Research paper

PyRQA—Conducting recurrence quantification analysis on very long time series efficiently

Tobias Rawald[a,b,*], Mike Sips[b], Norbert Marwan[c]

[a] Humboldt Universität zu Berlin, Germany
[b] GFZ German Research Centre for Geosciences, Potsdam, Germany
[c] Potsdam Institute for Climate Impact Research, Potsdam, Germany

A R T I C L E   I N F O

A B S T R A C T

PyRQA is a software package that efficiently conducts recurrence quantification analysis (RQA) on time series consisting of more than one million data points. RQA is a method from non-linear time series analysis that quantifies the recurrent behaviour of systems. Existing implementations to RQA are not capable of analysing such very long time series at all or require large amounts of time to calculate the quantitative measures. PyRQA overcomes their limitations by conducting the RQA computations in a highly parallel manner. Building on the OpenCL framework, PyRQA leverages the computing capabilities of a variety of parallel hardware architectures, such as GPUs. The underlying computing approach partitions the RQA computations and enables to employ multiple compute devices at the same time. The goal of this publication is to demonstrate the features and the runtime efficiency of PyRQA. For this purpose we employ a real-world example, comparing the dynamics of two climatological time series, and a synthetic example, reducing the runtime regarding the analysis of a series consisting of over one million data points from almost eight hours using state-of-the-art RQA software to roughly 69 s using PyRQA.

1. Introduction

Recurrence quantification analysis (RQA) is a method from non-linear time series analysis to quantify the recurrent behaviour of systems (Marwan et al., 2007). It has been successfully applied to characterise earthquake dynamics (Chelidze and Matcharashvili, 2015), spatially extended ecosystems (Proulx et al., 2009; Li et al., 2008), or climate variability (Zhao and Li, 2011), to identify nonlinear regimes and complex synchronization in solar variability (Kurths et al., 1994; Zolotova et al., 2009), to investigate past climate teleconnections (Marwan et al., 2003) and regime transitions (Donges et al., 2011).

RQA relies on the identification of line structures within *recurrence matrices*. Such a matrix is constructed by computing the mutual similarities of multi-dimensional vectors reconstructed from a given time series. Pairs of vectors are either considered similar or dissimilar, resulting in a binary decision. Matrix elements of the same value form line structures. Based on frequency distributions of line lengths, quantitative measures such as the *average line length* are derived.

The underlying algorithms of RQA have a time complexity of $O(N^2)$. Persisting the binary similarity matrix in the main memory of a compute system results in a quadratic space complexity as well.

These properties hamper an efficient analysis of time series consisting of more than one million data points. Although short time series are often typical in Geoscience, their size will grow strongly in the future by the increasing effort and success of collecting data and increasing temporal resolution.

Existing implementations of RQA suffer from one or multiple limitations, which hamper the analysis of very long time series. This includes the *memory limitation* (not being able to store matrices exceeding the size of the main memory), the *device limitation* (not being able to employ more than one compute device), and the *runtime limitation* (not being able to conduct the analysis in a reasonable amount of time).

PyRQA addresses those limitations by introducing concepts from parallel and distributed computing. The underlying computing approach subdivides the binary similarity matrix into multiple sub matrices. Each sub matrix is processed by a compute device in a massively parallel manner. Relying on the *OpenCL* framework, a variety of different hardware architectures, e.g., *graphics processing units* (GPUs), can be employed for processing. Furthermore, the computing approach enables to process multiple sub matrices by multiple compute devices at the same time. As will be shown in

---

* Corresponding author at: GFZ German Research Centre for Geosciences, Potsdam, Germany.
  E-mail addresses: tobias.rawald@gfz-potsdam.de (T. Rawald), mike.sips@gfz-potsdam.de (M. Sips), marwan@pik-potsdam.de (N. Marwan).

Section 8.2, those properties allow to reduce the runtime of processing a series consisting of over one million data points from almost eight hours to 69 s.

PyRQA is a free and open-source software package (Rawald and Sips, 2016a). Its computing approach has been successfully applied to a real-world time series from climate impact research that consists of more than one million data points (Rawald et al., 2014). Here, the runtime of conducting the analysis could be drastically reduced by performing the computations on multiple GPUs. In Rawald et al. (2015) it is evaluated, how the application of concepts from database engineering influence the performance of RQA processing. In the following, we focus on the features of *PyRQA* and the usage of its *application programming interface* (API).

The manuscript is structured as follows. Section 2 gives a short introduction to the basic concepts of RQA. Section 3 describes state-of-the-art RQA software, focussing on their features and limitations. In Section 4, the computing approach employed by *PyRQA* is examined in detail. Section 5 presents information on how to obtain the source code of *PyRQA* and the license under which it is released. In Section 6, the installation process is described. Section 7 outlines the analysis work-flow induced by *PyRQA*. In Section 8, comprehensive examples are presented, focusing on the features and benefits of *PyRQA*. In Section 9, a conclusion and an outlook on the future development of *PyRQA* is given.

2. Basic concepts of RQA

Environmental systems such as the Earth's climate system show dynamic behaviour, which is highly nonlinear. This behaviour can be observed by monitoring variables, e.g., the air temperature at a specific location. Recurrence analysis is applied, to identify patterns within the temporal dynamics of an environmental system. We refer to Marwan et al. (2007) for more detailed information on the topic of recurrence analysis.

Recurrence analysis comprises several methods, including the *recurrence plot*. It is based on the reconstruction of multi-dimensional vectors from a given time series using time-delay embedding (Packard et al., 1980). Those vectors represent the states of the system under investigation. The mutual similarity regarding all pairs of reconstructed vectors is determined by applying measures, such as the L_2-norm. Two vectors are either considered to be similar or dissimilar based on a neighbourhood condition, e.g., a fixed similarity threshold. The corresponding results are captured within a quadratic *recurrence matrix*.

