16,8311963	623,8098731
C:/Users/jmlga/Desktop/Group_04.csv	3	4	4500	0	0,8447397	637,0147806
C:/Users/jmlga/Desktop/Group_04.csv	4	3	4500	0	3,78614101	613,2895569
C:/Users/jmlga/Desktop/Group_04.csv	5	4	4500	0	3,47590645	630,0217133

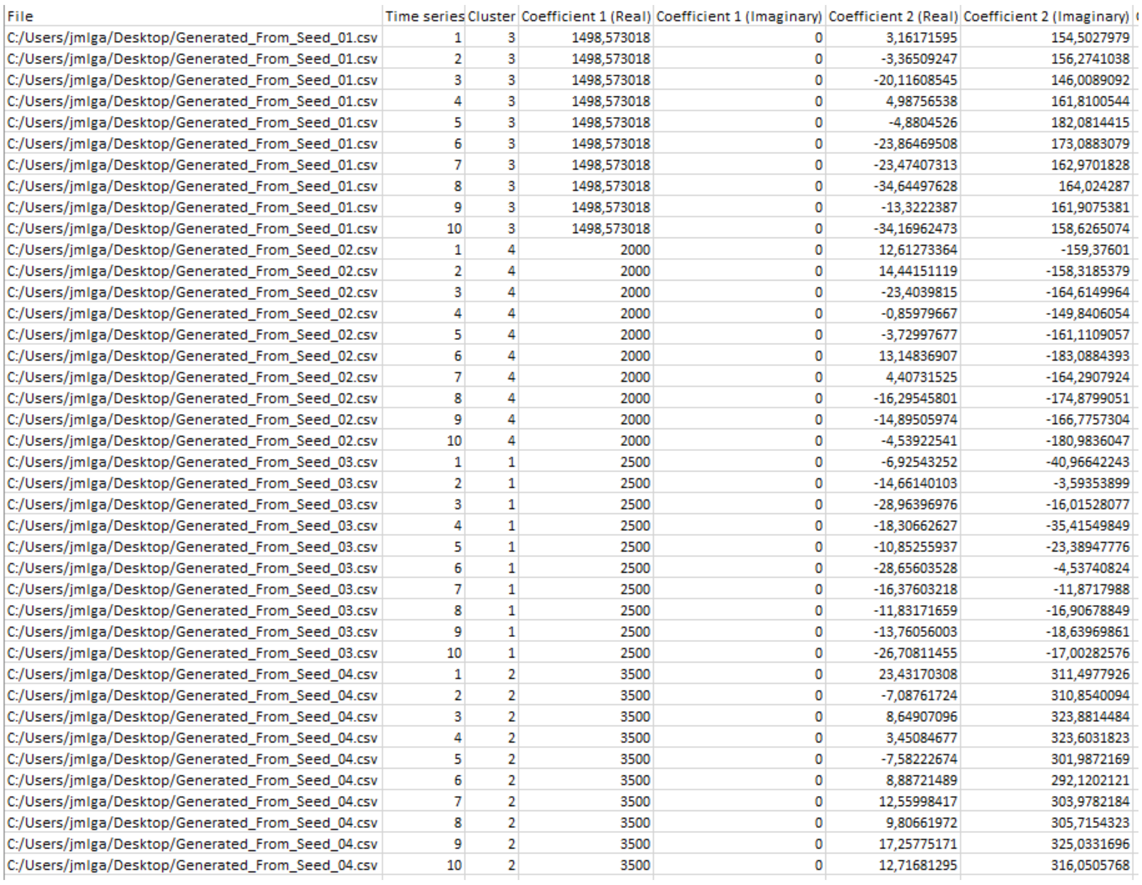
Figure 40: Test proposed by the supervisors

Since every file (Group_01, Group_02, Group_03 and Group_04) is generated with the same parameters, the clustering algorithm finds small similarities in the different time series to do the grouping. And the time series belonging to a specific file can be grouped in all the clusters. This demonstrate that the clustering is done independently from the file.

Another test proposed by the supervisors consist of the next steps:

1. Generate four time series, each one with different parameters as inputs. Trend, peaks or valleys can be added.
2. For each of them:
    1. Use it as a seed for a new generation of ten time series. Without adding trend, peaks, or valleys. And having a small standard deviation value.
    2. Save the new ten time series in specific CSV file.
3. Use the four CSV files as inputs for the data mining application.

