SynthATDelays: A Minimalist Python Package for the Generation of Synthetic Air Transport Delay Data

Carlson Moses Büth and Massimiliano Zanin *

Posted Date: 14 August 2025

doi: 10.20944/preprints202508.1091.v1

Keywords: air transport; delays; delay propagation; python package

 doi:10.20944/preprints202508.1091.v1

Article

SynthATDelays: A Minimalist Python Package for the Generation of Synthetic Air Transport Delay Data

Carlson Moses Büth [1] and Massimiliano Zanin [2,*]

[1] Institute for Cross-Disciplinary Physics and Complex Systems (IFISC), CSIC-UIB, Edifici Instituts Universitaris de Recerca, Campus UIB, 07122, Palma de Mallorca, Spain
[2] Institute for Cross-Disciplinary Physics and Complex Systems (IFISC), CSIC-UIB, Edifici Complex de Recerca de les Illes Balears, Parc Bit, 07120, Palma de Mallorca, Spain
* Correspondence: mzanin@ifisc.uib-csic.es; Tel.: +34-971-173290

Abstract

Within the endeavour of describing and analysing delays and their propagations in air transport, a major limitation is represented by the validation of the obtained results. While this can be overcome through synthetic models, those available in the literature mostly aim at simulating the system in a detailed and realistic way, resulting in high complexity and substantial computational costs. We here present *SynthATDelays*, a minimalist and modular Python package designed to simulate a virtual customisable air transport system and to provide synthetic delay data under tuneable conditions; it is thus designed to support the validation of data-based studies and pipelines. We describe its internal structure, and provide examples about how scenarios can be designed and executed.

Keywords: air transport; delays; delay propagation; python package

1. Introduction

In a way similar to virtually all fields of science and technology, in recent years data analysis has become an essential toolkit in the endeavour of describing and improving air transport. As a prototypical example, a large body of the literature has been devoted to data-based analyses of delays and their propagations, due to their relevance in the efficiency of the system, in its environmental impact, and in the customers' perceived value [1,2]. These include from the statistical characterisation of delays [3–7], to analyses aiming at describing relationships with external factors (e.g. adverse weather [8–11]), at extracting propagation patterns (e.g. through functional networks [12–16]), or at creating models able to forecast delays of future operations [17–20].

In spite of many undeniable successes, describing and analysing the dynamics of delays from real data entail an important limitation: the validation of the results. Given the inability to execute what-if scenarios or modify the system, researchers must trust that their findings accurately represent the underlying dynamics. To illustrate, suppose that a functional network analysis identifies a given airport as the main propagator of delays. In other scientific fields, such result would be validated (or rejected) by designing and performing a specific experiment under controlled conditions. Yet, if only real data can be used, it is highly impractical to verify whether adding resources to that airport would result in a dampened propagation—modifying the way an airport operates for the sake of science is unrealistic at best. Given this state of affairs, it is imperative to have a good understanding (and hence trust) of the data analysis pipeline. This can be achieved in two ways: through knowledge about the theoretical foundations of the used metrics, as e.g. under which conditions causality can be detected between time series; and by using synthetic data, which must be both realistic and tuneable.

The air transport community has a long history in developing models to simulate the dynamics of the system. From the scientific side, a handful of open-source agent-based models focusing on the macro-scale movements of aircraft and passengers have been proposed, as the one in Ref. [21] and Mercury [22]. Along this line, it is also worth mentioning Bluesky [23] and AirTrafficSim [24];

 doi:10.20944/preprints202508.1091.v1

compared to the previous ones, these focus on the microscale and allow testing hypotheses related to Air Traffic Control (ATC), while still being of potential usefulness in simulating the macroscopic dynamics. Finally, several commercial products are available, as e.g. the Total Airspace and Airport Modeler (TAAM), developed by Jeppesen; the Reorganized ATC Mathematical Simulator (RAMS), by EUROCONTROL; and the RAMS Plus, developed by ISA Software. All the above-mentioned examples share one commonality: they aim at providing the most realistic representation of the evolution of the system; in other words, given the real planned operations for one day, they try to yield the real outcome of that same day. This of course results in high complexity and computational costs.

In this contribution we present *SynthATDelays*, a Python package designed to produce synthetic delay information from highly tuneable scenarios. Compared to the aforementioned options, these scenarios are not aimed at mimicking the behaviour of the real system; but rather at testing, in a minimalist way, specific conditions and hypotheses, and how subsequent analyses are able to capture these. In other words, this package allows creating minimal toy models to test specific aspects of delay dynamics, including only the elements that are essential for the analysis, while avoiding (or, at least minimising) the complexity of the real system. To illustrate, the researcher can use this package to create a hypothetical system composed of a few airports, and generate time series representing the evolution of the average delays when changing their capacity. Similarly, different events and conditions can be simulated, e.g. the appearance of delays in specific routes, the dependence of different flights from the same crew, or the length of the buffer time between subsequent operations. This is achieved through a parsimonious yet modular structure, which allows for extensive customisation; but also at a reduced computational cost, enabling the batch analysis of numerous realisations in tens of seconds.