Matrix elements representing pairs of similar vectors are referred to as *recurrence points*, implying that the system recurs to a similar state. A recurrence plot is the visual representation of a recurrence matrix, encoding recurrence points as black dots and the remaining elements as white dots. These dots form small-scale structures, in particular lines. Visually inspecting those structures allows to draw conclusions regarding the temporal dynamics of a system.

A recurrence plot is only applicable for time series consisting of hundreds of data points, due to screen size limitations. Only parts of the plot can be displayed, if the number of data points is increased to hundreds of thousands. Furthermore, applying downsampling strategies may create artifacts within a recurrence plot, causing false interpretations (Marwan, 2011). Referring to the information captured within a recurrence plot, recurrence quantification analysis is a method to quantitatively assess its visual impression (Zbilut and Webber Jr., 2007). This quantification was later connected to theoretical understanding (Marwan et al., 2007).

RQA quantifies the line structures within the recurrence plot consisting of recurrence and non-recurrence points. It considers three different types of lines, each of them assigned with specific semantics:

1. diagonal lines (recurrence points),
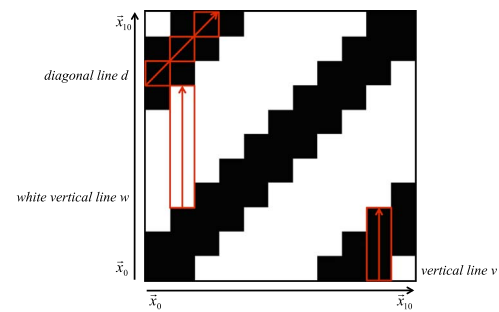2. vertical lines (recurrence points), and

Fig. 1. Basic RQA Concepts. The recurrence plot displayed is based on a time series created by observing an oscillating system. Eleven vectors, $\vec{x}_0$ to $\vec{x}_{10}$, are reconstructed from the series. A diagonal line d of length 3, a vertical line v of length 3 and a white vertical line w of length 5 are highlighted.

3. white vertical lines (non-recurrence points).

For each of those types, frequency distributions of their occurrences are computed. Based on these distributions, quantitative measures are derived, e.g., the portion of recurrence points that form diagonal lines (*determinism*, *DET*). The set of quantitative measures describes the temporal dynamics of a system under investigation. Unless stated otherwise, in the following the expression "diagonal and vertical lines" refers to all three line types. The basic concepts behind RQA are depicted in Fig. 1. The definition of the RQA measures can be found in Marwan et al. (2007).

3. State-of-the-art in RQA software

In the following, an overview of software that allows to conduct RQA or create recurrence plots is given.[1] The focus lies on implementations that are freely available as well as open-source and do not rely on proprietary software, such as MATLAB. Tools are distinguished regarding their functionality and their computational limitations.

Memory limitation:	The recurrence matrix does not fit into the main memory of the computing system. Matrices exceeding the size of the main memory can not be analysed as a whole or can not be analysed at all.
Device limitation:	The quantitative analysis is conducted on a single CPU. The computational capabilities of systems containing multiple CPUs or accelerators, such as GPUs, are not exploited.
Runtime limitation:	The quantitative analysis consumes large amounts of time, due to the limited exploitation of parallel processing capabilities.

The RQA software presented in the following suffers from specific limitations regarding the analysis of very long time series, consisting of more than one million data points. Table 1 maps each software tool to its limitations.

3.1. RQA X

RQA X (Keller, 2016) supports the construction of recurrence matrices based on a fixed radius neighbourhood. It allows to employ a variety of similarity measures, such as the L_2-norm and the L_∞-norm, to conduct the pairwise comparison of vectors. Regarding the quantitative analysis, RQA X does not provide any information regarding white vertical lines. The software is written in *Objective C* and offers

---

[1] A continuously maintained list of software is available at http://www.recurrence-plot.tk/programmes.php.

Table 1

Limitations of Existing RQA Software. Each software tool is mapped to its specific limitations. The tools that suffer from memory limitation do not necessarily consume large amounts of runtime, since the size of recurrence matrices is restricted, such that they are not capable of analysing very long time series as a whole. Nonetheless, they only make limited use of parallel processing techniques. Note, the tool recurr that is part of TISEAN is only capable of creating recurrence plots and does not support RQA.

Software Tool	Memory limitation	Device limitation	Runtime limitation
RQA X	✓	✓	(✓)
TISEAN (recurr)		✓	✓
crqa	✓	✓	(✓)
pyunicorn	✓	✓	(✓)
Commandline Recurrence Plots		✓	✓

similar functionality as *RQA Software* (Webber Jr., 2016).

RQA X is only capable of analysing recurrence matrices created from up to 40,000 vectors.[2] To compensate this restriction, it is possible to specify *epochs*, which are fixed-sized windows along the middle diagonal of the full recurrence matrix (Webber Jr. and Zbilut, 2005, p. 52).

RQA X performs its computations solely on CPU devices. It allows to conduct multiple quantitative analyses at the same time, by performing each analysis in a separate CPU thread.

3.2. Tisean

TISEAN, an acronym derived from *time series analysis*, is a collection of command line tools released under *GPL* license, which allow to analyse numerical time series. Version 3.0.1 comprises utilities, e.g., for generating time series and performing noise reduction as well as conducting linear and nonlinear time series analysis (Hegger et al., 1999).