The CSV file saved as result is shown in Figure 41.



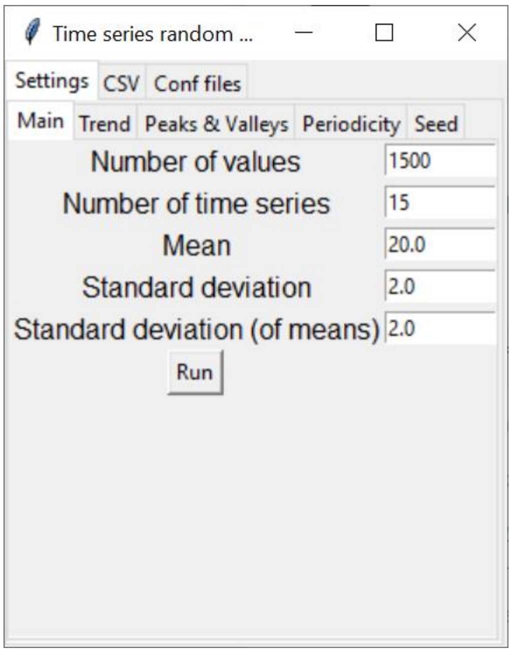
File	Time series	Cluster	Coefficient 1 (Real)	Coefficient 1 (Imaginary)	Coefficient 2 (Real)	Coefficient 2 (Imaginary)
C:/Users/jmlga/Desktop/Generated_From_Seed_01.csv	1	3	1498,573018	0	3,16171595	154,5027979
C:/Users/jmlga/Desktop/Generated_From_Seed_01.csv	2	3	1498,573018	0	-3,36509247	156,2741038
C:/Users/jmlga/Desktop/Generated_From_Seed_01.csv	3	3	1498,573018	0	-20,11608545	146,0089092
C:/Users/jmlga/Desktop/Generated_From_Seed_01.csv	4	3	1498,573018	0	4,98756538	161,8100544
C:/Users/jmlga/Desktop/Generated_From_Seed_01.csv	5	3	1498,573018	0	-4,8804526	182,0814415
C:/Users/jmlga/Desktop/Generated_From_Seed_01.csv	6	3	1498,573018	0	-23,86469508	173,0883079
C:/Users/jmlga/Desktop/Generated_From_Seed_01.csv	7	3	1498,573018	0	-23,47407313	162,9701828
C:/Users/jmlga/Desktop/Generated_From_Seed_01.csv	8	3	1498,573018	0	-34,64497628	164,024287
C:/Users/jmlga/Desktop/Generated_From_Seed_01.csv	9	3	1498,573018	0	-13,3222387	161,9075381
C:/Users/jmlga/Desktop/Generated_From_Seed_01.csv	10	3	1498,573018	0	-34,16962473	158,6265074
C:/Users/jmlga/Desktop/Generated_From_Seed_02.csv	1	4	2000	0	12,61273364	-159,37601
C:/Users/jmlga/Desktop/Generated_From_Seed_02.csv	2	4	2000	0	14,44151119	-158,3185379
C:/Users/jmlga/Desktop/Generated_From_Seed_02.csv	3	4	2000	0	-23,4039815	-164,6149964
C:/Users/jmlga/Desktop/Generated_From_Seed_02.csv	4	4	2000	0	-0,85979667	-149,8406054
C:/Users/jmlga/Desktop/Generated_From_Seed_02.csv	5	4	2000	0	-3,72997677	-161,1109057
C:/Users/jmlga/Desktop/Generated_From_Seed_02.csv	6	4	2000	0	13,14836907	-183,0884393
C:/Users/jmlga/Desktop/Generated_From_Seed_02.csv	7	4	2000	0	4,40731525	-164,2907924
C:/Users/jmlga/Desktop/Generated_From_Seed_02.csv	8	4	2000	0	-16,29545801	-174,8799051
C:/Users/jmlga/Desktop/Generated_From_Seed_02.csv	9	4	2000	0	-14,89505974	-166,7757304
C:/Users/jmlga/Desktop/Generated_From_Seed_02.csv	10	4	2000	0	-4,53922541	-180,9836047
C:/Users/jmlga/Desktop/Generated_From_Seed_03.csv	1	1	2500	0	-6,92543252	-40,96642243
C:/Users/jmlga/Desktop/Generated_From_Seed_03.csv	2	1	2500	0	-14,66140103	-3,59353899
C:/Users/jmlga/Desktop/Generated_From_Seed_03.csv	3	1	2500	0	-28,96396976	-16,01528077
C:/Users/jmlga/Desktop/Generated_From_Seed_03.csv	4	1	2500	0	-18,30662627	-35,41549849
C:/Users/jmlga/Desktop/Generated_From_Seed_03.csv	5	1	2500	0	-10,85255937	-23,38947776
C:/Users/jmlga/Desktop/Generated_From_Seed_03.csv	6	1	2500	0	-28,65603528	-4,53740824
C:/Users/jmlga/Desktop/Generated_From_Seed_03.csv	7	1	2500	0	-16,37603218	-11,8717988
C:/Users/jmlga/Desktop/Generated_From_Seed_03.csv	8	1	2500	0	-11,83171659	-16,90678849
C:/Users/jmlga/Desktop/Generated_From_Seed_03.csv	9	1	2500	0	-13,76056003	-18,63969861
C:/Users/jmlga/Desktop/Generated_From_Seed_03.csv	10	1	2500	0	-26,70811455	-17,00282576
C:/Users/jmlga/Desktop/Generated_From_Seed_04.csv	1	2	3500	0	23,43170308	311,4977926
C:/Users/jmlga/Desktop/Generated_From_Seed_04.csv	2	2	3500	0	-7,08761724	310,8540094
C:/Users/jmlga/Desktop/Generated_From_Seed_04.csv	3	2	3500	0	8,64907096	323,8814484
C:/Users/jmlga/Desktop/Generated_From_Seed_04.csv	4	2	3500	0	3,45084677	323,6031823
C:/Users/jmlga/Desktop/Generated_From_Seed_04.csv	5	2	3500	0	-7,58222674	301,9872169
C:/Users/jmlga/Desktop/Generated_From_Seed_04.csv	6	2	3500	0	8,88721489	292,1202121
C:/Users/jmlga/Desktop/Generated_From_Seed_04.csv	7	2	3500	0	12,55998417	303,9782184
C:/Users/jmlga/Desktop/Generated_From_Seed_04.csv	8	2	3500	0	9,80661972	305,7154323
C:/Users/jmlga/Desktop/Generated_From_Seed_04.csv	9	2	3500	0	17,25775171	325,0331696
C:/Users/jmlga/Desktop/Generated_From_Seed_04.csv	10	2	3500	0	12,71681295	316,0505768

