# JIDT: An information-theoretic toolkit for studying the dynamics of complex systems

Joseph T. Lizier

The University of Sydney

Australian Conference on Aritifical Life and Computational Intelligence (ACALCI)
Newcastle, Australia
February 5, 2015

# Java Information Dynamics Toolkit (JIDT)

https://code.google.com/p/information-dynamics-toolkit/

JIDT provides a standalone, open-source (GPL v3 licensed) implementation of information-theoretic measures of information processing in complex systems, i.e. information storage, transfer and modification.

JIDT includes implementations:

- Principally for transfer entropy, mutual information, their conditional variants, active information storage etc;
- For both discrete and continuous-valued data;
- Using various types of estimators (e.g. Kraskov-Stögbauer-Grassberger, linear-Gaussian, etc.).

# Java Information Dynamics Toolkit (JIDT)

JIDT is written in Java but directly usable in Matlab/Octave, Python, R, Julia, Clojure, etc.

JIDT requires almost zero installation.

JIDT is distributed with:

- A paper describing its design and usage;
    - J.T. Lizier, *Frontiers in Robotics and AI* 1:11, 2014; (arXiv:1408.3270)
- Full Javadocs;
- A suite of demonstrations, including in each of the languages listed above.

# JIDT tutorial – Objectives

Participants will:

- Understand measures of information dynamics;
- Be able to obtain and install JIDT distribution;
- Understand and run sample scripts in their chosen environment;
- Be able to modify sample scripts for new analysis;
- Know how and where to seek support information (wiki, Javadocs, mailing list, twitter).

# Hi, I'm Joe ..

- My background.
- Why I developed an information-theoretic toolkit.

# Tell me about yourselves ...

- Where are you from? Unis, other organisations?

# Tell me about yourselves ...

- Where are you from? Unis, other organisations?
- Which languages/environments are you using, and how much coding experience do you have?

# Tell me about yourselves ...

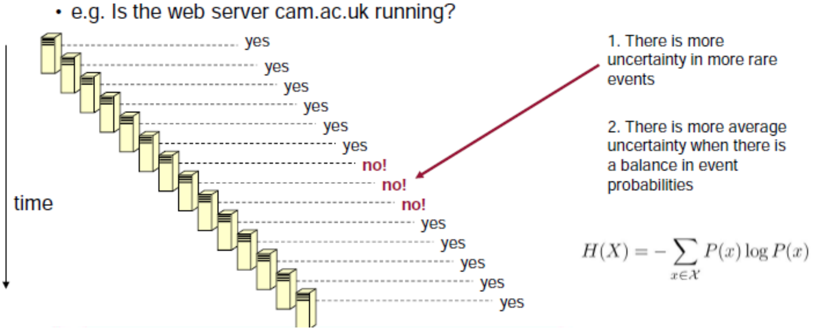
- Where are you from? Unis, other organisations?
- Which languages/environments are you using, and how much coding experience do you have?
- How familiar are you with information theory – what measures do you know? Are you using it for analysis yet?

# Tell me about yourselves ...

- Where are you from? Unis, other organisations?
- Which languages/environments are you using, and how much coding experience do you have?
- How familiar are you with information theory – what measures do you know? Are you using it for analysis yet?
- What types of information-theoretic analysis do you have planned (perhaps with JIDT)?

# Entropy

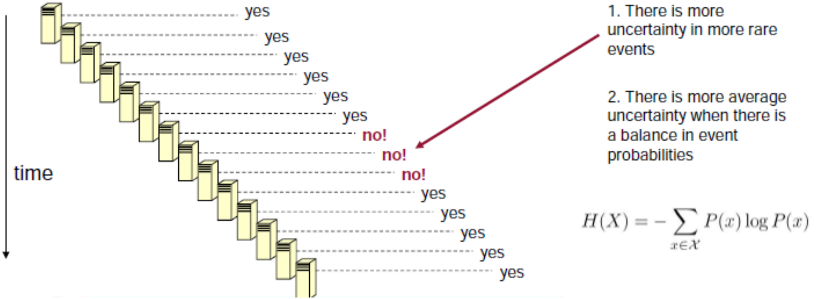
(Shannon) entropy is a measure of uncertainty.
Intuitively: if we want to know the value of a variable, there is uncertainty in what that value is before we inspect it (measured in bits).




$$H(X) = -\sum_{x \in X} P(x) \log P(x)$$

# Sample entropy calculation

- e.g. Is the web server cam.ac.uk running?



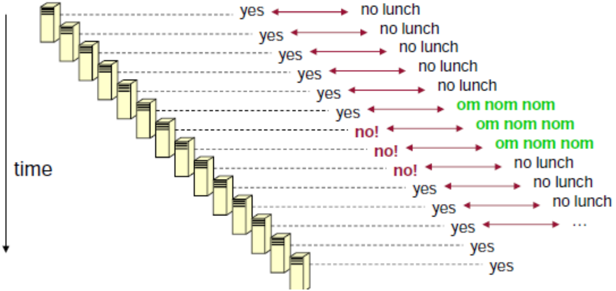

Here we have $p(\texttt{yes}) = 11/14$, $p(\texttt{no}) = 3/14$ – what is $H$?

| $x = \texttt{yes}\vert\texttt{no}$ | $p(x)$ | $-\log_2 p(x)$ | $-p(x)\log_2 p(x)$ |
|---|---|---|---|
| yes | 0.786 | 0.348 | 0.273 |
| no | 0.214 | 2.22 | 0.476 |
| | | | $H = 0.750$ bits |

# Conditional entropy

Uncertainty in one variable $X$ in the context of the known measurement of another variable $Y$.
Intuitively: how much uncertainty is there in $X$ after we know the value of $Y$?

e.g. How uncertain are we about the web server is running if we know the IT guy is at lunch?

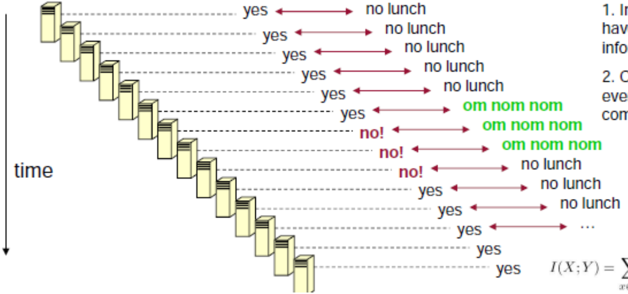



$$H(X|Y) = H(X,Y) - H(Y) = -\sum_{x \in X} \sum_{y \in Y} p(x,y) \log p(x|y)$$

# Mutual information

Information is a measure of uncertainty reduction.
Intuitively: common information is the amount that knowing the value of one variable tells us about another.

- e.g. How much common info b/w if IT guy is at lunch and the web server running?




1. Independent events have no common information
2. Completely dependent events have max common info

$$I(X;Y) = \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} P(x,y) \log \frac{P(x,y)}{P(x)P(y)}$$

# Information-theoretic quantities

Shannon entropy

$$H(X) = -\sum_{x} p(x) \log_2 p(x)$$
$$= \langle -\log_2 p(x) \rangle$$

Conditional entropy

$$H(X|Y) = -\sum_{x,y} p(x,y) \log_2 p(x|y)$$

Mutual information (MI)

$$I(X;Y) = H(X) + H(Y) - H(X,Y)$$
$$= \sum_{x,y} p(x,y) \log_2 \frac{p(x|y)}{p(x)}$$
$$= \left\langle \log_2 \frac{p(x|y)}{p(x)} \right\rangle$$

Conditional MI

$$I(X;Y|Z) = H(X|Z) + H(Y|Z) - H(X,Y|Z)$$
$$= \left\langle \log_2 \frac{p(x|y,z)}{p(x|z)} \right\rangle$$

# Local measures

We can write local (or point-wise) information-theoretic measures for specific observations/configurations $\{x, y, z\}$:

$$h(x) = -\log_2 p(x), \qquad i(x;y) = \log_2 \frac{p(x|y)}{p(x)}$$

$$h(x|y) = -\log_2 p(x|y), \qquad i(x;y|z) = \log_2 \frac{p(x|y,z)}{p(x|z)}$$

- We have $H(X) = \langle h(x) \rangle$ and $I(X;Y) = \langle i(x;y) \rangle$, etc.
- If $X, Y, Z$ are time-series, local values measure dynamics over time.

# What can we do with these measures in ALife/CI?

- Measure the diversity in agent strategies (Miramontes, 1995; Prokopenko et al., 2005).
- Measure long-range correlations as we approach a phase-transition (Ribeiro et al., 2008).
- Feature selection for machine learning (Wang et al., 2014).
- Quantify the information held in a response about a stimulus, and indeed about specific stimuli (DeWeese and Meister, 1999).
- Measure the common information in the behaviour of two agents (Sperati et al., 2008).
- Guide self-organisation using these measures (Prokopenko et al., 2006).
- ...

$\rightarrow$ Information theory is useful for answering specific questions about information content, shared information, and where and to what extent information about some variable is mirrored.

# Information dynamics

We *talk* about computation as:

- Memory
- Signalling
- Processing

Distributed computation is any process involving these features:

- Time evolution of cellular automata
- Information processing in the brain
- Gene regulatory networks computing cell behaviours
- Flocks computing their collective heading
- Ant colonies computing the most efficient routes to food
- The universe is computing its own future!