The remainder of the paper firstly describes the way simulations are performed (Sec. 2), including which options are available to the user, how individual operations are numerically simulated, and how results are organised. Next, Sec. 3 reports several examples of the use of the library: from a step-by-step tutorial, including the configuration of the initial flight network and the inclusion of different types of delays (Sec. 3.1); to different analyses of relevance in the context of functional network reconstruction (Secs. 3.2 and 3.3). Sec. 4 discusses some additional technical considerations, like how to install the package, its dependencies, and an analysis of the computational cost. Finally, Sec. 5 draws conclusions and proposes future development steps.

2. The Structure and Internal Logic of the Package

As depicted in Figure 1, the package is organised around three main blocks: the specification of the scenario to be executed, whose properties are centralised in a single class; its numerical simulation, handled by a single function; and the extraction of results at different levels of granularity. These three blocks are further described below.

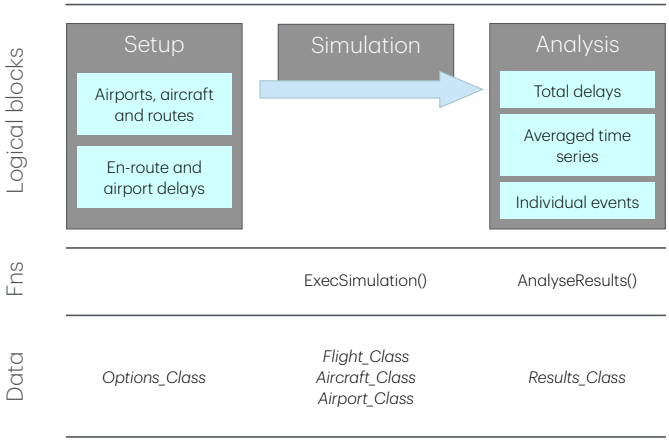

Figure 1. Logical organisation of the package: from left to right, the user starts by defining the scenario and its multiple options; for then executing the simulation; and extracting results at different granularity levels. The middle and bottom rows further reports the main involved functions, and the classes in which data are stored.

 doi:10.20944/preprints202508.1091.v1

2.1. Setting Up the Simulation

All information required to perform a simulation has to be introduced in an instance of the *Options_Class* class. This can be done either by manually populating it; or by using several predefined scenarios provided in the library—an example of this will be illustrated in Sec. 3. In what follows we provide an overview of the available options; additional details can be found in the documentation [25].

- Airports, including their number, their capacity (in operations per hour), and the flight time required to travel between them. Note that geographical information is not modelled; it is therefore possible to model non-Euclidean networks of flights.
- Aircraft, including their number, the minimum turnaround time between subsequent operations, and the buffer time left to recover delays.
- The route operated by each aircraft, defined as a list of airports that are visited sequentially. This allows, for instance, to define simple hub-and-spoke operations (e.g. $a \to b \to a$), more complex variations of the same (e.g. $a \to b \to a \to c \to a$), or triangular routes (e.g. $a \to b \to c \to a$).
- Delay generation. Delays can be defined to either affect flights on specific routes, or all flights landing on specific airports. The package provides several predefined functions to calculate their magnitude, including the use of delay profiles synthesised from real operations at major European airports; alternatively, the user can define custom ones - see the documentation for details and examples [25].
- Dependencies between routes. These model situations in which two flights operate different routes with one airport in common (e.g. $a \to b$ and $b \to c$), such that the latter cannot take off until the former has landed. This can thus be used to simulate the presence of connecting passengers, or the crew having to change aircraft. This further allows modelling the propagation of delays between airports that are not directly connected by a flight.
- Additional options, including e.g. the duration of nights, i.e. of a period in which flights cannot operate; or the number of days to be simulated.

2.2. The Simulation

The previously defined options are used as input for the main function performing the simulation (*ExecSimulation*, see the documentation [25]); this, in turn, is composed of two major phases.

The first phase entails defining the scheduling of all flights in the scenario. For each aircraft, its operations are created by following the corresponding route; the scheduled take-off time is calculated as the scheduled time of the previous landing, plus the turnaround time and the buffer time; conversely, the flight duration is obtained from the time distance between the corresponding pair of airports. Next, if so defined in the options, flights that should depart during the night hours are moved to the first hour of the morning. Note that, in this phase of the simulation, delays are not accounted for.

As a final point of the scheduling creation, pairs of flights operated by different aircraft are checked and eventually linked, to simulate dependencies due to connecting passengers or shared crews. This is done, with a probability defined by the user, whenever the departure of a flight operating between a and b is close in time to the landing of another flight from c to a—hence the latter aircraft is bringing the connecting passengers to a, and the former has to wait for them. This dependency between the two flights is stored for future use.