TISEAN includes two versions of the program recurr, one written in C and one written in FORTRAN, to compute the content of recurrence matrices. recurr computes the pairwise distances of the reconstructed vectors using the fixed radius neighbourhood in combination with the L_∞-norm. The output of recurr is a list of recurrence points, represented as pairs of integer values that may either be written to *stdout* or stored in a file. recurr does not provide any functionality to conduct RQA based on those recurrence points.

3.3. crqa

crqa is a package written in *R* and released under *GPL* license. It allows to conduct *cross recurrence quantification analysis* (cRQA) (Coco and Dale, 2014). The crqa package is partially based on the *Cross Recurrence Plot Toolbox* implemented in MATLAB (Marwan, 2016b). The method cRQA differs from traditional RQA by comparing the recurrent behaviour captured in two time series. The recurrence vectors reconstructed from each time series are assigned to one of the two axis of the recurrence matrix. The determination of the pairwise similarities as well as the detection of line structures is similar.

Among others, the package crqa provides the function crqa, which performs cRQA on two input time series. It can imitate traditional RQA by providing the same input time series twice. crqa computes the recurrence plot as well as nine quantitative measures, e.g., *recurrence rate* and *determinism*. The function constructs a cross recurrence matrix based on a fixed radius neighbourhood.

The data structures employed by crqa persist the recurrence matrix in the main memory of the computing system. Hence, it is only

capable of analysing recurrence matrices that fit into main memory, limiting the length of the time series to investigate to a couple of thousands. crqa uses parallelisation techniques by executing multiple CPU threads while constructing and analysing the recurrence matrix.

3.4. pyunicorn

pyunicorn is a Python package released under *BSD* license that conducts complex network as well as recurrence analysis (Donges et al., 2015). Amongst others, it allows to perform RQA and to create recurrence plots. To improve the efficiency of the computations, it includes code fragments that are written in C, C++ and FORTRAN.

pyunicorn persists the recurrence matrix within the main memory during the detection of diagonal and vertical lines. Depending on the size of the main memory available, this property limits the length of the time series that can be processed to only a couple of thousands of data points. This renders the analysis time series consisting of over one million data points impossible.

3.5. Commandline recurrence plots

Commandline Recurrence Plots allows to compute recurrence plots and to conduct recurrence quantification analysis (Marwan, 2016a). Version 1.13z of the tool can be obtained in compiled form for a variety of platforms, including Linux, Mac OS X, Windows, HP-UX and Solaris. Note, its source code is not publicly available.

The focus of Commandline Recurrence Plots is conducting RQA. The identification of line structures is performed without storing the recurrence matrix in the memory of the computing system. The similarity values referring to pairs of vectors are rather computed on the fly, while sequentially inspecting the elements within diagonals and columns of the matrix. This allows to analyse recurrence matrices of almost arbitrary size.

Commandline Recurrence Plots conducts the computations solely in a single CPU thread. This does only employ a fraction of the computing capabilities provided by multi-core CPUs.

4. Distributed and parallel computing approach

We present insights on the underlying computing approach of *PyRQA*. The focus lies on the RQA processing, although it also supports the creation of recurrence plots. *PyRQA* overcomes the limitations of existing RQA software, as described in Section 3, by:

- subdividing the full recurrence matrix into multiple sub matrices that fit into the memory,
- distributing the processing of the sub matrices across multiple compute devices, and
- conducting the computations within a single sub matrix in a massively parallel manner.

In the following, those aspects are explained in detail.

4.1. Divide & recombine

Recurrence matrices exceeding the size of the memory available cannot be stored in their entirety. The computing approach of *PyRQA* applies the concept *Divide & Recombine* (Guha et al., 2012) to enable their processing. The recurrence matrix is subdivided into a set of sub matrices. The calculation of the pairwise vector similarities as well as the detection of line structures is performed for each sub matrix individually. The separate sub matrix results are recombined into global data structures, which serve as the basis for computing the RQA measures.

Lines may cross the vertical and horizontal borders of adjacent sub matrices. *PyRQA* employs additional data structures, to ensure their

---

[2] This value is hardcoded in the file RQAPrefsController.m as maximumBatchWindowSize.

correct detection. The *carryover buffers* store the length of lines that reach the outer borders of the sub matrix currently inspected. An individual carryover buffer is provided for each line type. Those intermediate line lengths are used as an input for the line detection in adjacent sub matrices. The overhead for storing and maintaining those carryover buffers is marginal.

Dividing the full recurrence matrix has two major benefits. First, it overcomes the *memory limitation*, since the size of the sub matrices can be chosen such that they fit into the memory available. Second, the processing of sub matrices can be distributed across multiple compute devices, thus there is no *device limitation*.

More information on the functionality of the carryover buffers as well as a specific sub matrix processing order can be found in Rawald et al. (2014).

4.2. Massively parallel sub matrix processing

The software tools presented in Section 3 only use the parallel processing capabilities provided by a CPU or none at all. This neglects the fact that the quantitative analysis can be subdivided into a set of distinct operators, where each operator itself can be processed in a massively parallel manner. Those operators are applied to each sub matrix individually.

The computing approach of *PyRQA* distinguishes between the operators:

  I. *create_recurrence_matrix*,
 II. *detect_diagonal_lines*, and
III. *detect_vertical_lines*.

There is the constraint that the *create_recurrence_matrix* operator has to be executed before the line detection operators are started. Each operator can be mapped to a type of atomic tasks:

  I. the similarity comparison of a single pair of reconstructed vectors,
 II. the inspection of a single diagonal of the recurrence matrix regarding diagonal lines, and
III. the inspection of a single column of the recurrence matrix regarding vertical and white vertical lines.