# Information dynamics

We *talk* about computation as:

- Memory
- Signalling
- Processing

Idea: quantify computation via:

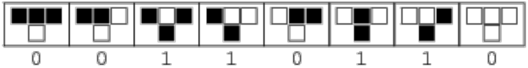
- Information storage
- Information transfer
- Information modification

Distributed computation is any process involving these features:

- Time evolution of cellular automata
- Information processing in the brain
- Gene regulatory networks computing cell behaviours
- Flocks computing their collective heading
- Ant colonies computing the most efficient routes to food
- The universe is computing its own future!

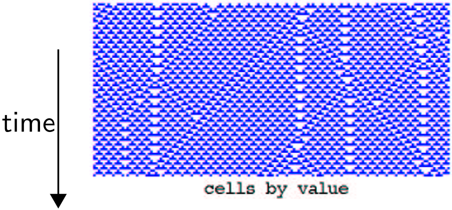
General idea: by quantifying intrinsic computation in the language it is normally described in, we can understand how nature computes and why it is complex.

# Motivating example: cellular automata




CAs: simple dynamical systems;

known causal structure and rules.

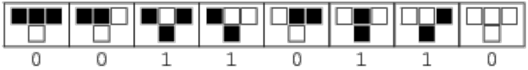



(Wuensche, 1999)

Emergent structure:

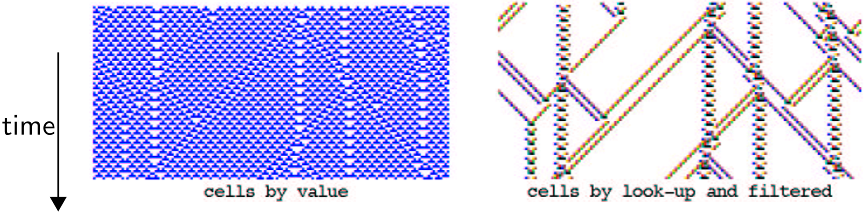
- Domain, blinkers
- Particles
  - Gliders, domain walls
- Collisions

# Motivating example: cellular automata




CAs: simple dynamical systems;

known causal structure and rules.




(Wuensche, 1999)

Emergent structure:

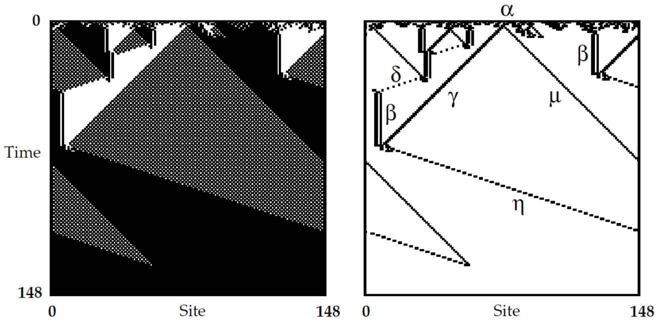
- Domain, blinkers
- Particles
  - Gliders, domain walls
- Collisions

**Conjectured** to represent:

- Information storage
- Information transfer
  - ”
- Information modification

It's easy to identify which components store, transfer and modify information in a PC – it's not so easy in complex systems.

# Motivating example: cellular automata




Mitchell et al. (1994, 1996) used GAs to evolve CAs to solve specific computational tasks.

In attempting the density classification task (above), the CA uses:

- domains and blinkers $\beta$ to store information;
- gliders $\gamma$,$\eta$ to transfer information;
- glider collisions e.g. $\gamma + \beta \rightarrow \eta$ to modify/process information.

# Information dynamics

We *talk* about computation as:

- Memory
- Signalling
- Processing

**Information dynamics**

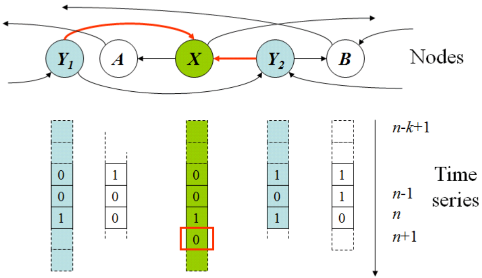
- Information storage
- Information transfer
- Information modification

Key properties of the information dynamics approach:

- A focus on individual operations of computation rather than overall complexity;
- Alignment with descriptions of dynamics in specific domains;
- A focus on the local scale of info dynamics in space-time;
- Information-theoretic basis directly measures computational quantities:
    - Captures non-linearities;
    - Is applicable to, and comparable between, any type of time-series.

# Information dynamics

Key question: how is the next state of a variable in a complex system **computed**?



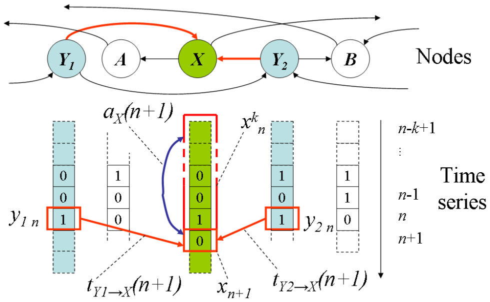

Complex system as a multivariate time-series of states

Q: Where does the information in $x_{n+1}$ come from, and how can we measure it?

Q: How much was stored, how much was transferred, can we partition them or do they overlap?

# Information dynamics

Studies computation of the next state of a target variable in terms of information storage, transfer and modification: (Lizier et al., 2008, 2010, 2012)

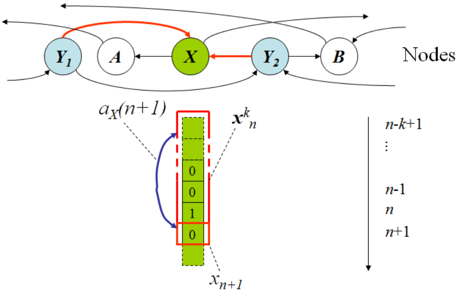



The measures examine:

- State updates of a target variable;
- Dynamics of the measures in space and time.

# Active information storage (Lizier et al., 2012)

How much information about the next observation $X_{n+1}$ of process $X$ can be found in its past state $\mathbf{X_n^{(k)}} = \{X_{n-k+1} \dots X_{n-1}, X_n\}$?



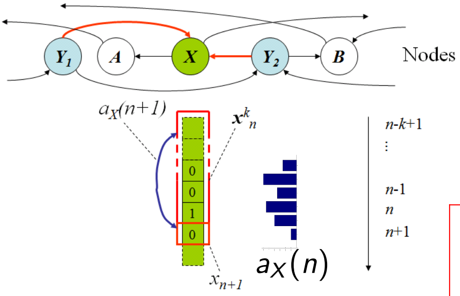

Active information storage:

$$A_X = I(X_{n+1}; \mathbf{X_n^{(k)}}) \\ = \left\langle \log_2 \frac{p(x_{n+1} | \mathbf{x_n^{(k)}})}{p(x_{n+1})} \right\rangle$$

Average information from past state that is in use in predicting the next value.

# Active information storage (Lizier et al., 2012)

How much information about the next observation $X_{n+1}$ of process $X$ can be found in its past state $\mathbf{X_n^{(k)}} = \{X_{n-k+1} \ldots X_{n-1}, X_n\}$?




$$A_X = \langle a_X(n) \rangle$$

Active information storage:

$$A_X = I(X_{n+1}; \mathbf{X_n^{(k)}}) = \left\langle \log_2 \frac{p(x_{n+1} | \mathbf{x_n^{(k)}})}{p(x_{n+1})} \right\rangle$$

Average information from past state that is in use in predicting the next value.

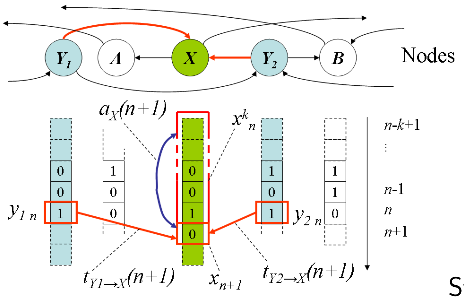
**Local** active information storage:

$$a_X(n) = \log_2 \frac{p(x_{n+1} | \mathbf{x_n^{(k)}})}{p(x_{n+1})}$$

Information from a specific past state that is in use in predicting the specific next value.

# Information transfer

How much information about the state transition $\mathbf{X_n^{(k)}} \rightarrow X_{n+1}$ of $X$ can be found in the past state $\mathbf{Y_n^{(l)}}$ of a source process $Y$?




Transfer entropy: (Schreiber, 2000)

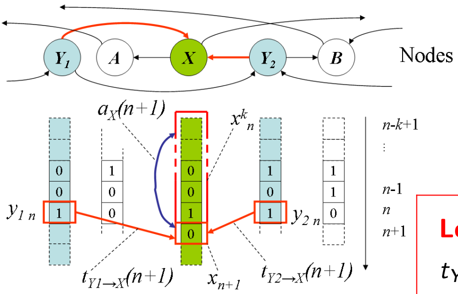
$$T_{Y\rightarrow X} = I(\mathbf{Y_n^{(l)}}; X_{n+1} \mid \mathbf{X_n^{(k)}})$$
$$= \left\langle \log_2 \frac{p(x_{n+1}|\mathbf{x_n^{(k)}}, \mathbf{y_n^{(l)}}))}{p(x_{n+1}|\mathbf{x_n^{(k)}}))} \right\rangle$$

Average info from source that helps predict next value in context of past.

Storage and transfer are complementary:
$H_X = A_X + T_{Y\rightarrow X} +$ higher order terms

# Information transfer

How much information about the state transition $\mathbf{X_n^{(k)}} \rightarrow X_{n+1}$ of $X$ can be found in the past state $\mathbf{Y_n^{(l)}}$ of a source process $Y$?