Figure 41: Test proposed by the supervisors

Every cluster is grouping the time series of a specific file. So, adding small randomness to different seeds allow to generate new time series which maintain a common statistical base. Although there are different versions of the same seed, the clustering algorithm recognise its similarity.

4 Results and conclusions

The final application interfaces are represented next by several figures. First, the random time series generator interface is shown in Figures 42, 43, 44, 45, 46, 47, 48 and 49. This application will allow to generate random sere


Time series random ...

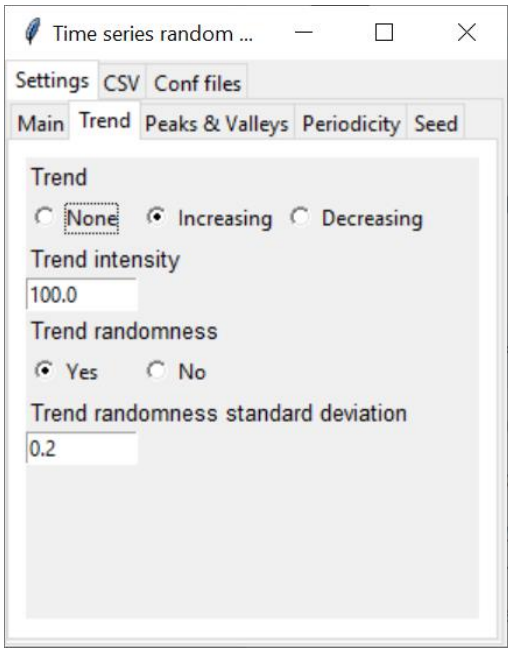
Settings | CSV | Conf files

Main | Trend | Peaks & Valleys | Periodicity | Seed

Number of values: 1500
Number of time series: 15
Mean: 20.0
Standard deviation: 2.0
Standard deviation (of means): 2.0

Run

Figure 45: Settings - Main tab. Final version.