Each atomic task is fully independent of any other task of the same operator. This property allows to execute multiple tasks of the same operator in parallel. Each operator is assigned with a maximum degree of parallelism (DOP_{max}), based on the number of reconstructed vectors N:

  I. $DOP_{max} = N^2$,
 II. $DOP_{max} = 2N - 1$ (non-symmetric recurrence matrix) / $DOP_{max} = N - 1$ (symmetric recurrence matrix), and
III. $DOP_{max} = N$.

DOP_{max} captures the maximum number of tasks that can run in parallel per operator. This parallel execution contributes to overcoming the *runtime limitation* by maximising the utilisation of the computing resources of a single compute device.

4.3. Technical details

PyRQA employs the OpenCL framework for heterogeneous computing (Stone et al., 2010). OpenCL is designed to exploit the parallel computing capabilities of multi-core devices, such as CPUs, and many-core devices, such as GPUs. We have chosen OpenCL, because it is supported by a variety of compute devices from different hardware vendors.

The processing of OpenCL is subdivided between a single *host device* and one or more *compute devices*. The source code of *PyRQA*

consists of a *host program* written in *Python* and *kernel functions* written in *OpenCL C*. The kernel functions capture the atomic tasks of each operator and are executed by the compute devices.

Detailed information regarding the performance of a set of RQA implementations using OpenCL running on compute devices from different hardware vendors is provided by Rawald et al. (2015).

5. Source code and license

PyRQA is distributed via the *Python Package Index* (Rawald and Sips, 2016a). Its contents are free of charge as well as open-source and released under version 2.0 of the *Apache License*. The source files of version 0.1.0 can be downloaded at Rawald and Sips (2016d). It is planned to open the existing GitLab repository for public access in the near future (Rawald and Sips, 2016c).

6. Installation

PyRQA can be installed using the command line tool *pip* (PyPA, 2016): `pip install PyRQA`

The *PyRQA* package defines dependencies regarding additional Python packages that are required for execution, including *NumPy* (Numpy Developers, 2016), *PyOpenCL* (Klöckner, 2016), and *Pillow* (Lundh and Contributors, 2016). Using *pip*, these dependencies are installed automatically.

PyRQA may require the installation of OpenCL related software, such as custom hardware drivers and an OpenCL runtime. The amount of software to be installed is device-specific and varies between hardware vendors:

A. http://developer.amd.com/tools-and-sdks/opencl-zone
B. http://software.intel.com/en-us/articles/opencl-drivers⟩
C. http://developer.nvidia.com/opencl⟩

Apple as a key driver of OpenCL provides support within the latest versions of *Mac OS X* by default. If *PyRQA* is unable to detect any OpenCL compute device, exceptions are thrown during its execution.

7. PyRQA analysis workflow

This section describes the typical workflow of using *PyRQA* as well as the relevant package contents. An analysis is structured into four processing steps:

1. retrieving a time series to investigate,
2. assigning values to the analysis parameters,
3. creating the analysis computation, and
4. retrieving the final computing results.

Table 2
Overview of object classes. For each processing step, the object classes used in the following examples are captured. For each object class, a short description is provided. A complete documentation of all object classes of *PyRQA* is given by Rawald and Sips (2016b).

Step	Object class	Description
1.	FileReader	Extract time series from file
2.	Settings	Set of analysis parameters
	EuclideanMetric	Similarity measure
	FixedRadius	Neighbourhood condition
3.	RQAComputation	RQA computation
	RecurrencePlotComputation	Recurrence plot computation
	OpenCL	OpenCL environment
4.	RQAResult	RQA result
	RecurrencePlotResult	Recurrence plot result

Those steps are similar for conducting RQA as well as creating a recurrence plot. Note, the example source code displayed in the following refers to a RQA scenario.

Adhering to the object-oriented programming approach, *PyRQA* comprises software components that encapsulate the functionality used in each processing step. A comprehensive documentation of the *PyRQA* API is provided by Rawald and Sips (2016b). An overview of the object classes mentioned in following is given in Table 2.

7.1. Retrieving a time series to investigate

A common format to represent time series data are text files containing values separated by delimiters, such as *comma-separated values* (CSV). Each separated column within such a file by convention represents a time series referring to a specific observational variable. The object class `FileReader` provides means to extract such series from delimiter-separated files (see List. 1).

The static method `file_as_float_array` reads a column from an input file and transforms it into an array of floating point values. The `delimiter` refers to a single character that separates the individual columns. The `column`, from which the time series should be extracted, is referred to by an integer, starting at zero. It is possible to omit a fixed number of lines at the beginning of the file, by specifying an `offset`.

Listing 1:
Retrieving a Time Series.

```
from pyrqa.file_reader import FileReader

time_series = FileReader.file_as_float_array('csv_data.txt',
                                             delimiter=',',
                                             column=3,
                                             offset=10)
```

7.2. Assigning values to the analysis parameters

RQA and recurrence plot rely on the specification of a set of input parameters. The values of those parameters are encapsulated within an object of the class `Settings`. The creation of such an object is depicted in List. 2. The set of parameters include:

- the input `time_series`,
- the `embedding_dimension` and the `time_delay` parameter used during the vector reconstruction,
- the `similarity_measure` used to compare the vectors, as well as
- the `neighbourhood` used to detect neighbouring vectors.

Version 0.1.0 of *PyRQA* supports the similarity measures L_1-norm, L_2-norm and L_∞-norm. A fixed radius can be applied as neighbourhood condition. Moreover, RQA relies on the specification of minimum line lengths, including:

- the `min_diagonal_line_length`,
- the `min_vertical_line_length`, and
- the `min_white_vertical_line_length`.