Transfer entropy: (Schreiber, 2000)

$$T_{Y\rightarrow X} = I(\mathbf{Y_n^{(l)}}; X_{n+1} \mid \mathbf{X_n^{(k)}})$$
$$= \left\langle \log_2 \frac{p(x_{n+1}|\mathbf{x_n^{(k)}},\mathbf{y_n^{(l)}}))}{p(x_{n+1}|\mathbf{x_n^{(k)}}))} \right\rangle$$

Average info from source that helps predict next value in context of past.

**Local** transfer entropy: (Lizier et al., 2008)

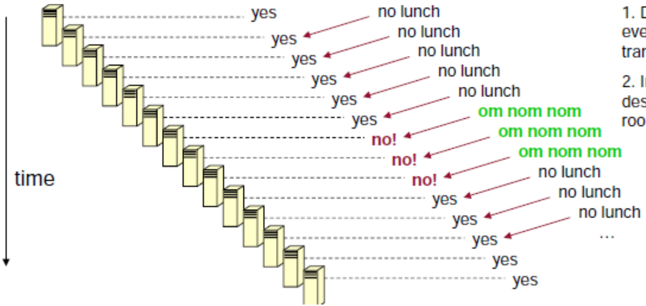
$$t_{Y\rightarrow X}(n) = \log_2 \frac{p(x_{n+1}|\mathbf{x_n^{(k)}},\mathbf{y_n^{(l)}}))}{p(x_{n+1}|\mathbf{x_n^{(k)}}))}$$

Information from a specific observation about the specific next value.

# Information transfer

Transfer entropy measures directed coupling between time-series. Intuitively: the amount of information that a source variable tells us about a destination, in the context of the destination's current state.

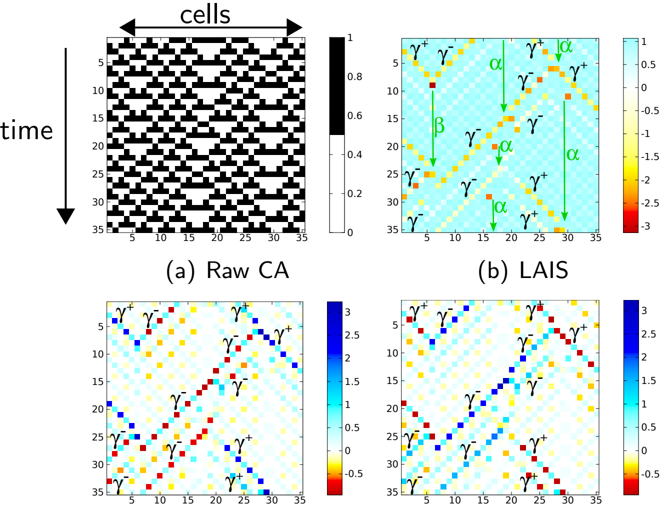
e.g. How much does knowing the IT guy is at lunch tell us about the web server running, given its previous state?




1. Directionaly dependent events have max info transfer

2. Internal predictabiity in destination allows no room for transfer

# Information dynamics in CAs




(a) Raw CA (b) LAIS

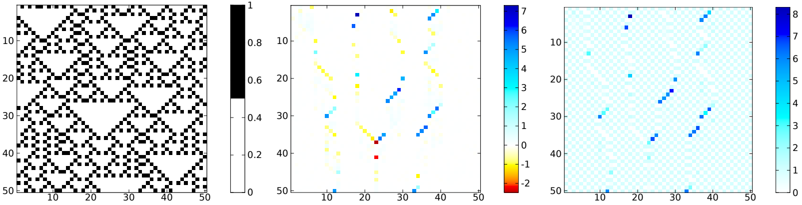
(c) LTE right (d) LTE left

Domains and blinkers are the dominant information storage entities.

Gliders are the dominant information transfer entities.

# Other transfer entropy characteristics

TE can be made conditional $T_{Y \to X|Z} = I(\mathbf{Y_n^{(l)}}; X_{n+1} \mid \mathbf{X_n^{(k)}}, \mathbf{Z_n^{(m)}})$ or multivariate $T_{Y \to X|Z} = I(\{\mathbf{Y_n^{(l)}}, \mathbf{Z_n^{(m)}}\}; X_{n+1} \mid \mathbf{X_n^{(k)}})$ (Lizier et al., 2008, 2010, 2011)




(a) Raw CA (b) LTE left (c) LTE left conditional

Lizier et al. (2008, 2014)

Computed over delay $u$ as $T_{Y \to X|Z} = I(\mathbf{Y_{n-u+1}^{(l)}}; X_{n+1} \mid \mathbf{X_n^{(k)}})$ (Wibral et al., 2013).

# Discrete: plug-in estimator

For discrete variables $x$ and $y$, to compute $H(X,Y)$

1. estimate: $p(x,y) = \frac{\texttt{count}(X=x,Y=y)}{N}$, where $N$ is our sample size;
2. plug-in each estimated PDF to $H(X,Y)$ to get $\hat{H}(X,Y)$

# Discrete: plug-in estimator

For discrete variables $x$ and $y$, to compute $H(X, Y)$

1. estimate: $p(x, y) = \frac{\texttt{count}(X=x, Y=y)}{N}$, where $N$ is our sample size;
2. plug-in each estimated PDF to $H(X, Y)$ to get $\hat{H}(X, Y)$

Bias: expected offset of estimated value from a finite sample set from the true underlying value of the measure. There are several available bias correction techniques.
Variance: variance in estimated values of the measure (from a finite sample set) around the expected value.

# Discrete: plug-in estimator

For discrete variables $x$ and $y$, to compute $H(X, Y)$

1. estimate: $p(x, y) = \frac{\texttt{count}(X{=}x, Y{=}y)}{N}$, where $N$ is our sample size;
2. plug-in each estimated PDF to $H(X, Y)$ to get $\hat{H}(X, Y)$

Bias: expected offset of estimated value from a finite sample set from the true underlying value of the measure. There are several available bias correction techniques.
Variance: variance in estimated values of the measure (from a finite sample set) around the expected value.

A simple way to handle continuous variables is to discretise or bin them.

# Continuous variables → Differential entropy

Differential entropy:

$$H_D(X) = -\int_{S_X} f(x) \log f(x) dx$$

for PDF $f(x)$, where $S_X$ is the set where $f(x) > 0$.

Evaluate all measures as sums and differences of $H_D(X)$ terms.

The properties of $H_D(X)$ are slightly odd ... however, the properties of $I_D(X;Y)$ are the same as for discrete variables.

JIDT includes 3 estimation methods for differential entropy based MIs (and conditional MIs) ...

# Gaussian model

If a multivariate $\mathbf{X}$ (of $d$ dimensions) is Gaussian distributed (Cover and Thomas, 1991):

$$H(\mathbf{X}) = \frac{1}{2} \ln \left[ (2\pi e)^d \mid \Omega_{\mathbf{X}} \mid \right]$$

(in *nats*) where $\mid \Omega_{\mathbf{X}} \mid$ is the determinant of the $d \times d$ covariance matrix $\Omega_{\mathbf{X}} = \overline{\mathbf{X}\mathbf{X}^T}$.
Any measure is computed as sums and differences of these joint entropies.

Pros: fast ($O\left(Nd^2\right)$), parameter free
Cons: subject to the linear-model assumption

# Kernel estimation

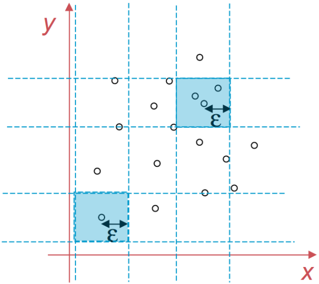
Estimate PDFs with a *kernel function* $\Theta$, measuring "similarity" between pairs of samples $\{x_n, y_n\}$ and $\{x_{n'}, y_{n'}\}$ using a resolution or *kernel width* $r$.

E.g.: $$\hat{p}_r(x_n, y_n) = \frac{1}{N} \sum_{n'=1}^{N} \Theta\left( \left| \begin{pmatrix} x_n - x_{n'} \\ y_n - y_{n'} \end{pmatrix} \right| - r \right).$$

By default $\Theta$ is the step kernel ($\Theta(x > 0) = 0$, $\Theta(x \leq 0) = 1$), and the norm $|\cdot|$ is the maximum distance.

Pros: model-free (captures non-linearities)
Cons: sensitive to $r$, is biased, less time-efficient (though can be reduced to $O(N \log N)$).

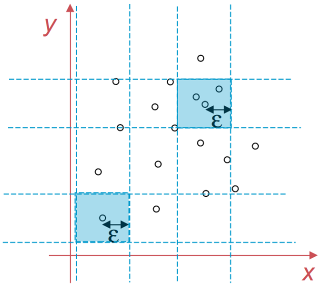
# Estimating $p(x,y)$, $p(x)$ and $p(y)$

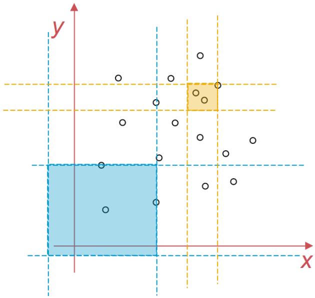



## Kernel estimation

- Fixed width $r = \epsilon$
- MI: “How does knowing $x$ within $r$ help me predict $y$ within $r$?”

# Estimating $p(x, y)$, $p(x)$ and $p(y)$




**Kernel estimation**

- Fixed width $r = \epsilon$
- MI: "How does knowing $x$ within $r$ help me predict $y$ within $r$?"

**Kraskov (KSG) technique** (Kraskov et al., 2004)

- Dynamic width $r$ and bias correction
- MI: "How does knowing $x$ within the $K$ neasrest neighbours in the joint space help me predict $y$?"