With this information ready, the simulation proper is executed, based on a discrete-event model with incremental time progression. At each time t, which is incremented in steps of one minute, the condition of each aircraft is updated according to the following rules:

- If the aircraft is idle, find its first scheduled flight not already executed; if its scheduled departure time is less than or equal to t, the aircraft is added to the airport queue, which is used to account for its limited capacity. The program also checks for dependencies, and the flight is activated only if the preceding one has already landed. The actual landing time is calculated by adding the distance between the departure and arrival airports, and any other en-route or airport delays defined in the options of the scenario. Finally, the status of the aircraft is updated to airborne.

 doi:10.20944/preprints202508.1091.v1

- If the aircraft is airborne and the current time t is equal or greater than the landing time, the status of the aircraft is changed to that corresponding to the turnaround process.
- Finally, if the aircraft is performing the turnaround and the time passed is greater than the minimum turnaround time, the status of the aircraft is changed back to idle, and the whole process repeats.

Note that the buffer time is implicitly used to reduce delays after landing, by being included in the scheduling but not in the real operations. To illustrate, suppose that both the minimum turnaround and buffer times are set to 60 minutes; and that the aircraft was scheduled to land at 15:00, but encountered a 30-minute delay. The subsequent departure would have been scheduled at 17:00. This is calculated as 15:00 with the addition of one hour each for turnaround and buffer. Consequently, the aircraft is prepared for departure at 16:30. This time is derived from the actual landing time of 15:30, incremented by the 60-minute minimum turnaround time. As a result, 30 minutes of the 60-minute buffer are utilised to mitigate the delay. This ensures that the next flight departs punctually.

Conversely, three conditions can cause delays to propagate: (i) when the delays are larger than the buffer time; (ii) when delays result in the concentration of operations in the same time window, thus surpassing the capacity of the airport; and (iii) when flights have dependencies between them, such that one of them has to wait for the arrival of a second one. By tuning the respective parameters (i.e. the buffer time, the airport capacity, and the probability of having dependencies), the researcher can manipulate the effective delay propagation in the system.

2.3. Analysis of the Results

Once the simulation is completed, its results can be accessed in two ways, depending on the granularity required by the subsequent analyses.

On the one hand, information about all executed flights is stored in an array of *Flight_Class* objects, including the corresponding (scheduled and actual) departure and landing times. These objects are further referenced in lists representing all aircraft and airports in the system, which can be accessed to obtain statistics at those levels. To illustrate, the user can extract the delay of all flights operated by an aircraft, or of all flights landing at a given airport.

On the other hand, the package provides a function to obtain macroscale time series that are commonly used in the context of delay analysis: average departure and landing delays across all airports, and number of landing and departure operations (*AnalyseResults*, see the documentation [25]). In all cases, the resolution of these time series can be customised, thus supporting multiscale analyses.

3. Examples

After this introduction on the structure of the package, we are here going to show how it can be used for research purposes. We start with a basic tutorial on the use of the different functions (Sec. 3.1), for then tackling two relevant questions: how delay propagation is modulated by the buffer time and by links between flights (Sec. 3.2), and the impact of using different metrics for detecting the propagation of delays (Sec. 3.3).

3.1. Step-by-Step Tutorial

In this section we are going to provide a step-by-step example of the use of the library; the interested reader will find additional details in the documentation of the same, including a set of tutorials that illustrate the main functions in a user-friendly way [25].

The simplest way of setting up a new simulation is through one of the provided scenarios, i.e. predefined self-contained configurations, which can later be tuned by the researcher according to their needs. To illustrate, we will start with the scenario named *Scenario_RandomConnectivity*, in which a set of airports are randomly connected by flights. The code, including initial imports, is reported in Listing 1.

 doi:10.20944/preprints202508.1091.v1

Listing 1. Basic initialisation code, using one of the predefined scenarios.

```
import synthatdelays as satd

Options = satd.Scenario_RandomConnectivity(
    numAirports = 6, numAircraft = 40, bufferTime = 0.25, seed = 0
)
```

This scenario accepts four parameters: the number of airports to be simulated (here set to 6), the number of aircraft connecting them (here 40), the buffer time (here 0.25 hours, i.e. 15 minutes), and the initial random number seed. The result, saved in the variable *Options*, is the instance of a class containing all the information to perform the simulation. To illustrate, accessing the variable Options.airportCapacity will yield the array array ([30., 30., 30., 30., 30., 30.])—in other words, all six airports are initialised with a maximum capacity of 30 operations per hour. This option object can already be passed to the main function performing the simulation, as shown in Listing 2.

Listing 2. Call to the function to execute the simulation, taking as input the options created in Listing 1.

```
(executedFlights, Airports, Aircraft) = satd.ExecSimulation(Options)
```

The output is composed of three lists: executedFlights, with individual flights that have been simulated; and Airports and Aircraft, respectively aggregating information for each airport and aircraft.