Listing 2:
Assigning Values to the Analysis Parameters.

```
from pyrqa.settings import Settings
from pyrqa.metric import EuclideanMetric
from pyrqa.neighbourhood import FixedRadius

settings = Settings(time_series,
                    embedding_dimension=2,
                    time_delay=3,
                    similarity_measure=EuclideanMetric,
                    neighbourhood=FixedRadius(radius=2.0),
                    min_diagonal_line_length=2,
                    min_vertical_line_length=2,
                    min_white_vertical_line_length=2)
```

7.3. Creating the analysis computation

PyRQA comprises two object classes that create computation objects based on the settings and other arguments specified; `RecurrencePlotComputation` and `RQAComputation`. This is realised using the static method `create` that is provided by both object classes.

The only required argument of the `create` method is a `Settings` object, capturing the analysis parameters. An optional argument is the OpenCL environment, which is encapsulated within an object of the class `OpenCL` (see List. 3). The OpenCL environment is determined automatically, if the `opencl` keyword argument is not assigned.

On its creation, an `OpenCL` object discovers the OpenCL platforms as well as compute devices available in the computing system. There are two ways to determine the OpenCL environment. First, platform and device selection can be performed using the method `create_environment`. This method employs the `platform_id` and the list of `device_ids` that are given as parameters to the constructor of the `OpenCL` object. Those IDs can be determined by a prior inspection of the platforms and devices available. Second, the discovery can be performed using the method `create_environment_command_line`. Here, the selection is conducted via command line input. The method called depends on the value of the boolean constructor argument `command_line`.

Listing 3:
Creating the Analysis Computation.

```
from pyrqa.opencl import OpenCL
from pyrqa.computation import RQAComputation

opencl = OpenCL(platform_id=0,
                device_ids=(0,),
                command_line=True,
                optimisations_enabled=False)

computation = RQAComputation.create(settings,
                                    opencl=opencl)
```

The `OpenCL` object is further responsible for compiling the kernel functions written in OpenCL C. In this regard, the compiler can be advised to activate certain optimisations, such as relaxed math operations. Several optimisations are activated by default. Those *default optimisations* are platform and device specific. To avoid inconsistency regarding the computing results across multiple plaforms, it might be required to deactivate those default optimisations. For this reason, the constructor of the `OpenCL` object provides the flag `optimisations_enabled`, which is set to `False` by default.

7.4. Retrieving the final computing results

Having created the computation object, the execution is started by calling its method `run`, which controls the processing of the analytical

operators. `run` returns the final computing results, which are encapsulated in an object of the class `RQAResult` or `RecurrencePlotResult`, depending on the analysis method chosen. Conducting RQA, the corresponding result contains a value for each quantitative measure. Those measures are members of the `RQAResult` object.

Listing 4:
Retrieving the Final Computing Results.

```
result = computation.run()
print result.determinism
```

8. Application examples

This section presents two examples regarding the application of *PyRQA*, focussing explicitly on the reproducibility of the computing results as well as the performance characteristics of the implementation. The first example refers to real-world climatological data that is freely available. It addresses the capabilities of the RQA method itself and the qualitative reproducibility of the quantitative measures. The second example employs a synthetic series that exceeds one million data points. It highlights the performance improvements gained by using *PyRQA* in combination with massively parallel computing hardware.

8.1. Climatological data

In this example, we investigate the recurrence properties of air temperature during a winter and a summer month. The recurrence properties should allow to compare the temperature dynamics between summer and winter season, e.g., whether the temperature would evolve in a more erratic or more regular way.

For this analysis, we have selected the measuring station at the *Asheville Regional Airport* in North Carolina. The temperature measurements are provided by the *National Oceanic and Atmospheric Administration* (NOAA) of the United States of America as part of the *Quality Controlled Local Climatological Data* (National Centers for Environmental Information, 2016). Measurement data can be obtained by querying the corresponding web form (selection: `Hourly (10A)`). The hourly dry-bulb temperature in degree Celsius in January and July 2016 are investigated (column: `DryBulbCelsius`). Only those measurements are considered, which refer to minute 54 of each hour. This results in two time series consisting of 744 data points.

Similar parameter assignments as in Rawald et al. (2014) are used for reconstructing the system states. The recurrence threshold has been selected in such a way that the recurrence rate in both examples is roughly 10% (Schinkel et al., 2008). The corresponding recurrence plots are shown in Fig. 2 and the RQA results are presented in Table 3. We have repeated the RQA by using the *Commandline Recurrence Plots* software (see Section 3), in order to check the reproducibility.

The recurrence plots of January and July 2016 exhibit different appearances. In July, the recurrence plot consists of many diagonal lines, clearly indicating daily variation. The January recurrence plot contrastingly consists of fewer diagonal lines and empty regions that express a less stationary dynamics during this season. Most of the RQA measures quantify these differences, only the measures RR, L_{max}, and DIV are equal.

Regarding January, we find higher values in DET and L than for July, but this difference comes from increased occurrences of extended black regions in the January recurrence plot. This is confirmed by the elevated values of LAM, TT, and V_{max} for January. Higher values in the entropy measures L_{entr} and V_{entr} reveal the more complex distribution of

diagonal and vertical lines, as it is also visible in the recurrence plot by the extended white regions and interrupted diagonal lines for January.

All these quantitative results allow us to conclude that during the summer months the temperature dynamics evolve more periodic and less irregular than during the winter months. Note, these tentative results and conclusions are derived from just one year and would need deeper investigation and statistical justification that are outside the scope of this paper. The focus of this application example is rather to demonstrate the features of *PyRQA* based on a real-world example using time series that are freely available.

Regarding the reproducibility test, we have recalculated the RQA measures from the two temperature time series by applying the *Commandline Recurrence Plots* software, using the exact same parameters as for the *PyRQA* computations. We found exactly the same results (see the parenthesised values in Table 3), confirming the validness of the *PyRQA* implementation and the reproducibility of its results.