# KSG estimators (Kraskov et al., 2004)

Improve on box-kernel estimation with lower bias via:

- Harnessing Kozachenko-Leonenko entropy estimators;
- Using nearest-neighbour counting, with a fixed number $K$ of neighbours in the full joint space.

# KSG estimators (Kraskov et al., 2004)

Improve on box-kernel estimation with lower bias via:

- Harnessing Kozachenko-Leonenko entropy estimators;
- Using nearest-neighbour counting, with a fixed number $K$ of neighbours in the full joint space.

There are two algorithms; algorithm 1 gives:

$$I^{(1)}(X;Y) = \psi(K) - \langle \psi(n_x + 1) + \psi(n_y + 1) \rangle + \psi(N),$$

(in *nats*) where $\psi$ denotes the digamma function.
Extensions to conditional MI are available (Frenzel and Pompe, 2007; Gomez-Herrero et al., 2010; Wibral et al., 2014).

Pros: model-free, bias corrected, best of breed in terms of data efficiency and accuracy, and is effectively parameter free (w.r.t $K$).

Cons: less time-efficient (though fast nearest neighbour searching reduces this to $O(KN \log N)$).

# Why JIDT?

JIDT is unique in the combination of features it provides:

- Large array of measures, including all conditional/multivariate forms of the transfer entropy, and complementary measures such as active information storage.
- Wide variety of estimator types and applicability to both discrete and continuous data

# Measure-estimator combinations

As of V1.2 distribution:

| Measure | | Discrete | Continuous estimators | | | |
|---|---|---|---|---|---|---|
| Name | Notation | estimator | Gaussian | Box-Kernel | Kraskov *et al.*(KSG) | Permutation |
| Entropy | $H(X)$ | ✓ | ✓ | ✓ | $\star$ | |
| Entropy rate | $H_{\mu X}$ | ✓ | *Use two multivariate entropy calculators* | | | |
| Mutual information (MI) | $I(X;Y)$ | ✓ | ✓ | ✓ | ✓ | |
| Conditional MI | $I(X;Y \mid Z)$ | ✓ | ✓ | | ✓ | |
| Multi-information | $I(\mathbf{X})$ | ✓ | | $\checkmark^u$ | $\checkmark^u$ | |
| Transfer entropy (TE) | $T_{Y\rightarrow X}$ | ✓ | ✓ | ✓ | ✓ | $\checkmark^u$ |
| Conditional TE | $T_{Y\rightarrow X\mid Z}$ | ✓ | $\checkmark^u$ | | $\checkmark^u$ | |
| Active information storage | $A_X$ | ✓ | $\checkmark^u$ | $\checkmark^u$ | $\checkmark^u$ | |
| Predictive information | $E_X$ | ✓ | $\checkmark^u$ | $\checkmark^u$ | $\checkmark^u$ | |
| Separable information | $S_X$ | ✓ | | | | |

# Why JIDT?

JIDT is unique in the combination of features it provides:

- Large array of measures, including all conditional/multivariate forms of the transfer entropy, and complementary measures such as active information storage.
- Wide variety of estimator types and applicability to both discrete and continuous data
- Local measurement for all estimators;
- Statistical significance calculations for MI, TE;
- No dependencies on other installations (except Java);
- Lots of demos and information on website/wiki:
  - https://code.google.com/p/information-dynamics-toolkit/

# Why implement in Java?

The Java implementation of JIDT gives us several fundamental features:

- Platform agnostic, requiring only a JVM;
- Object-oriented code, with a hierachical design to interfaces for each measure, allowing dynamic swapping of estimators for the same measure;
- JIDT can be directly called from Matlab/Octave, Python, R, Julia, Clojure, etc, adding efficiency for higher level code;
- Automatic generation of Javadocs.

# Installation

1. Download the latest full distribution from https://code.google.com/p/information-dynamics-toolkit/wiki/Downloads or http://bit.ly/jidt-download
2. Unzip it to your prefered location for the distribution
3. To be able to use it, you will need the `infodynamics.jar` on your classpath.

# Installation

1. Download the latest full distribution from https://code.google.com/p/information-dynamics-toolkit/wiki/Downloads or http://bit.ly/jidt-download
2. Unzip it to your prefered location for the distribution
3. To be able to use it, you will need the `infodynamics.jar` on your classpath.

That's it!

# Installation – caveats

1. You'll need a JRE installed (should come automatically with Matlab/Octave/Python)
2. You need `ant` if you want to rebuild the project using `build.xml`
3. You need `junit` if you want to run the unit tests
4. Additional preparation may be required to use JIDT in GNU Octave or Python ...

# Check that your environment works

Java:

1. Run `demos/java/example1TeBinaryData.sh` or `.bat`

Matlab/Octave:

1. For Octave version $< 3.8$, first follow steps on the wiki, including installing `octave-java` from `octave-forge`.
2. Run `demos/octave/example1TeBinaryData.m`

Python:

1. Install `jPype` to connect Python to Java
2. Run `demos/python/example1TeBinaryData.py`

In case of issues, see the wiki pages on Non-Java environments or the Instructor.

# Contents of distribution

- `license-gplv3.txt` - GNU GPL v3 license;
- `infodynamics.jar` library file;
- Documentation
- Source code in `java/source` folder
- Unit tests in `java/unittests` folder
- `build.xml` ant build script
- Demonstrations of the code in `demos` folder.

# Documentation

Included in the distribution:

- `readme.txt`;
- `InfoDynamicsToolkit.pdf` – a pre-print of the publication introducing JIDT;
- `javadocs` folder – documents the methods and various options for each estimator class;
- PDFs describing each demo in the `demos` folder;

Also see:

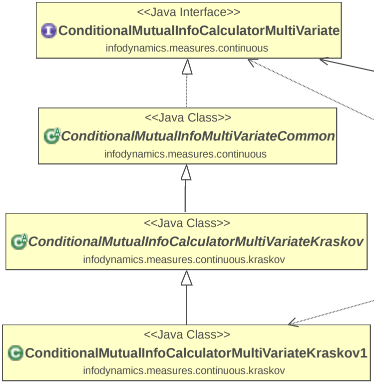
- The wiki pages on the JIDT website
- This presentation! (via JIDT wiki)
- Our email discussion list `jidt-discuss` on Google groups.

# Source code structure

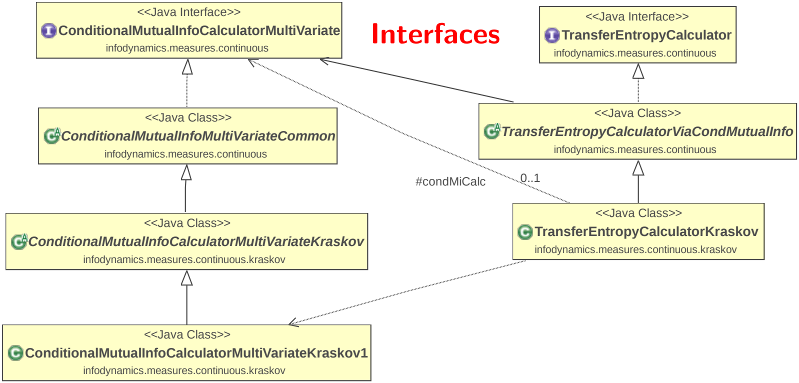
Source code at `java/source` is organised into the following Java *packages* (mapping directly to subdirectories):

- `infodynamics.measures`
  - `infodynamics.measures.discrete` – for discrete data;
  - `infodynamics.measures.continuous` – for continuous data
    - top level: Java *interfaces* for each measure, then
    - a set of sub-packages (`gaussian`, `kernel`, `kozachenko`, `kraskov` and `symbolic`) containing *implementations* of such estimators for these interfaces.
  - `infodynamics.measures.mixed` – experimental discrete-to-continuous MI calculators
- `infodynamics.utils` – utility functions
- `infodynamics.networkinference` – higher-level algorithms

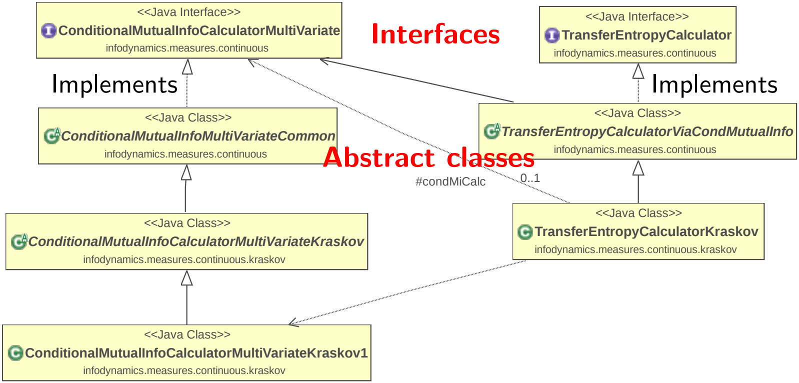
# Architecture for calculators on continuous data



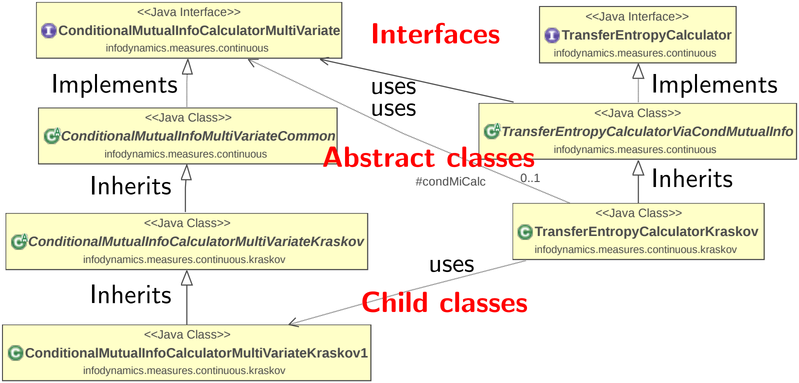
# Architecture for calculators on continuous data



# Architecture for calculators on continuous data



# Architecture for calculators on continuous data




Return to JIDT contents

# Demos

JIDT is ditributed with the following demos:

- Simple Java Demos
    - Mirrored in Matlab/Octave, Python, R, Julia, Clojure.
- Recreation of Schreiber's original transfer entropy examples;
- Information dynamics in Cellular Automata;
- Detecting interaction lags;
- Interregional coupling;
- Behaviour of null/surrogate distributions;

All have documentation provided to help run them.