Before delving deeper into the results, let us add some delays. As previously explained, these can be of two types, respectively affecting routes or airports. Starting from the former, we can add uniformly distributed delays across all routes using the function ERD_Normal, as shown in Listing 3.

Listing 3. Code to define enroute delays, execute the simulation, and finally extracting high-level results about the delay evolution.

```
Options.enRouteDelay = []
Options.enRouteDelay_params = []
Options.enRouteDelay.append(satd.ERD_Normal)
Options.enRouteDelay_params.append([[-1], [-1], 0.0, 0.05])

(executedFlights, Airports, Aircraft) = satd.ExecSimulation(Options)

allResults = satd.AnalyseResults(
    (executedFlights, Airports, Aircraft), Options
)
```

The fourth line defines the parameters to be passed to the delay function, namely: the set of origin and destination airports defining those routes (with -1 indicating all airports); and the average and standard deviation of the normal distribution from which delays are drawn. Note that this function adds a delay proportional to the duration of each flight; hence, random value of 0.05 implies that the total duration in increased by a 5%.

With this defined, we can submit the results to the AnalyseResults function, which synthesises high-level time series from the individual operations—see last three lines of Listing 3. To illustrate, we can access the variable allResults.avgArrivalDelay[:, 0] to obtain a time series of the average arrival delay per hour in the first airport; and allResults.avgDepartureDelay[:, 0] for the equivalent at departure. Both time series are represented in the left panel of Figure 2. Note how departure delays are only positive, as by construction aircraft cannot take off before the scheduled time; and furthermore how, thanks to the buffer time, most of the arrival delays are absorbed before the next departure.

 doi:10.20944/preprints202508.1091.v1

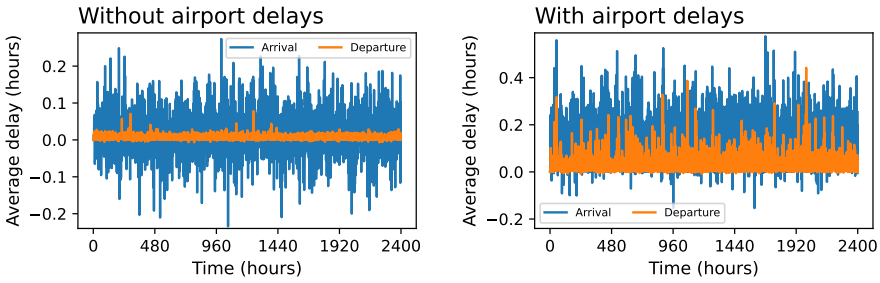

Figure 2. Graphical representation of the time series generated with the code described in Sec. 3. Left and right panels depict the results respectively without and with airport-based delays. Blue and orange lines correspond to arrival and departure average delays for the first airport.

We can next add some delays at a specific airport, in this case the first one, using the function AD_AbsNormal; as the name implies, delays are calculated as the absolute value of random numbers drawn from a normal distribution. The code for this is included in Listing 4.

Listing 4. Definition of airport-based delays.

```
Options.airportDelay = []
Options.airportDelay_params = []
Options.airportDelay.append(satd.AD_AbsNormal)
Options.airportDelay_params.append([[0], 0.05, 0.15])
```

The same time series are now represented in the right panel of Figure 2. As is to be expected, delays are now substantially larger and mostly positive; still, the buffer time is able to partially compensate for them.

3.2. What Is the Impact of the Buffer Time and of Links Between Flights?

The role of the buffer time is easily depicted, i.e. it must reduce the propagation of delays, and hence their total amount; yet, one may be interested in describing the exact transition, i.e. whether increasing the buffer time decreases the total delay linearly or otherwise. To achieve this aim, we here start from a random connectivity scenario of six airports and 80 aircraft; and with two sources of enroute delays, a Gaussian one across all routes, and an exponential one (see the function ERD_Disruptions) only affecting the route $a_1 \to a_2$ (a_1 and a_2 being two airports randomly chosen in each simulation). We further add links between flights, i.e. situations in which one of them cannot depart until a second one has arrived; this is done across all pairs of routes, and with a variable probability (between 0% and 20%).