8.2. Synthetic data

This example demonstrates the performance improvements of employing *PyRQA* in combination with parallel computing hardware, in particular multi-core CPUs and many-core GPUs. The runtime is compared to conducting the same analysis using the *Commandline Recurrence Plots* software. Note, the other tools presented in Section 3 are not capable of conducting RQA on such very long time series at all or allow only to analyse parts of it.

For the purpose of reproducibility, a synthetic series based on the sine function consisting of 1,000,001 data points is generated using the Python package NumPy (see List. 5). This series serves as an input to the RQA computation. An embedding dimension of 2, an time delay of 2 and a fixed radius of 1.0 are applied as RQA parameters. The minimum line lengths are set to 2.

Listing 5:
Creation of Synthetic Series.

```
import numpy as np

x = np.linspace(0,
                1000 * np.pi,
                1000001)
time_series = np.sin(x)
```

The computing system employed for conducting the runtime experiments contains an Intel Core i7-3820 CPU providing 4 cores and 8 threads. Each core runs at up to 3.8 GHz. The CPU accesses 16 GB of main memory. The computing system further includes two NVIDIA GeForce GTX 690 GPUs. Each GPU is equipped with two graphics processors. Each processor is provided with 2 GB of dedicated memory.

The total runtimes of executing the following configurations are compared in Table 4 and Fig. 3:

PyRQA (4x GPU):	*PyRQA* using the four GPU processors of the two NVIDIA GeForce GTX 690.
PyRQA (1x GPU):	*PyRQA* using a single GPU processor of one NVIDIA GeForce GTX 690.
PyRQA (1x CPU):	*PyRQA* using the Intel Core i7-3820 CPU.
CRP:	*Commandline Recurrence Plots* using the Intel Core i7-3820 CPU.

The OpenCL kernels are compiled with default optimisations on each platform. Each total runtime value is computed as the average of
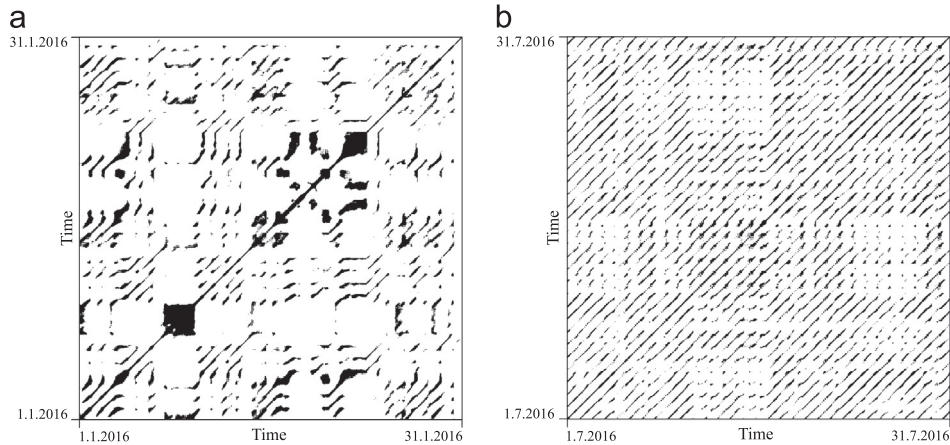
a   b

Fig. 2. Recurrence Plots of the Hourly Dry-bulb Temperature at Asheville Regional Airport, using *PyRQA*.

Table 3
RQA results of the Hourly dry-bulb temperature at Asheville Regional Airport for January and July 2016, using *PyRQA*. Results from the *Commandline Recurrence Plots* software are parenthesised and in agreement with the *PyRQA* results.

RQA measure	January 2016	July 2016
Recurrence rate (*RR*)	0.10 (0.10)	0.10 (0.10)
Determinism (*DET*)	0.94 (0.94)	0.88 (0.88)
Average diagonal line length (*L*)	7.80 (7.80)	5.75 (5.75)
Longest diagonal line length (L_{max})	732 (732)	732 (732)
Divergence (*DIV*)	0.001 (0.001)	0.001 (0.001)
Entropy diagonal lines (L_{entr})	2.67 (2.67)	2.14 (2.14)
Laminarity (*LAM*)	0.97 (0.97)	0.94 (0.94)
Trapping time (*TT*)	7.01 (7.01)	3.63 (3.63)
Longest vertical line length (V_{max})	62 (62)	13 (13)
Entropy vertical lines (V_{entr})	2.62 (2.62)	1.65 (1.65)

Table 4
Runtime results of the synthetic application example. The runtimes for four different configurations are presented. In addition, for each pair of configurations the corresponding speedup is listed.

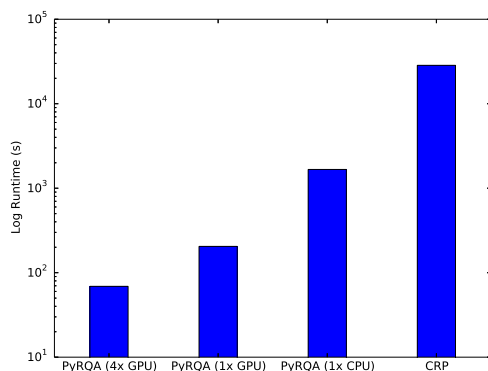| Configuration	Runtime (s)	Speedup in Comparison to		
		PyRQA (1× GPU)	*PyRQA* (1× CPU)	*CRP*
PyRQA (4× GPU)	68.94	2.97	24.19	413.13
PyRQA (1× GPU)	204.81	–	8.14	139.06
PyRQA (1× CPU)	1,667.58	–	–	17.08
CRP	28,480.09	–	–	–

Fig. 3. Runtime Results of the synthetic application example. The runtimes are depicted using a logarithmic scale.

four experimental runs.