# Simple Java Demos

There are 8 demo scripts here highlighting different data types and estimation techniques.
We'll walk through:

1. Example 1 - a typical calling pattern on discrete data; and
2. Example 4 - a typical calling pattern on continuous data.

These examples use transfer entropy calculators, but note that the general paradigm for all calculators is the same.

# Simple Demo 1 – Discrete Data

Open:

`demos/java/infodynamics/demos/Example1TeBinaryData.java`

OR

`demos/octave/Example1TeBinaryData.m`

OR

`demos/python/Example1TeBinaryData.py`

that you ran earlier

# Simple Demo 1 – Discrete Data

Open:

`demos/java/infodynamics/demos/Example1TeBinaryData.java`

OR

`demos/octave/Example1TeBinaryData.m`

OR

`demos/python/Example1TeBinaryData.py`

that you ran earlier

1. Run it again

# Simple Demo 1 – Discrete Data

Open:
`demos/java/infodynamics/demos/Example1TeBinaryData.java`
OR
`demos/octave/Example1TeBinaryData.m`
OR
`demos/python/Example1TeBinaryData.py`
that you ran earlier

1. Run it again
2. Observe how the classpath is pointed to `infodynamics.jar`:
   - Java: `java` command line in `.sh`/`.bat` (or in IDE);
   - Matlab/Octave: `javaaddpath()` statement;
   - Python: `startJVM()` statement.

# Simple Java Demo 1 – Discrete Data

3. Examine the code (excerpt from .java file below)

```java
int arrayLengths = 100;
RandomGenerator rg = new RandomGenerator();
// Generate some random binary data:
int[] sourceArray = rg.generateRandomInts(
    arrayLengths, 2);
int[] destArray = new int[arrayLengths];
destArray[0] = 0;
System.arraycopy(sourceArray, 0, destArray, 1,
    arrayLengths - 1);
// Create a TE calculator and run it:
TransferEntropyCalculatorDiscrete teCalc = new
    TransferEntropyCalculatorDiscrete(2, 1);
teCalc.initialise();
teCalc.addObservations(sourceArray, destArray);
double result = teCalc.
    computeAverageLocalOfObservations();
```

# Simple Java Demo 1 – Discrete Data

```java
int arrayLengths = 100;
RandomGenerator rg = new RandomGenerator();
// Generate some random binary data:
int[] sourceArray = rg.generateRandomInts(arrayLengths, 2);
int[] destArray = new int[arrayLengths];
destArray[0] = 0;
System.arraycopy(sourceArray, 0, destArray, 1, arrayLengths - 1);
// Create a TE calculator and run it:
TransferEntropyCalculatorDiscrete teCalc = new TransferEntropyCalculatorDiscrete(2, 1);
teCalc.initialise();
teCalc.addObservations(sourceArray, destArray);
double result = teCalc.computeAverageLocalOfObservations();
```

4. Note: Discrete data represented as `int[]` arrays:
   1. with values in the range $0 \ldots base - 1$, where e.g. base=2 for binary.
   2. for time-series measures, the array is indexed by time.
   3. for multivariate time-series, we use `int[][]` arrays, indexed first by time then variable number.

# Simple Java Demo 1 – Discrete Data – Usage Paradigm

```
int arrayLengths = 100;
RandomGenerator rg = new RandomGenerator();
// Generate some random binary data:
int[] sourceArray = rg.generateRandomInts(arrayLengths, 2);
int[] destArray = new int[arrayLengths];
destArray[0] = 0;
System.arraycopy(sourceArray, 0, destArray, 1, arrayLengths - 1);
// Create a TE calculator and run it:
TransferEntropyCalculatorDiscrete teCalc = new TransferEntropyCalculatorDiscrete
     (2, 1);
teCalc.initialise();
teCalc.addObservations(sourceArray, destArray);
double result = teCalc.computeAverageLocalOfObservations();
```

1. Construct the calculator, providing parameters
   1. Always check Javadocs for which parameters are required.
   2. Here the parameters are the number of possible discrete symbols per sample (2, binary), and history length for TE ($k = 1$).
   3. Constructor syntax is different for Matlab/Octave/Python.

# Simple Java Demo 1 – Discrete Data – Usage Paradigm

```
1  int arrayLengths = 100;
2  RandomGenerator rg = new RandomGenerator();
3  // Generate some random binary data:
4  int[] sourceArray = rg.generateRandomInts(arrayLengths, 2);
5  int[] destArray = new int[arrayLengths];
6  destArray[0] = 0;
7  System.arraycopy(sourceArray, 0, destArray, 1, arrayLengths - 1);
8  // Create a TE calculator and run it:
9  TransferEntropyCalculatorDiscrete teCalc = new TransferEntropyCalculatorDiscrete
       (2, 1);
10 teCalc.initialise();
11 teCalc.addObservations(sourceArray, destArray);
12 double result = teCalc.computeAverageLocalOfObservations();
```

2. Initialise the calculator prior to:
   1. use, or
   2. re-use (e.g. looping back from line 12 back to line 10 to examine different data).
   3. This clears PDFs ready for new samples.

# Simple Java Demo 1 – Discrete Data – Usage Paradigm

```java
int arrayLengths = 100;
RandomGenerator rg = new RandomGenerator();
// Generate some random binary data:
int[] sourceArray = rg.generateRandomInts(arrayLengths, 2);
int[] destArray = new int[arrayLengths];
destArray[0] = 0;
System.arraycopy(sourceArray, 0, destArray, 1, arrayLengths - 1);
// Create a TE calculator and run it:
TransferEntropyCalculatorDiscrete teCalc = new TransferEntropyCalculatorDiscrete
      (2, 1);
teCalc.initialise();
teCalc.addObservations(sourceArray, destArray);
double result = teCalc.computeAverageLocalOfObservations();
```

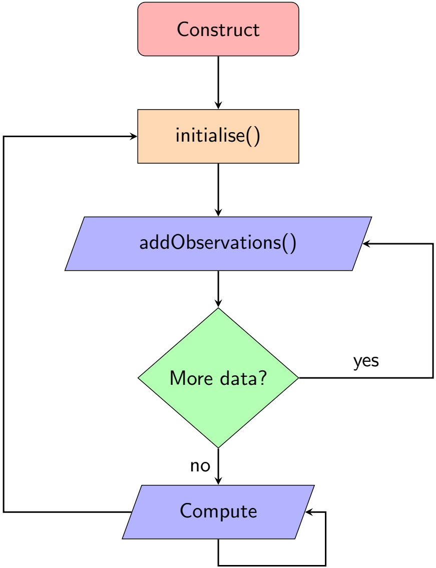
3. Supply the data to the calculator to construct PDFs:
   1. `addObservations()` may be called multiple times;
   2. Convert arrays into Java format:
      - From Matlab/Octave using our `octaveToJavaIntArray(array)`, etc., scripts.
      - From Python using `JArray(JInt, numDims)(array)`, etc.

# Simple Java Demo 1 – Discrete Data – Usage Paradigm

```java
int arrayLengths = 100;
RandomGenerator rg = new RandomGenerator();
// Generate some random binary data:
int[] sourceArray = rg.generateRandomInts(arrayLengths, 2);
int[] destArray = new int[arrayLengths];
destArray[0] = 0;
System.arraycopy(sourceArray, 0, destArray, 1, arrayLengths - 1);
// Create a TE calculator and run it:
TransferEntropyCalculatorDiscrete teCalc = new TransferEntropyCalculatorDiscrete
      (2, 1);
teCalc.initialise();
teCalc.addObservations(sourceArray, destArray);
double result = teCalc.computeAverageLocalOfObservations();
```

4. Compute the measure:
    1. Value is always returned in bits for discrete calculators.
    2. Result here approaches 1 bit since destination copies the (random) source.
    3. Other computations include:
        1. `computeLocalOfPreviousObservations()` for local values
        2. `computeSignificance()` to compute $p$-values of measures of predictability (see Appendix A5 of paper for description).