Time series random ...

Settings | CSV | Conf files

Main | Trend | Peaks & Valleys | Periodicity | Seed

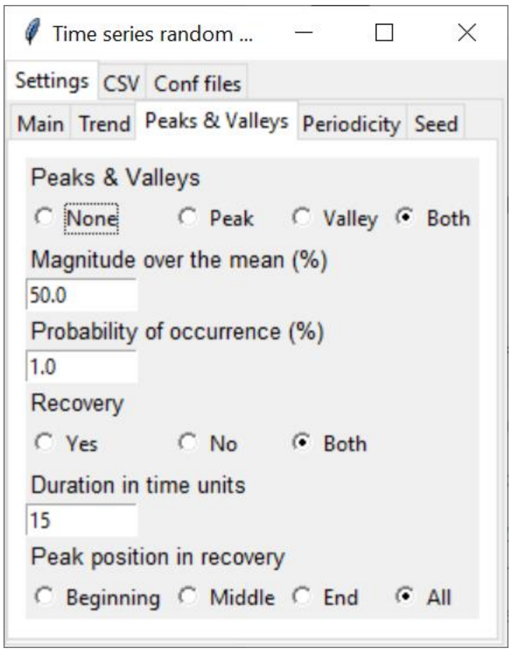
Trend
None [✓ Increasing] Decreasing

Trend intensity
100.0

Trend randomness
[✓ Yes] No

Trend randomness standard deviation
0.2

Figure 44: Settings - Trend tab. Final version.


Time series random ...

Settings | CSV | Conf files

Main | Trend | Peaks & Valleys | Periodicity | Seed

Peaks & Valleys
None Peak Valley [✓ Both]

Magnitude over the mean (%)
50.0

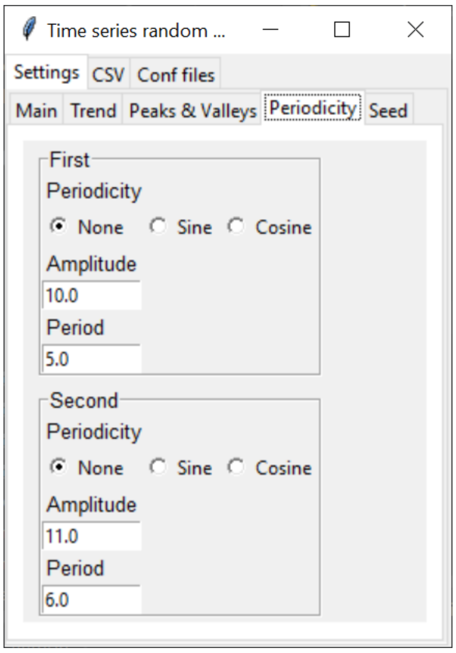
Probability of occurrence (%)
1.0

Recovery
Yes No [✓ Both]

Duration in time units
15

Peak position in recovery
Beginning Middle End [✓ All]

Figure 43: Settings - Peaks & Valleys tab. Final version.


Time series random ...

Settings | CSV | Conf files

Main | Trend | Peaks & Valleys | Periodicity | Seed

First

Periodicity
[✓ None] Sine Cosine

Amplitude
10.0

Period
5.0

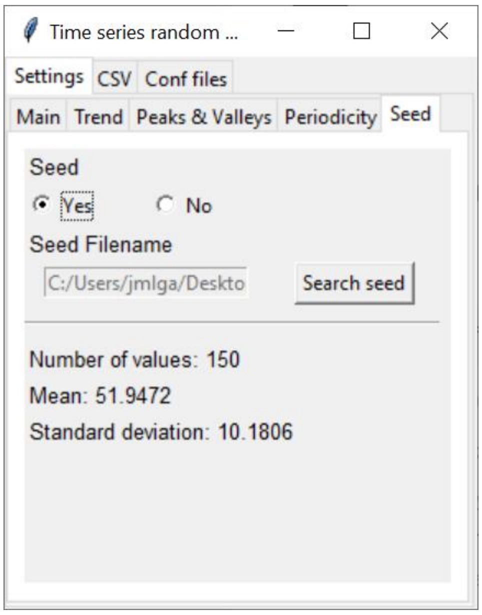
Second

Periodicity
[✓ None] Sine Cosine

Amplitude
11.0

Period
6.0

Figure 42: Settings-Periodicity tab. Final version.