Executing the synthetic example on four GPU processors using

PyRQA has the lowest total runtime, roughly 69 s. It delivers a speedup of a factor of almost three in comparison to using only a single GPU processor. The initial overhead of creating the OpenCL environment as well as the final computation of the RQA measures reduce the theoretical speedup of four. The speedup of using four GPUs in comparison to using the *Commandline Recurrence Plots* software is more than 400, signalling a drastic performance improvement.

A major benefit of using the OpenCL framework is to leverage the parallel computing capabilities of a variety of devices, including multi-core CPUs. Executing *PyRQA* on the Core i7 CPU delivers a speedup of 17 in comparison to executing *Commandline Recurrence Plots* on the same device. Among others, this is due to performing the processing in eight CPU threads, instead of one.

9. Conclusion and outlook

PyRQA is a free and open-source Python package that allows to efficiently conduct RQA for time series consisting of more than one million data points. Its major contributions are:

Distributed and parallel computing approach:	The underlying computing approach of *PyRQA* employs the concept Divide & Recombine. It subdivides the full recurrence matrix into sub matrices, performs the detection of diagonal and vertical line structures within a sub matrix in a massively parallel manner across multiple compute devices, and recombines the individual sub matrix results into a global result.
Usage of OpenCL:	*PyRQA* employs the OpenCL framework for heterogenous computing. This allows to conduct RQA in a parallel fashion on a variety of compute devices relying on different architectures, including GPUs and CPUs, from different hardware vendors, without having to modify the source code.

The combination of both contributions allows *PyRQA* to overcome the limitations of existing RQA implementations, which are either not capable of performing RQA on very long time series at all or consume large amounts of time. Using a synthetic series consisting of 1,000,001 data points, it is shown that the runtime for performing RQA could be reduced from almost eight hours using a state-of-the-art software tool to roughly 69 s. Those dramatic performance improvements enable novel applications, such as multi-scale recurrence analysis (Sips et al.,

2016).

The development of *PyRQA* is ongoing. The list of planned features include an improved separation of the analytical operators within the source code as well as an automatic selection of the best-performing RQA implementation at runtime. User feedback furthermore suggests to support the quantitative analysis of precomputed recurrence matrices.

Acknowledgments

Appendix A. Supplementary data

Supplementary data associated with this article can be found in the online version at http://dx.doi.org/10.1016/j.cageo.2016.11.016.

References

Chelidze, T., Matcharashvili, T., 2015. Dynamical patterns in seismology. In: Webber, C.L.Jr., Marwan, N. (Eds.), Recurrence Quantification Analysis – Theory and Best Practices. Springer, Cham, 291–334, http://dx.doi.org/10.1007/978-3-319-07155-8_10..

Coco, M.I., Dale, R., 2014. Cross-recurrence quantification analysis of categorical and continuous time series: an r package. Front. Psychol. 5, 510. http://dx.doi.org/10.3389/fpsyg.2014.00510.

Donges, J.F., Donner, R.V., Trauth, M.H., Marwan, N., Schellnhuber, H.J., Kurths, J., 2011. Nonlinear detection of paleoclimate-variability transitions possibly related to human evolution. Proc. Natl. Acad. Sci. 108, 20422–20427. http://dx.doi.org/10.1073/pnas.1117052108.

Donges, J.F., Heitzig, J., Beronov, B., Wiedermann, M., Runge, J., Feng, Q.Y., Tupikina, L., Stolbova, V., Donner, R.V., Marwan, N., Dijkstra, H.A., Kurths, J., 2015. Unified functional network and nonlinear time series analysis for complex systems science: the pyunicorn package. Chaos, 25, ⟨http://dx.doi.org/10.1063/1.4934554⟩.

Guha, S., Hafen, R., Rounds, J., Xia, J., Li, J., Xi, B., Cleveland, W.S., 2012. Large complex data: divide and recombine (D & R) with RHIPE. Stat 1, 53–67. http://dx.doi.org/10.1002/sta4.7.

Hegger, R., Kantz, H., Schreiber, T., 1999. Practical implementation of nonlinear time series methods: the TISEAN package. Chaos 9, 413–435, ⟨http://dx.doi.org/10.1063/1.166424⟩.

Keller, A., 2016. RQA X – Source Code. URL: ⟨http://www.recurrence-plot.tk/rqax.zip⟩.

Klöckner, A., 2016. PyOpenCL. ⟨http://mathema.tician.de/software/pyopencl/⟩.

Kurths, J., Schwarz, U., Sonett, C.P., Parlitz, U., 1994. Testing nonlinearity in radiocarbon data. Nonlinear Process. Geophys. 1, 72–75.

Li, S., Zhao, Z., Liu, F., 2008. Identifying spatial pattern of NDVI series dynamics using recurrence quantification analysis – a case study in the region around Beijing, China. Eur. Phys. J. – Spec. Top. 164, 127–139. http://dx.doi.org/10.1140/epjst/e2008-00839-y.

Lundh, F., Contributors, 2016. Pillow – The friendly PIL fork. ⟨https://python-pillow.org/⟩.

Marwan, N., 2011. How to avoid potential pitfalls in recurrence plot based data analysis. Int. J. Bifurc. Chaos 21, 1003–1017, ⟨http://dblp.uni-trier.de/db/journals/ijbc/ijbc21.html#Marwan11⟩.

Marwan, N. 2016a. Commandline Recurrence Plots. URL: ⟨http://tocsy.pik-potsdam.de/commandline-rp.php⟩.