# Discrete Data – Usage Paradigm



# Simple Demo 4 – Continuous Data

Open:

`demos/java/infodynamics/demos/Example4TeContinuousDataKrask`

OR

`demos/octave/Example4TeContinuousDataKraskov.m`

OR

`demos/python/Example4TeContinuousDataKraskov.py`

# Simple Demo 4 – Continuous Data

Open:

`demos/java/infodynamics/demos/Example4TeContinuousDataKrask`

OR

`demos/octave/Example4TeContinuousDataKraskov.m`

OR

`demos/python/Example4TeContinuousDataKraskov.py`

1. Run it as you did for example 1.

# Simple Java Demo 4 – Continuous Data

3. Examine the code (excerpt from .java file below) – can you notice anything different to the discrete case?

```
double[] sourceArray, destArray;
// ...
// Import values into sourceArray and destArray
// ...
TransferEntropyCalculatorKraskov teCalc = new
    TransferEntropyCalculatorKraskov();
teCalc.setProperty("k", "4");
teCalc.initialise(1);
teCalc.setObservations(sourceArray, destArray);
double result = teCalc.
    computeAverageLocalOfObservations();
```

# Simple Java Demo 4 – Continuous Data

```java
double[] sourceArray, destArray;
// ...
// Import values into sourceArray and destArray
// ...
TransferEntropyCalculatorKraskov teCalc = new TransferEntropyCalculatorKraskov()
    ;
teCalc.setProperty("k", "4");
teCalc.initialise(1);
teCalc.setObservations(sourceArray, destArray);
double result = teCalc.computeAverageLocalOfObservations();
```

4. Note: Continuous data represented as `double[]` arrays:
   1. for time-series measures, the array is indexed by time.
   2. for multivariate time-series, we use `double[][]` arrays, indexed first by time then variable number.

# Simple Java Demo 4 – Continuous Data – Usage Paradigm

```java
double[] sourceArray, destArray;
// ...
// Import values into sourceArray and destArray
// ...
TransferEntropyCalculatorKraskov teCalc = new TransferEntropyCalculatorKraskov()
    ;
teCalc.setProperty("k", "4");
teCalc.initialise(1);
teCalc.setObservations(sourceArray, destArray);
double result = teCalc.computeAverageLocalOfObservations();
```

1. Construct the calculator, *possibly* providing parameters
    1. Always check Javadocs for which parameters are required.
    2. For continuous calculators, parameters may always be provided later (see next slide) to allow dynamic instantiation.
    3. Constructor syntax is different for Matlab/Octave/Python.

# Simple Java Demo 4 – Continuous Data – Usage Paradigm

```java
double[] sourceArray, destArray;
// ...
// Import values into sourceArray and destArray
// ...
TransferEntropyCalculatorKraskov teCalc = new TransferEntropyCalculatorKraskov();
teCalc.setProperty("k", "4");
teCalc.initialise(1);
teCalc.setObservations(sourceArray, destArray);
double result = teCalc.computeAverageLocalOfObservations();
```

2. Set properties for the calculator (new method for continuous):
    1. Check the Javadocs for available properties for each calculator;
    2. E.g. here we set the number $k$ of nearest neighbours for the KSG calculation.
    3. Property names and values are always key-value pairs of `String` objects;
    4. Only guaranteed to hold after the next `intialise()` call.
    5. Properties can easily be extracted and set from a file (see Simple Demo 6).

# Simple Java Demo 4 – Continuous Data – Usage Paradigm

```java
double[] sourceArray, destArray;
// ...
// Import values into sourceArray and destArray
// ...
TransferEntropyCalculatorKraskov teCalc = new TransferEntropyCalculatorKraskov();
teCalc.setProperty("k", "4");
teCalc.initialise(1);
teCalc.setObservations(sourceArray, destArray);
double result = teCalc.computeAverageLocalOfObservations();
```

3. Initialise the calculator prior to:
   1. use or re-use, as for Discrete.
   2. This clears PDFs ready for new samples, and finalises any new property settings.
   3. There may be several overloaded forms taking different arguments. In the above, `teCalc.initialise(1)` sets history length $k = 1$. We could also call `teCalc.initialise(k, tau_k, l, l_tau, delay)` here to specify embedding dimensions and source-target delay. Check Javadocs for options here.

# Simple Java Demo 4 – Continuous Data – Usage Paradigm

```java
double[] sourceArray, destArray;
// ...
// Import values into sourceArray and destArray
// ...
TransferEntropyCalculatorKraskov teCalc = new TransferEntropyCalculatorKraskov();
teCalc.setProperty("k", "4");
teCalc.initialise(1);
teCalc.setObservations(sourceArray, destArray);
double result = teCalc.computeAverageLocalOfObservations();
```

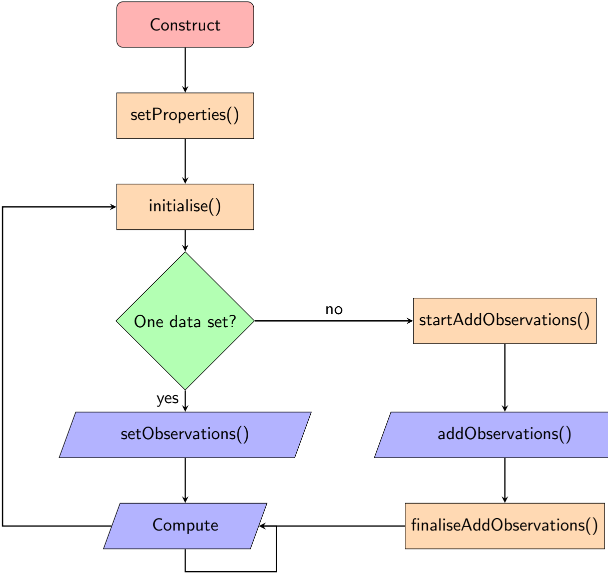
4. Supply the data to the calculator to construct PDFs:
    1. `setObservations()` may be called once, OR
    2. call `addObservations()` multiple times, in between `startAddObservations()` and `finaliseAddObservations()` calls.
    3. Convert arrays into Java format:
        - From Matlab/Octave using our `octaveToJavaDoubleArray(array)`, etc., scripts.
        - From Python using `JArray(JDouble, numDims)(array)`, etc.

# Simple Java Demo 4 – Continuous Data – Usage Paradigm

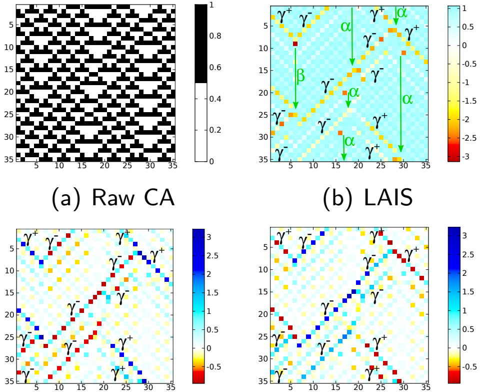
```java
double[] sourceArray, destArray;
// ...
// Import values into sourceArray and destArray
// ...
TransferEntropyCalculatorKraskov teCalc = new TransferEntropyCalculatorKraskov();
teCalc.setProperty("k", "4");
teCalc.initialise(1);
teCalc.setObservations(sourceArray, destArray);
double result = teCalc.computeAverageLocalOfObservations();
```

5. Compute the measure:
   1. Value may be in *bits* (kernel) OR in *nats* (LSG, Gaussian) calculators.
   2. Result here approaches 0.17 nats since destination is correlated with the (random) source.
   3. Other computations include:
      1. `computeLocalOfPreviousObservations()` for local values
      2. `computeSignificance()` to compute *p*-values of measures of predictability (see Appendix A5 of paper for description).

# Continuous Data – Usage Paradigm



# Many other demos – e.g. local dynamics in CAs



(a) Raw CA (b) LAIS

(c) LTE right (d) LTE left

See PDF documentation for `demos/octave/CellularAutomata/` to recreate, e.g. run `GsoChapterDemo2013.m`.

# Exercise

1. Read and understand the calculation of transfer entropy between heart rate and breath rate measurements (data from Rigney et al. (1993)) in:
   1. `demos/java/infodynamics/java/-schreiberTransferEntropyExamples/-HeartBreathRateKraskovRunner.java`
   2. `demos/octave/SchreiberTransferEntropyExamples/-runHeartBreathRateKraskov.m`
   3. `demos/python/SchreiberTransferEntropyExamples/-runHeartBreathRateKraskov.py` (available here in SVN, not in V1.2 distribution)

# Exercise

2. Task:
    1. Compute Mutual Information between the heart and breath rate time-series data in that example (samples 2350 to 3550, inclusive),
    2. Using a Kraskov (KSG) estimator, algorithm 2, with 4 nearest neighbours.
    3. You can start from the Transfer Entropy demo on this data set, in your preferred environment, and modify it to compute MI..

# Exercise

2. Task:
   1. Compute Mutual Information between the heart and breath rate time-series data in that example (samples 2350 to 3550, inclusive),
   2. Using a Kraskov (KSG) estimator, algorithm 2, with 4 nearest neighbours.
   3. You can start from the Transfer Entropy demo on this data set, in your preferred environment, and modify it to compute MI..

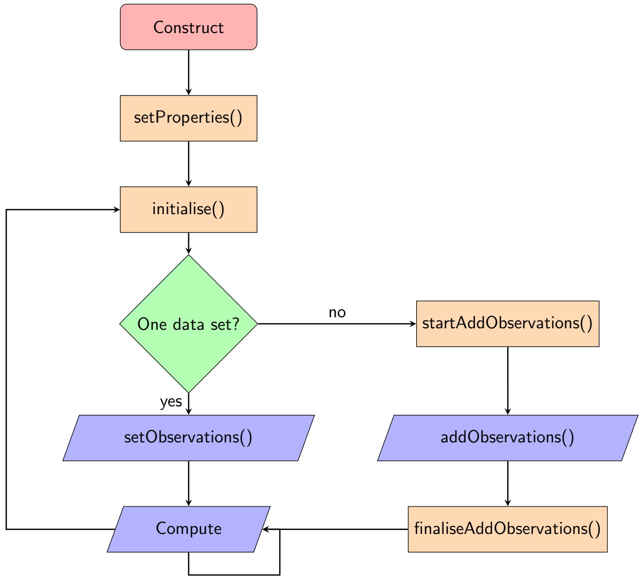
Answer: 0.123 nats.