Time series random ...

Settings | CSV | Conf files

Main | Trend | Peaks & Valleys | Periodicity | Seed

Seed

[✓ Yes] No

Seed Filename

C:/Users/jmlga/Deskto

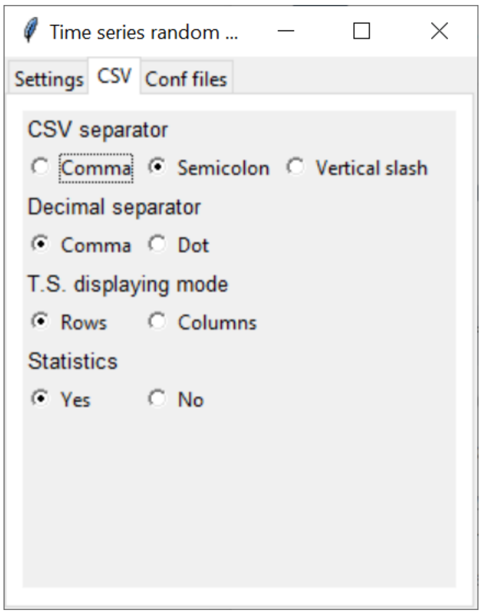
Search seed

Number of values: 150

Mean: 51.9472

Standard deviation: 10.1806

Figure 46: Settings - Seed tab. Final version.


Time series random ...

Settings | CSV | Conf files

CSV separator

Comma [✓ Semicolon] Vertical slash

Decimal separator

[✓ Comma] Dot

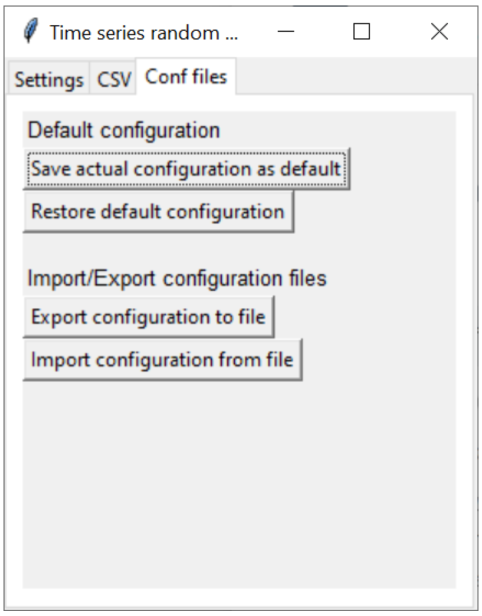
T.S. displaying mode

[✓ Rows] Columns

Statistics

[✓ Yes] No

Figure 47: CSV tab. Final version.


Time series random ...

Settings | CSV | Conf files

Default configuration

Save actual configuration as default

Restore default configuration

Import/Export configuration files

Export configuration to file

Import configuration from file

Figure 49: Configuration files tab. Final version.