Results, as a function of the buffer time and of the percentage of linked flights, are depicted in Figure 3. Specifically, the left panel reports the evolution of the total delay time, i.e. the sum of the landing delay of all flights; while the central and right panels report the number of functional links (i.e. pairs of airports between which a propagation is detected), as yielded by the Granger Causality and Transfer Entropy tests—details on these are included below. It can be appreciated that, as expected, the total delay drops with increasing buffer times; and that such drop is mostly linear. On the other hand, the number of functional links presents a sharper phase transition: it drops to zero (or at least, to a very small number) as soon as the buffer time surpasses a threshold. Finally, and also not surprisingly, both the total delay and the threshold are larger for increasing percentages of linked flights. Conclusions on these results will be drawn below.

 doi:10.20944/preprints202508.1091.v1

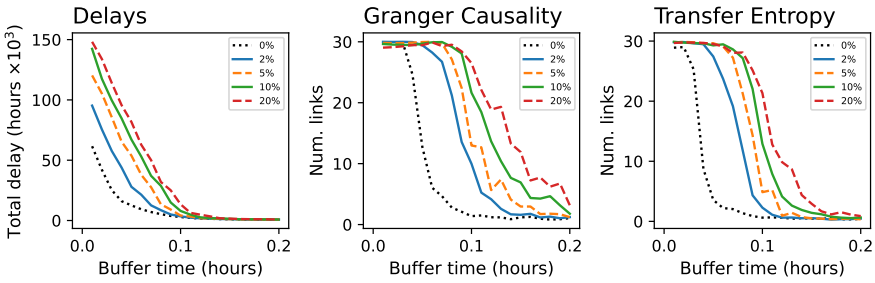

Figure 3. Evolution of the total landing delay (left panel), and number of functional links detected by the Granger Causality (centre) and the Transfer Entropy (right), as a function of the buffer time (in hours, X axis) and the percentage of linked flights (line colours and styles, see legends). See main text for the definition of the simulation scenario.

3.3. Which Functional Metric Ought to Be Used?

One of the open questions in the reconstruction of functional networks of delay propagations [12–16] is which functional metric should be used, among the many alternatives available in the literature. In other words, given two time series representing the evolution of delays at two airports a and b, the aim is to apply a metric on these time series able to detect whether the delays in a have an impact on b. Even when restricting to directional causality metrics, i.e. discarding correlation metrics and other non-directed tests, several alternatives are available, each one offering advantages and limitations. We here test a few options, by resorting to a minimal scenario composed of three airports (a, b and c), connected by a group of five aircraft operating between airports a and b, and by a similar group between b and c. Delays are added according to a Gaussian distribution, with airport a further experiencing higher delays between $11:00$ and $13:00$. Finally, all flights operating the segment $b \to c$ are linked to flights operating $a \to b$.

The objective of this scenario is to assess whether a propagation of delays is detected between airports a and c. As delays mostly appear on the former, but both airports are not connected by direct flights, the only source of propagation must be the linkage between flights—in other words, we are only measuring reactionary delays.

Functional links, i.e. potential propagation instances between a and c, are evaluated by applying the following metrics and tests on the average hourly arrival delay time series:

- Granger Causality (GC). The GC test was initially proposed to test for causal relations in economic time series, but has since then found wide-ranging applications in various scientific and technical fields [26,27]. It is based on the idea of "predictive causality" [28], i.e. situations in which past values of one time series provide statistically significant information about future values of another time series, beyond what is explained by the past values of the latter. The test compares two linear models: a restricted model that predicts future values using only its own past delays, and an unrestricted model that incorporates past values from the second time series. Whenever the latter has a higher prediction accuracy, it is said that the latter time series "Granger-causes" the former one. Note that, in spite of its popularity, it is strictly not a test for causality, as highlighted by Granger himself [29]; or, to use the words of Ref. [30], "Granger causality is designed to measure effect, not mechanism".
- Continuous Ordinal Patterns (COP). Method based on pre-processing the time series under analysis using Continuous Ordinal Patterns (COP) [31]; for then applying the same GC test as described above [32]. COP are patterns (here of length 4) that are compared against sub-windows of the original time series; they thus quantify the presence of specific non-linear structures, making these explicit for the GC test, and hence overcoming the linear nature of the latter.
- Transfer Entropy (TE). The TE from X to Y is defined as the amount of uncertainty reduced in the future values of Y by knowing the past values of X, after considering the past values of Y [33].

 doi:10.20944/preprints202508.1091.v1

Such uncertainty is calculated as an entropy, which is in turn calculated using two estimators of the underlying probability distributions.

1. Ordinal estimator: the probability distribution is obtained by mapping the time series into an ordinal space, created by "permutation patterns"—i.e. the rank order of values inside small sub-windows of the original series [34]. We here consider pattern lengths (also called embedding dimensions) of $D = 3$, 4 and 5.
2. Metric estimator: this approach, also called the Kozachenko-Leonenko estimator [35], uses a nearest-neighbours approach to estimate the entropy of a continuous random variable, as the expectation of the logarithm of the density. We here use $k = 4$, 6 and 8 nearest neighbours.

The implementation of this metric corresponds to the one included in the *infomeasure* Python package [36].

In all cases, the maximum lag, i.e. the maximum number of hours that may take for the delays to propagate, is set to four. Additionally, the TE values have been converted to p-values using permutation tests with 100 randomly shuffled time series—note that the GC test, and hence the COP one, natively yield p-values.