# Exercise

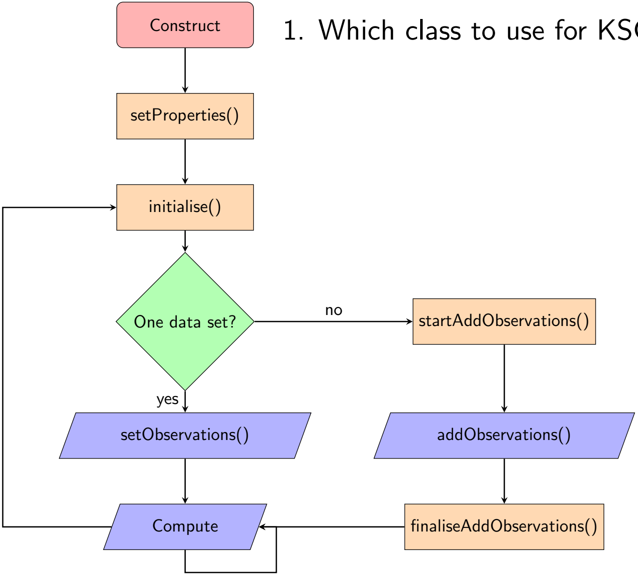
2. Task:
    1. Compute Mutual Information between the heart and breath rate time-series data in that example (samples 2350 to 3550, inclusive),
    2. Using a Kraskov (KSG) estimator, algorithm 2, with 4 nearest neighbours.
    3. You can start from the Transfer Entropy demo on this data set, in your preferred environment, and modify it to compute MI..
    4. Hint: You can see how a Kraskov MI calculator is used in Simple Demo 6 in the distribution.

Answer: 0.123 nats.

# Exercise – Hint – remember the usage paradigm!



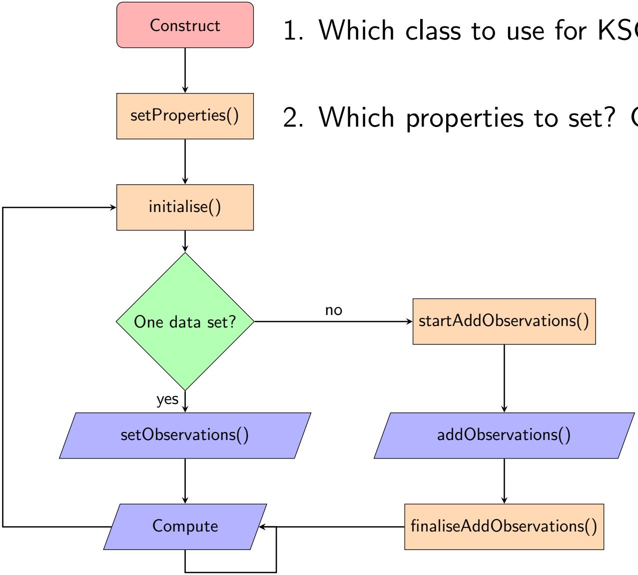
# Exercise – Hint – remember the usage paradigm!



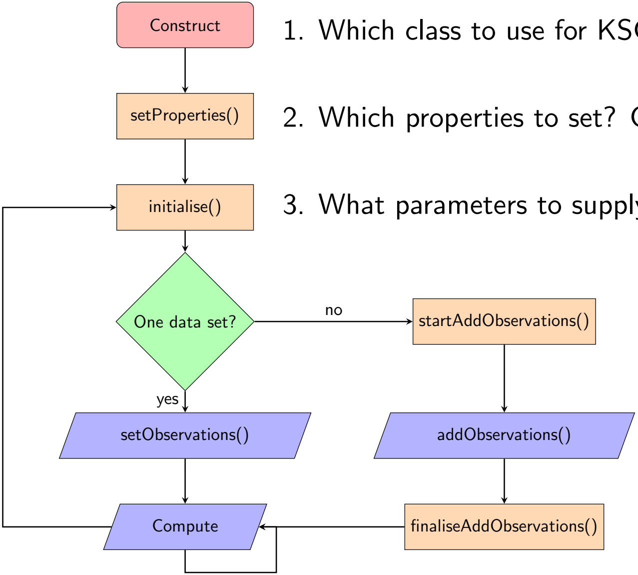

1. Which class to use for KSG MI algorithm 2?

# Exercise – Hint – remember the usage paradigm!

1. Which class to use for KSG MI algorithm 2?
2. Which properties to set? Check Javadocs!

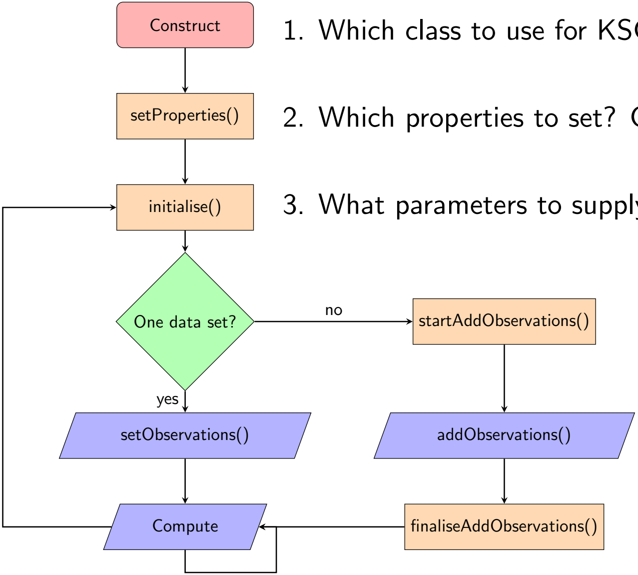


# Exercise – Hint – remember the usage paradigm!



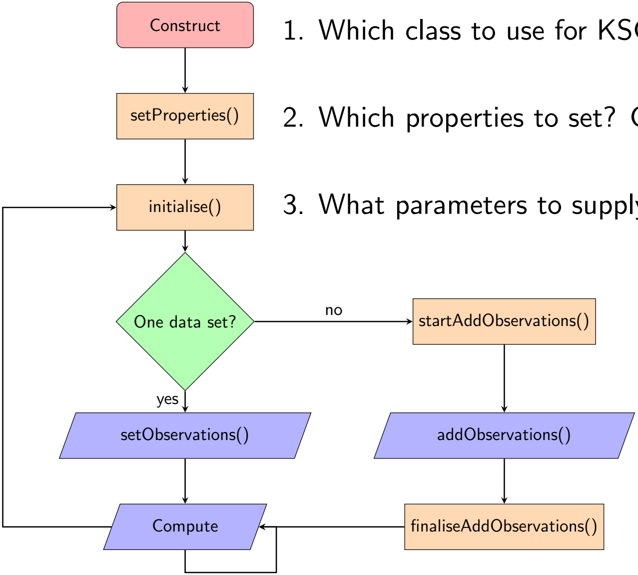

1. Which class to use for KSG MI algorithm 2?
2. Which properties to set? Check Javadocs!
3. What parameters to supply? Check Javadocs!

# Exercise – Hint – remember the usage paradigm!




1. Which class to use for KSG MI algorithm 2?
2. Which properties to set? Check Javadocs!
3. What parameters to supply? Check Javadocs!
4. Supply one set of data

# Exercise – Hint – remember the usage paradigm!




1. Which class to use for KSG MI algorithm 2?
2. Which properties to set? Check Javadocs!
3. What parameters to supply? Check Javadocs!
4. Supply one set of data
5. Calculate!

# Exercise – sample answer

```java
import infodynamics.measures.continuous.kraskov.MutualInfoCalculatorMultiVariateKraskov2;
// ...
// New KSG MI (algorithm 2) calculator:
miCalc = new MutualInfoCalculatorMultiVariateKraskov2();
miCalc.initialise(1,1); // univariate calculation
miCalc.setProperty("k", "4"); // 4 nearest neighbours
miCalc.setObservations(heart, chestVol);
double miHeartToBreath = miCalc.computeAverageLocalOfObservations();
```

# Debrief

How did you find the exercise?

Was it difficult? Which parts?

Did you know where to find the information you needed?

Any questions arising from the exercise?

# Exercise – Challenge task 1

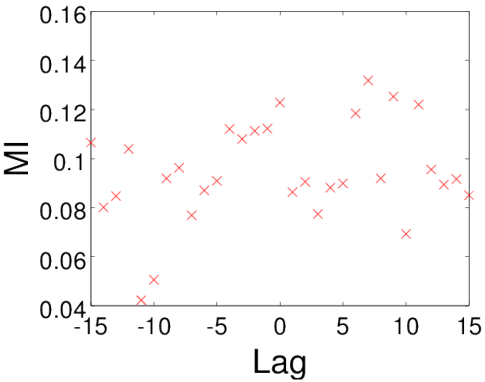
Extend to compute MI(heart;breath), for a variety of **lags** between the two time-series. E.g., investigate lags of [0, 1, ..., 14, 15].

**HINT**: You could either:

- shift the time-series with respect to eachother to affect the lag, or
- (cleaner) check out the available properties for this MI calculator in the Javadocs. (Challenge: what to do if you want to use negative lags, i.e. a positive lag from breath to heart, with this property?)

# Exercise – Challenge task 1

Extend to compute MI(heart;breath), for a variety of **lags** between the two time-series. E.g., investigate lags of [0, 1, ..., 14, 15].




What would you interpret from the results? Can you think of some logical further investigations here?