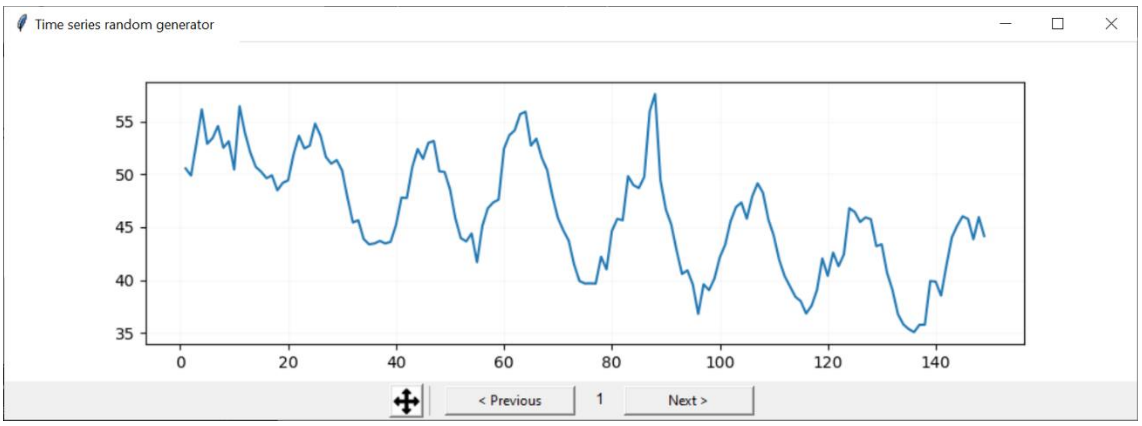
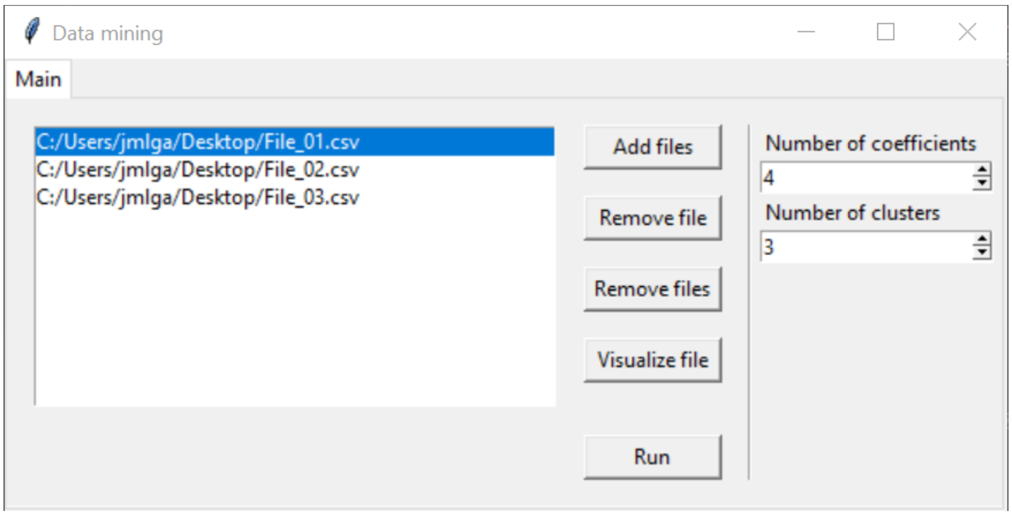


Figure 48: Visualization window. Final version.

The second application interface is shown in Figure 50.



C:/Users/jmlga/Desktop/File_01.csv
C:/Users/jmlga/Desktop/File_02.csv
C:/Users/jmlga/Desktop/File_03.csv

Number of coefficients: 4
Number of clusters: 3

Figure 50: Data mining application - Final version

The random time series generator tool has shown to be a very powerful tool as it allows to build many different types of time series under controlled assumptions by setting many different parameters. This makes possible the generation of synthetic time series in many different domains. Also, the seed functionality implemented in the last increment has enhanced this option and it makes it easier to achieve.

Given a specific sequence from any domain, the application provides the possibility of adding different components:

- Noise: in the form of random numbers (Main tab)
- Trend
- Spontaneous appearance of peaks or valleys
- Periodicity
- Seed

Thus, from a specific sequence, thousands can be created. In addition, it is possible to visualize the results instantly, obtaining a first view of how the application has performed.

On the other hand, the data mining application has shown that the generated time series with the same parameters keep statistical properties in common.

In Section 3.2.2.6, it is demonstrated that running the time series generator several times, changing the parameters (trend, periodicity…) in each run, and using all this generated time series as input to the clustering algorithm, it will most likely return all the time series grouped according to the generated source file. In other words, statistically speaking, the closest time series are those generated under the same parameter selection.

However, it is necessary to focus on how many coefficients the algorithm is using. A too small or too big number can lead to poor or exact representations of the time series, leading to bad results on the partitional clustering.

Thanks to the visualization of the time series along the Fourier transform, it is easy to see how the coefficients affect the result.