Results for all tests, in terms of the evolution of the median $\log_{10}$ of p-values, are depicted in Figure 4. Several interesting conclusions can be drawn. First of all, the selection of the parameter in the TE with the ordinal estimator has a major impact, defining whether the propagation is detected or not; yet, the same does not occur in the case of the metric estimator. Secondly, and not surprisingly, all tests benefit from longer time series—something that is challenging to achieve in the case of real data due to the non-stationarity of the system, as previously discussed in Ref. [37].

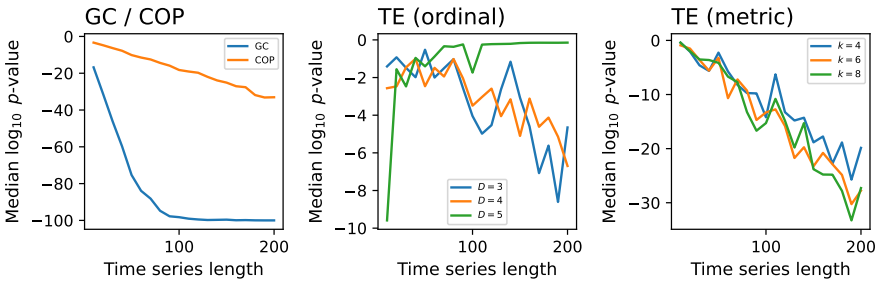

Figure 4. Evolution of the $\log_{10}$ of the p-values yielded by several functional metrics, for time series generated by the synthetic model described in Sec. 3.3. From left to right, the metrics include the GC and COP, and the TE with two different estimators - see main text for details. For the TE, different lines correspond to different values of the estimator parameter, see legend. Results are obtained as the median over 200 independent realisations.

Not less importantly, the GC test is the one yielding the smallest p-values. Inasmuch smaller p-values indicate stronger causal relationships, this result can have a two-fold explanation. On the one hand, the GC test has generally weaker requirements in terms of the minimum time series length; hence, the weak results of the TE may be due to a lack of data, but this cannot explain the weak results of COP. On the other hand, this may point to the fact that the underlying propagation phenomenon is a linear one; non-linear tests, as COP and TE, may be less suited for detecting it. This suggests an interesting hypothesis: while the propagation of delays between pairs of routes may be a linear process, the systemic propagation, i.e. as seen in the macroscale across multiple routes, becomes non-linear, as illustrated by the phase transitions of Figure 3. Notably, this hypothesis could here be considered thanks to the flexibility of the model; evaluating the same in real data would be a challenging task at best.

4. Additional Technical Considerations

The package is freely available both in the PyPI (https://pypi.org/project/synthatdelays/) and Conda-forge (https://github.com/conda-forge/synthatdelays-feedstock) repositories; instructions

 doi:10.20944/preprints202508.1091.v1

for its installation are available in the aforementioned links. It supports Python versions 3.11 and above, including the latest Python 3.14, and only uses standard external libraries—Numpy [38] and Statsmodels [39] being its only dependencies. The code has extensively been tested, with a coverage (at the time of writing) of 99%. The source code is freely available in a GitLab repository (https://gitlab.com/MZanin/synth-at-delays), alongside documentation and feedback tools—the reader is encouraged to submit bugs and improvement requests through it.

As previously mentioned, one of the advantages of this package is the reduced computational cost necessary to perform a simulation. To illustrate this, Figure 5 reports the time required to rum a complete simulation—results were obtained using a single core of a 3.8 GHz Intel Core i7 processor. It can be appreciated that the run time scales almost linearly with the number of flights, while the number of airports has a negligible impact. Notably, small size simulations only require few seconds to complete, thus making complete statistical characterisations viable even in standard laptops.

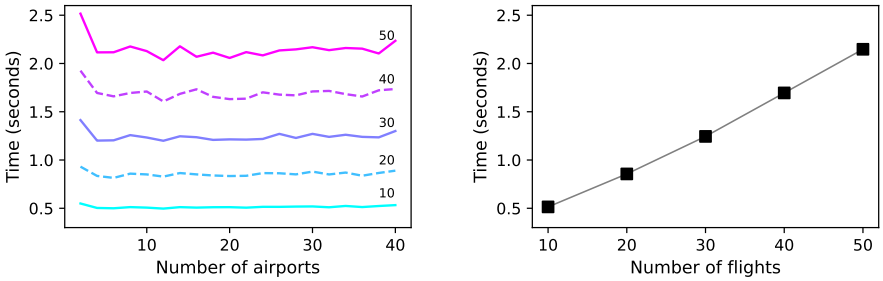

Figure 5. Analysis of the computational cost. The left panel reports the average time required to run a full simulation as a function of the number of aircraft (different lines, see text on top of them) and of airports (X axis). The right panel reports the same information as a function of the number of flights; each point is thus the average of a curve on the left panel. Results correspond to the average over 20 independent realisations.