# Summary

What I wanted you to take away tody:

- Understand measures of information dynamics;
- Be able to obtain and install JIDT distribution;
- Understand and run sample scripts in their chosen environment;
- Be able to modify sample scripts for new analysis;
- Know how and where to seek support information (wiki, Javadocs, mailing list, twitter).

# Summary

What I wanted you to take away tody:

- Understand measures of information dynamics;
- Be able to obtain and install JIDT distribution;
- Understand and run sample scripts in their chosen environment;
- Be able to modify sample scripts for new analysis;
- Know how and where to seek support information (wiki, Javadocs, mailing list, twitter).

Did we get there?

# Summary

What I wanted you to take away tody:

- Understand measures of information dynamics;
- Be able to obtain and install JIDT distribution;
- Understand and run sample scripts in their chosen environment;
- Be able to modify sample scripts for new analysis;
- Know how and where to seek support information (wiki, Javadocs, mailing list, twitter).

Did we get there?
Did you get what you came here for?

# Summary

What I wanted you to take away tody:

- Understand measures of information dynamics;
- Be able to obtain and install JIDT distribution;
- Understand and run sample scripts in their chosen environment;
- Be able to modify sample scripts for new analysis;
- Know how and where to seek support information (wiki, Javadocs, mailing list, twitter).

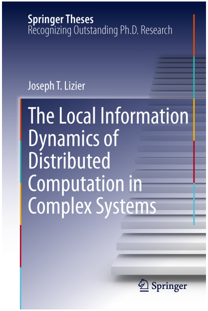
Did we get there?
Did you get what you came here for?
Any other questions?

# Final messages

We're seeking a Postdoc and PhD students for our Complex Systems group at University of Sydney – talk to me if interested

# Final messages

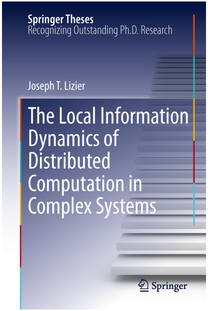
We're seeking a Postdoc and PhD students for our Complex Systems group at University of Sydney – talk to me if interested




*"The local information dynamics of distributed computation in complex systems"*, J. T. Lizier, Springer, Berlin (2013).

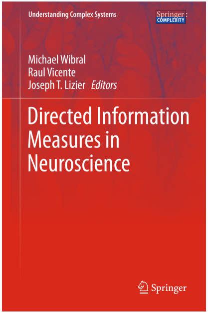
# Final messages

We're seeking a Postdoc and PhD students for our Complex Systems group at University of Sydney – talk to me if interested




*"The local information dynamics of distributed computation in complex systems"*, J. T. Lizier, Springer, Berlin (2013).

*"Directed information measures in neuroscience"*, edited by M. Wibral, R. Vicente and J. T. Lizier, Springer, Berlin (2014). In Press



# References I


T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley-Interscience, New York, 99th edition, Aug. 1991. ISBN 0471062596. URL http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20&path=ASIN/0471062596.

M. R. DeWeese and M. Meister. How to measure the information gained from one symbol. *Network: Computation in Neural Systems*, 10:325–340, 1999.

S. Frenzel and B. Pompe. Partial Mutual Information for Coupling Analysis of Multivariate Time Series. *Physical Review Letters*, 99(20):204101+, Nov. 2007. doi: 10.1103/physrevlett.99.204101. URL http://dx.doi.org/10.1103/physrevlett.99.204101.

G. Gomez-Herrero, W. Wu, K. Rutanen, M. C. Soriano, G. Pipa, and R. Vicente. Assessing coupling dynamics from an ensemble of time series. Aug. 2010. URL http://arxiv.org/abs/1008.0539.

A. Kraskov, H. Stögbauer, and P. Grassberger. Estimating mutual information. *Physical Review E*, 69(6):066138+, June 2004. doi: 10.1103/physreve.69.066138. URL http://dx.doi.org/10.1103/physreve.69.066138.

J. T. Lizier, M. Prokopenko, and A. Y. Zomaya. Local information transfer as a spatiotemporal filter for complex systems. *Physical Review E*, 77(2):026110+, Feb. 2008. doi: 10.1103/physreve.77.026110. URL http://dx.doi.org/10.1103/physreve.77.026110.

J. T. Lizier, M. Prokopenko, and A. Y. Zomaya. Information modification and particle collisions in distributed computation. *Chaos*, 20(3):037109+, 2010. doi: 10.1063/1.3486801. URL http://dx.doi.org/10.1063/1.3486801.

# References II

J. T. Lizier, J. Heinzle, A. Horstmann, J.-D. Haynes, and M. Prokopenko. Multivariate information-theoretic measures reveal directed information structure and task relevant changes in fMRI connectivity. *Journal of Computational Neuroscience*, 30 (1):85–107, 2011. doi: 10.1007/s10827-010-0271-2. URL http://dx.doi.org/10.1007/s10827-010-0271-2.

J. T. Lizier, M. Prokopenko, and A. Y. Zomaya. Local measures of information storage in complex distributed computation. *Information Sciences*, 208:39–54, Nov. 2012. ISSN 00200255. doi: 10.1016/j.ins.2012.04.016. URL http://dx.doi.org/10.1016/j.ins.2012.04.016.

J. T. Lizier, M. Prokopenko, and A. Y. Zomaya. Measuring the dynamics of information processing on a local scale in time and space. In M. Wibral, R. Vicente, and J. T. Lizier, editors, *Directed Information Measures in Neuroscience*. Springer Berlin Heidelberg, 2014. in press.

O. Miramontes. Order-disorder transitions in the behavior of ant societies. *Complexity*, 1(3):56–60, 1995.

M. Mitchell, J. P. Crutchfield, and P. T. Hraber. Evolving Cellular Automata to Perform Computations: Mechanisms and Impediments. *Physica D*, 75:361–391, 1994.

# References III

M. Mitchell, J. P. Crutchfield, and R. Das. Evolving Cellular Automata with Genetic Algorithms: A Review of Recent Work. In E. D. Goodman, W. Punch, and V. Uskov, editors, *Proceedings of the First International Conference on Evolutionary Computation and Its Applications, Moscow*, Russia, 1996. Russian Academy of Sciences. URL http://www.santafe.edu/~{}evca/evabstracts.html.

M. Prokopenko, P. Wang, P. Valencia, D. Price, M. Foreman, and A. Farmer. Self-Organizing Hierarchies in Sensor and Communication Networks. *Artificial Life*, 11(4):407–426, 2005.

M. Prokopenko, V. Gerasimov, and I. Tanev. Evolving Spatiotemporal Coordination in a Modular Robotic System. In S. Nolfi, G. Baldassarre, R. Calabretta, J. C. T. Hallam, D. Marocco, J.-A. Meyer, O. Miglino, and D. Parisi, editors, *From Animals to Animats 9: Proceedings of the Ninth International Conference on the Simulation of Adaptive Behavior (SAB'06)*, volume 4095 of *Lecture Notes in Computer Science*, pages 558–569. Springer, Berlin Heidelberg, 2006. doi: 10.1007/11840541_46. URL http://dx.doi.org/10.1007/11840541_46.

A. S. Ribeiro, S. A. Kauffman, J. Lloyd-Price, B. Samuelsson, and J. E. S. Socolar. Mutual information in random Boolean models of regulatory networks. *Physical Review E*, 77(1):011901, 2008.

D. R. Rigney, A. L. Goldberger, W. Ocasio, Y. Ichimaru, G. B. Moody, and R. Mark. Multi-Channel Physiological Data: Description and Analysis. In A. S. Weigend and N. A. Gershenfeld, editors, *Time Series Prediction: Forecasting the Future and Understanding the Past*, pages 105–129. Addison-Wesley, Reading, MA, 1993.

# References IV


T. Schreiber. Measuring Information Transfer. *Physical Review Letters*, 85(2): 461–464, July 2000. doi: 10.1103/physrevlett.85.461. URL http://dx.doi.org/10.1103/physrevlett.85.461.

V. Sperati, V. Trianni, and S. Nolfi. Evolving coordinated group behaviours through maximisation of mean mutual information. *Swarm Intelligence*, 2(2-4):73–95, 2008.

X. R. Wang, J. T. Lizier, T. Nowotny, A. Z. Berna, M. Prokopenko, and S. C. Trowell. Feature selection for chemical sensor arrays using mutual information. *PLoS ONE*, 9(3):e89840+, Mar. 2014. doi: 10.1371/journal.pone.0089840.

M. Wibral, N. Pampu, V. Priesemann, F. Siebenhühner, H. Seiwert, M. Lindner, J. T. Lizier, and R. Vicente. Measuring Information-Transfer Delays. *PLoS ONE*, 8(2): e55809+, Feb. 2013. doi: 10.1371/journal.pone.0055809. URL http://dx.doi.org/10.1371/journal.pone.0055809.

M. Wibral, J. T. Lizier, S. Vögler, V. Priesemann, and R. Galuske. Local active information storage as a tool to understand distributed neural information processing. *Frontiers in Neuroinformatics*, 8:1+, 2014. ISSN 1662-5196. doi: 10.3389/fninf.2014.00001. URL http://dx.doi.org/10.3389/fninf.2014.00001.

# References V

A. Wuensche. Classifying cellular automata automatically: Finding gliders, filtering, and relating space-time patterns, attractor basins, and the Z parameter. *Complexity*, 4(3):47–66, 1999. doi: 10.1002/(sici)1099-0526(199901/02)4:3\%3C47::aid-cplx9\%3E3.0.co;2-v. URL http://dx.doi.org/10.1002/(sici)1099-0526(199901/02)4:3%3C47::aid-cplx9%3E3.0.co;2-v.