Marwan, N., 2016b. CROSS RECURRENCE PLOT TOOLBOX 5.20 (R30.5). URL: ⟨http://tocsy.pik-potsdam.de/CRPtoolbox/⟩.

Marwan, N., Romano, M.C., Thiel, M., Kurths, J., 2007. Recurrence plots for the analysis of complex systems. Phys. Rep. 438, 237–329. http://dx.doi.org/10.1016/j.physrep.2006.11.001.

Marwan, N., Trauth, M.H., Vuille, M., Kurths, J., 2003. Comparing modern and Pleistocene ENSO-like influences in NW Argentina using nonlinear time series analysis methods. Clim. Dyn. 21, 317–326. http://dx.doi.org/10.1007/s00382-003-0335-3.

National Centers for Environmental Information, 2016. Quality Controlled Local Climatological Data. URL: ⟨http://www.ncdc.noaa.gov/qclcd/QCLCD?prior=N⟩.

Numpy Developers, 2016. NumPy. ⟨http://www.numpy.org/⟩.

Packard, N.H., Crutchfield, J.P., Farmer, J.D., Shaw, R.S., 1980. Geometry from a time series. Phys. Rev. Lett. 45, 712–716. http://dx.doi.org/10.1103/PhysRevLett.45.712.

Proulx, R., Côté, P., Parrott, L., 2009. Multivariate recurrence plots for visualizing and quantifying the dynamics of spatially extended ecosystems. Ecol. Complex. 6, 37–47. http://dx.doi.org/10.1016/j.ecocom.2008.10.003.

PyPA, 2016. pip. URL: ⟨https://pip.pypa.io/en/stable/⟩.

Rawald, T., Sips, M., 2016a. PyRQA. URL: ⟨https://pypi.python.org/pypi/PyRQA⟩.

Rawald, T., Sips, M., 2016b. PyRQA – Documentation. URL: ⟨https://www.pythonhosted.org/PyRQA/⟩.

Rawald, T., Sips, M., 2016c. PyRQA – GitLab Repository. URL: ⟨https://gitlab.com/tobiasr/PyRQA⟩.

Rawald, T., Sips, M., 2016d. PyRQA – Source Files. URL: ⟨https://pypi.python.org/packages/31/7d/dee51c852ad8a4d23c8a1184df779499863b80978349fef574c024a19f91/PyRQA-0.1.0.tar.gz⟩

Rawald, T., Sips, M., Marwan, N., Dransch, D., 2014. Fast Computation of Recurrences in Long Time Series. In: Translational Recurrences. From Mathematical Theory to Real-World Applications. Springer Proceedings in Mathematics & Statistics, vol. 103. Springer International Publishing, pp. 17–29.

Rawald, T., Sips, M., Marwan, N., Leser, U., 2015. Massively Parallel Analysis of Similarity Matrices on Heterogeneous Hardware. In: Proceedings of the Workshops of the EDBT/ICDT 2015 Joint Conference (EDBT/ICDT), Brussels, Belgium, March 27th, 2015, pp. 56–62. URL: ⟨http://ceur-ws.org/Vol-1330/paper-11.pdf⟩.

Schinkel, S., Dimigen, O., Marwan, N., 2008. Selection of recurrence threshold for signal detection. Eur. Phys. J. Spec. Top. 164, 45–53. http://dx.doi.org/10.1140/epjst/e2008-00833-5.

Sips, M., Witt, C., Rawald, T., Marwan, N., 2016. Torwards visual analytics for the exploration of large sets of time series. In: Webber, Jr., L.C., Ioana, C., Marwan, N., (Eds.), Recurrence Plots and Their Quantifications: Expanding Horizons: Proceedings of the 6th International Symposium on Recurrence Plots, Grenoble, France, 17-19 June 2015. Cham: Springer International Publishing, pp. 3–17. Url ⟨http://dx.doi.org/10.1007/978-3-319-29922-8_1⟩. http://dx.doi.org/10.1007/978-3-319-29922-8_1.

Stone, J.E., Gohara, D., Shi, G., 2010. OpenCL: a parallel programming standard for heterogeneous computing systems. Comput. Sci. Eng. 12, 66–73. http://dx.doi.org/10.1109/MCSE.2010.69, URL: ⟨http://doi.ieeecomputersociety.org/10.1109/MCSE.2010.69⟩.

Webber Jr., C.L., 2016. Charles L. Webber, Jr., Ph.D. - RQA Software. URL ⟨http://homepages.luc.edu/~cwebber/⟩.

Webber, C.L., Jr., Zbilut, J.P., 2005. Recurrence quantification analysis of nonlinear dynamical systems. In: Riley, M.A., Van Orden, G.C. (Eds.), Tutorials in Contemporary Nonlinear Methods for the Behavioral Sciences Web Book. National Science Foundation, U.S., 26–94.

Zbilut, J.P., Webber, C.L., Jr., 2007. Recurrence quantification analysis: introduction and historical context. Int. J. Bifurc. Chaos 17, 3477–3481. http://dx.doi.org/10.1142/S0218127407019238.

Zhao, Z.Q., Li, S.C., 2011. Identifying spatial patterns and dynamic of climate change using recurrence quantification analysis – case study of Qinghai-Tibet plateau. Int. J. Bifurc. Chaos 21, 1127–1139. http://dx.doi.org/10.1142/S0218127411028933.

Zolotova, N.V., Ponyavin, D.I., Marwan, N., Kurths, J., 2009. Long-term asymmetry in the wings of the butterfly diagram. Astron. Astrophys. 505, 197–201. http://dx.doi.org/10.1051/0004-6361/200811430.