5. Discussion and Conclusions

In this contribution we presented and described *SynthATDelays*, a Python package designed to generate realistic, yet minimalist, synthetic delay data of an air transport system. It provides a tool to establish lab conditions for working with air traffic delay data, thus filling the missing safe ground between delay data and their analysis, and facilitating the verification of analysis programs and scientific workflows. This approach enables the empirical testing of hypotheses concerning aircraft delays, as well as our capacity to articulate such delays and the underlying causes. In essence, it contributes to the scientific robustness of associated data-based analyses.

Note that the characteristics and functionalities here described correspond to version 1.0.0 of the package; and that these may have evolved at the time of reading. We are specifically planning to extend the customisation options available to the user, including tuneable buffer times per airport and routes; and tuneable aircraft performances. We further invite the community to submit suggestions and feature requests through the GitLab repository.

Author Contributions: Conceptualization, Massimiliano Zanin; Methodology, Carlson Büth; Software, Carlson Büth and Massimiliano Zanin; Supervision, Massimiliano Zanin; Writing - original draft, Carlson Büth and Massimiliano Zanin; Writing - review & editing, Carlson Büth and Massimiliano Zanin. All authors have read and agreed to the published version of the manuscript.

Funding: This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 851255). This work was partially supported by the María de Maeztu project CEX2021-001164-M funded by the MICIU/AEI/10.13039/501100011033 and FEDER, EU.

Data Availability Statement: The software library described in this work is freely available at https://gitlab.com/MZanin/synth-at-delays, alongside with code to reproduce the shown examples.

Use of Artificial Intelligence: AI or AI-assisted tools were not used in drafting any aspect of this manuscript.

 doi:10.20944/preprints202508.1091.v1

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Carlier, S.; De Lépinay, I.; Hustache, J.C.; Jelinek, F. Environmental impact of air traffic flow management delays. In Proceedings of the 7th USA/Europe air traffic management research and development seminar (ATM2007). Citeseer, 2007, Vol. 2, p. 16.
2. Peterson, E.B.; Neels, K.; Barczi, N.; Graham, T. The economic cost of airline flight delay. *Journal of Transport Economics and Policy (JTEP)* **2013**, *47*, 107–121.
3. Cao, Y.; Zhu, C.; Wang, Y.; Li, Q. A method of reducing flight delay by exploring internal mechanism of flight delays. *Journal of Advanced Transportation* **2019**, *2019*, 7069380.
4. Wang, Y.; Cao, Y.; Zhu, C.; Wu, F.; Hu, M.; Duong, V.; Watkins, M.; Barzel, B.; Stanley, H.E. Universal patterns in passenger flight departure delays. *Scientific reports* **2020**, *10*, 6890.
5. Mitsokapas, E.; Schäfer, B.; Harris, R.J.; Beck, C. Statistical characterization of airplane delays. *Scientific Reports* **2021**, *11*, 7855.
6. Wang, Z.; Liao, C.; Hang, X.; Li, L.; Delahaye, D.; Hansen, M. Distribution prediction of strategic flight delays via machine learning methods. *Sustainability* **2022**, *14*, 15180.
7. Olivares, F.; Zanin, M. Quantifying Deviations from Gaussianity with Application to Flight Delay Distributions. *Entropy* **2025**, *27*, 354.
8. Schultz, M.; Lorenz, S.; Schmitz, R.; Delgado, L. Weather impact on airport performance. *Aerospace* **2018**, *5*, 109.
9. Zanin, M.; Zhu, Y.; Yan, R.; Dong, P.; Sun, X.; Wandelt, S. Characterization and prediction of air transport delays in China. *Applied Sciences* **2020**, *10*, 6165.
10. de Oliveira, M.; Eufrásio, A.B.R.; Guterres, M.X.; Murça, M.C.R.; de Arantes Gomes, R. Analysis of airport weather impact on on-time performance of arrival flights for the Brazilian domestic air transportation system. *Journal of Air Transport Management* **2021**, *91*, 101974.
11. Rodríguez-Sanz, Á.; Cano, J.; Rubio Fernandez, B. Impact of weather conditions on airport arrival delay and throughput. *Aircraft engineering and aerospace technology* **2022**, *94*, 60–78.
12. Zanin, M. Can we neglect the multi-layer structure of functional networks? *Physica A: Statistical Mechanics and its Applications* **2015**, *430*, 184–192.
13. Pastorino, L.; Zanin, M. Air delay propagation patterns in Europe from 2015 to 2018: An information processing perspective. *Journal of Physics: Complexity* **2021**, *3*, 015001.
14. Zhang, X.; Zhao, S.; Mei, H.; et al. Analysis of airport risk propagation in Chinese air transport network. *Journal of Advanced Transportation* **2022**, *2022*.
15. Pastorino, L.; Zanin, M. Local and Network-Wide Time Scales of Delay Propagation in Air Transport: A Granger Causality Approach. *Aerospace* **2023**, *10*, 36.
16. Chen, S.; Du, W.; Liu, R.; Cao, X. Finding spatial and temporal features of delay propagation via multi-layer networks. *Physica A: Statistical Mechanics and its Applications* **2023**, *614*, 128526.
17. Rebollo, J.J.; Balakrishnan, H. Characterization and prediction of air traffic delays. *Transportation research part C: Emerging technologies* **2014**, *44*, 231–241.
18. Monmousseau, P.; Delahaye, D.; Marzuoli, A.; Féron, E. Predicting and analyzing US air traffic delays using passenger-centric data-sources. In Proceedings of the ATM 2019, 13th USA/Europe Air Traffic Management Research and Development Seminar, 2019.
19. Liu, Y.; Liu, Y.; Hansen, M.; Pozdnukhov, A.; Zhang, D. Using machine learning to analyze air traffic management actions: Ground delay program case study. *Transportation Research Part E: Logistics and Transportation Review* **2019**, *131*, 80–95.
20. Huynh, T.K.; Cheung, T.; Chua, C. A systematic review of flight delay forecasting models. In Proceedings of the 2024 7th International Conference on Green Technology and Sustainable Development (GTSD). IEEE, 2024, pp. 533–540.
21. Grether, D.; Fürbas, S.; Nagel, K. Agent-based modelling and simulation of air transport technology. *Procedia Computer Science* **2013**, *19*, 821–828.
22. Delgado, L.; Gurtner, G.; Weiszer, M.; Bolic, T.; Cook, A. Mercury: an open source platform for the evaluation of air transport mobility. *13th SESAR Innovation Days* **2023**.
23. Hoekstra, J.M.; Ellerbroek, J. Bluesky ATC simulator project: an open data and open source approach. In Proceedings of the Proceedings of the 7th international conference on research in air transportation. FAA/Eurocontrol Washington, DC, USA, 2016, Vol. 131, p. 132.