Personally, I am happy with the knowledge learned and the work done. The fact of having done the project from scratch and having fulfilled the requirements gives me confidence to tackle any project in my professional future.

Using an iterative and incremental methodology has been very positive and has allowed us to guide and complete the work in the correct line, offering small applications that have been used to test the different functionalities. I think it is a very flexible methodology that requires less initial effort for analysis and design. It also allows a continuous adapting to a changing vision of the project by the client (supervisors).

I also positively value the study made of some tools such as the random number generation carried out by NumPy, the Fourier analysis of NumPy... The idea of deeply understanding which algorithms internally use these APIs is fascinating and this approach will undoubtedly help me in my professional future.

Bibliography

Box, G., & E. Muller, M. (1958). *A Note on the Generation of Random Normal Deviates.*

C. Montgomery, D., & C. Runger, G. (2014). *Applied Statistics and Probability for Engineers* (6 ed.). Wiley.

Cardot, H., Peggy, C., & Jean-Marie, M. (2012, June). A fast and recursive algorithm for clustering large datasets with k-medians. *Computational Statistics & Data Analysis, 56*(6), 1434-1449.

Colin. (2011, September). *The Ziggurat Algorithm for Random Gaussian Sampling.* Retrieved from Heliosphan: https://heliosphan.org/zigguratalgorithm/zigguratalgorithm.html

Hunter, J. (2008). *Matplotlib*. Retrieved from https://matplotlib.org/users/history.html

James, J. (2011). *A Student's Guide to Fourier Transforms* (3 ed.). Cambridge University Press.

Kinderman, A., & Monahan, J. (1977). *ACM Transactions on Mathemaical Software.*

Knuth, D. (1997). *Seminumerical Algorithms. The Art of Computer Programming* (3 ed.).

MacKay, D. (2003). *Information Theory, Inference and Learning Algorithms.* Cambridge University Press.

Marsaglia, G. (2003). Xorshift RNG's. *Journal of Statistical Software*.

Marsaglia, G., & Bray, T. (1964). A Convenient Method for Generating Normal Variables. *SIAM Review, 6*(3), 260-264.

Matplotlib contributors. (n.d.). *Matplotlib*. Retrieved from https://matplotlib.org/

Matsumoto, M., & Nishimura, T. (1998). Mersenne Twister. *ACM Transactions on Modeling and Computer Simulation, 8*, 3-30.

Numpy contributors. (n.d.). *NumPy*. Retrieved from https://numpy.org/

O'Neill, M. E. (2014). *PCG: A Family of Simple Fast Space-Efficient Statistically.*

Panneton, F., L'Ecuyer, P., & Matsumoto, M. (2006, March). Improved Long-Period Generators Based on Linear Recurrences. *ACM Transactions on Mathematical Software, 32*, 1-16.

Pgfplots contributors. (n.d.). *Fourier Transform.* Retrieved from Pgfplots: http://pgfplots.net/media/tikz/examples/PNG/fourier-transform.png

PyInstaller contributors. (n.d.). *PyInstaller*. Retrieved from https://www.pyinstaller.org/

Python contributors. (n.d.). *Python*. Retrieved from https://docs.python.org/

Reay, D. (2012). Fast Fourier Transform. In D. Reay, *Digital Signal Processing with the OMAP-L138 eXperimenter* (pp. 212-223). New Jersey: John Wiley & Sons, Inc.

Roseman, M. (2019). *Modern tkinter for busy developers* (2 ed.). Late Afternoon Press.

Rousseeuw, P., & Kaufman, L. (1987). Clustering by means of Medoids. In *Statistical Data Analysis Based on the L1-Norm and RElated Methods* (pp. 405-416). Brussels.

Scikit-learn developers. (n.d.). *Scikit-learn*. Retrieved from https://scikit-learn.org/

Wikimedia Commons contributors. (n.d.). *Wikimedia Commons, the free media repository.* Retrieved from https://commons.wikimedia.org/

Wikipedia contributors. (n.d.). *Wikipedia, The Free Encyclopedia*. Retrieved from https://en.wikipedia.org/

Wolfram developers. (n.d.). *Wolfram.* Retrieved from https://reference.wolfram.com/