 doi:10.20944/preprints202508.1091.v1

24. Hui, K.Y.; Nguyen, C.H.; Lui, G.N.; Liem, R.P. AirTrafficSim: An open-source web-based air traffic simulation platform. *Journal of Open Source Software* **2023**, *8*, 4916. https://doi.org/10.21105/joss.04916.
25. SynthATDelays Documentation. https://gitlab.com/MZanin/synth-at-delays/-/wikis/Home/. Accessed: 2025-08-15.
26. Granger, C.W. Investigating causal relations by econometric models and cross-spectral methods. *Econometrica: journal of the Econometric Society* **1969**, pp. 424–438.
27. Shojaie, A.; Fox, E.B. Granger causality: A review and recent advances. *Annual Review of Statistics and Its Application* **2022**, *9*, 289–319.
28. Diebold, F.X. *Elements of forecasting*; South-Western College Pub. Cincinnati, OH, USA, 1998.
29. Granger, C.W. Causality, cointegration, and control. *Journal of Economic Dynamics and Control* **1988**, *12*, 551–559.
30. Barrett, A.B.; Barnett, L. Granger causality is designed to measure effect, not mechanism. *Frontiers in neuroinformatics* **2013**, *7*, 6.
31. Zanin, M. Continuous ordinal patterns: Creating a bridge between ordinal analysis and deep learning. *Chaos: An Interdisciplinary Journal of Nonlinear Science* **2023**, *33*.
32. Zanin, M. Augmenting granger causality through continuous ordinal patterns. *Communications in Nonlinear Science and Numerical Simulation* **2024**, *128*, 107606.
33. Schreiber, T. Measuring information transfer. *Physical review letters* **2000**, *85*, 461.
34. Bandt, C.; Pompe, B. Permutation entropy: a natural complexity measure for time series. *Physical review letters* **2002**, *88*, 174102.
35. Kozachenko, L. Sample estimate of the entropy of a random vector. *Probl. Pered. Inform.* **1987**, *23*, 9.
36. Büth, C.M.; Acharya, K.; Zanin, M. Infomeasure: A Comprehensive Python Package for Information Theory Measures and Estimators. *Scientific Reports* **2025**, *15*, 29323. https://doi.org/10.1038/s41598-025-14053-5.
37. Acharya, K.; Olivares, F.; Zanin, M. How representative are air transport functional complex networks? A quantitative validation. *Chaos: An Interdisciplinary Journal of Nonlinear Science* **2024**, *34*.
38. Harris, C.R.; Millman, K.J.; Van Der Walt, S.J.; Gommers, R.; Virtanen, P.; Cournapeau, D.; Wieser, E.; Taylor, J.; Berg, S.; Smith, N.J.; et al. Array programming with NumPy. *nature* **2020**, *585*, 357–362.
39. Seabold, S.; Perktold, J.; et al. Statsmodels: econometric and statistical modeling with python. *SciPy* **2010**, *7*, 92–96.