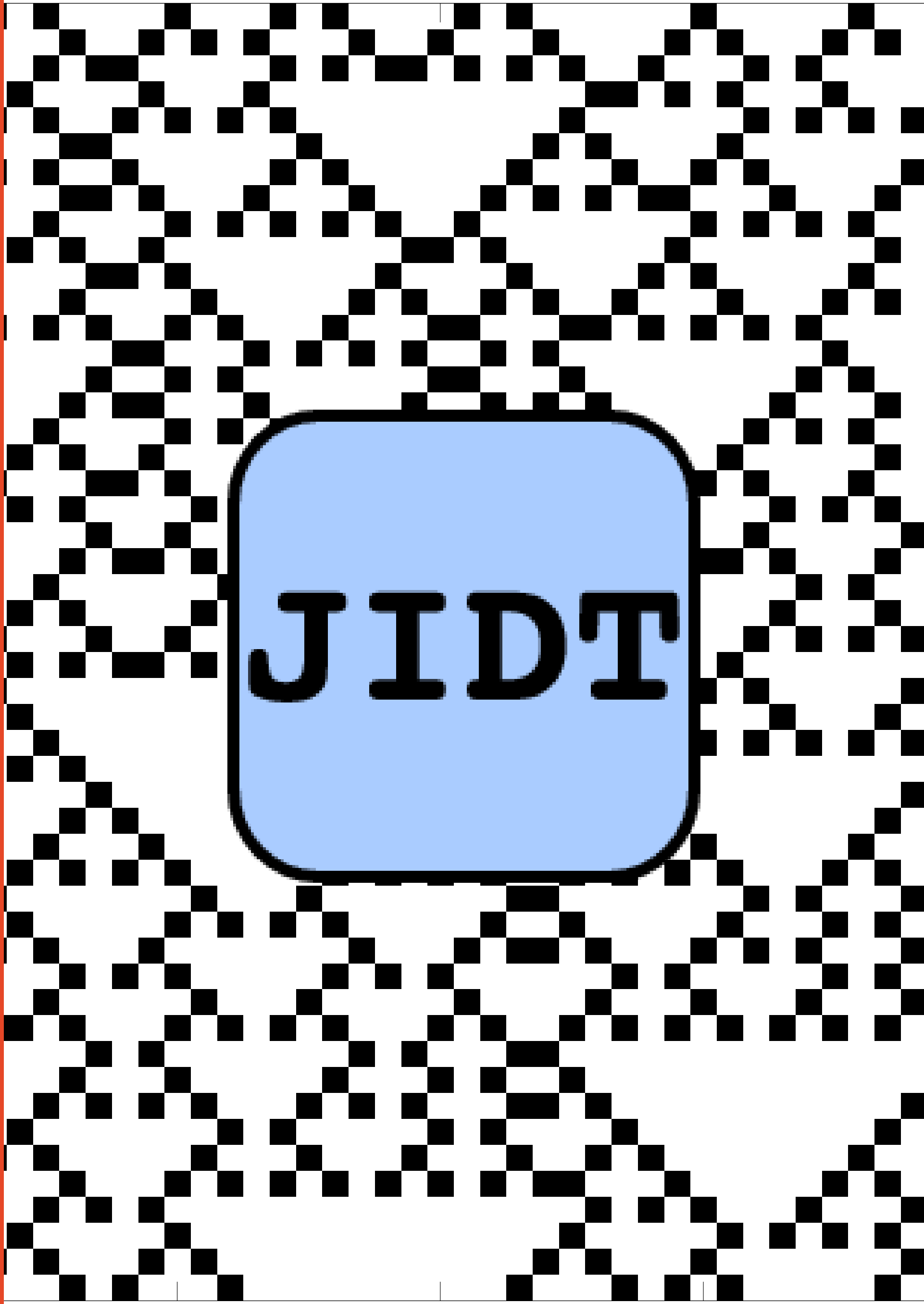
Introduction to JIDT

Dr. Joseph Lizier

THE UNIVERSITY OF SYDNEY



Introduction to JIDT: session outcomes

- Understanding of what JIDT offers for information-theoretic calculations;
- Ability to obtain and install JIDT distribution;
- Ability to use JIDT AutoAnalyser to make simple information-theoretic calculations on discrete data;
- Understand how and where to seek further information on JIDT.

- Primary references:
  - Lizier, "JIDT: An information-theoretic toolkit for studying the dynamics of complex systems", Frontiers in Robotics and AI, 1:11, 2014.

Java Information Dynamics Toolkit (JIDT)

- 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.).

- Available on github: http://github.com/jlizier/jidt/

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
  - A full tutorial and exercises, and course (*in progress*);
  - Full Javadocs;
  - A suite of demonstrations, including in each of the languages listed above;
  - A GUI for push-button analysis and code template generation.

- Code credits: JL, Ipek Özdemir, Pedro Martínez Mediano, …

Why use 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 version 1.2: (adapted table from paper)

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

– More now, and more coming (including Partial Information Decomposition) …

Why use 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://github.com/jlizier/jidt/wiki
  - GUI tool for easy calculation and code template generation!

Why implement in Java?

– Platform agnostic, requiring only a JVM;
– High performance coupled with
– Object-oriented code, with a hierarchical 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

- https://github.com/jlizier/jidt/wiki/Installation
- Beginners:
  1. Download the latest full distribution by following the Download link at https://github.com/jlizier/jidt/
  2. Unzip it to your preferred location for the distribution
- Advanced users:
  1. Take a git fork/clone at https://github.com/jlizier/jidt/
     a. Run `ant` (or better `ant dist`)

- To be able to use it, you will need the `infodynamics.jar` file on your classpath.

- That's it!

Installation – caveats

1. You'll need a JRE installed (Version $\geq$ 6)
   – Comes automatically with Matlab installation (maybe with some Octave-java or Python-JPype installations)
2. Advanced users / developers, you need:
   1. full Java SE / JDK to develop in Java or to change the source code;
   2. ant if you want to rebuild the project using build.xml;
   3. junit if you want to run the unit tests.
   4. CUDA installation if you want to utilise GPU (*documentation to come*)
3. Additional preparation may be required to use JIDT in GNU Octave or Python …

Check that your environment works

JIDT

- Java
  - Run `demos/java/example1TeBinaryData.sh` (linux/mac) or `.bat` (windows)
- Matlab/Octave
  - For Octave version < 3.8, first follow steps on the wiki, including installing `octave-java` from `octave-forge`.
  - Run `demos/octave/example1TeBinaryData.m`
- Python
  - Python2: `pip install jpype` to connect Python to Java
  - Python3: `pip install jpype1` to connect Python to Java
  - 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

JIDT

- Included in the distribution:
  - `readme.txt`;
  - `InfoDynamicsToolkit.pdf` – a pre-print of the publication introducing JIDT;
  - `tutorial` folder (a full tutorial presentation and sample exercises – also via JIDT wiki) and more detailed `course` folder;
  - `javadocs` folder (documents the methods and various options for each estimator class);
  - PDFs describing each demo in the `demos` folder;

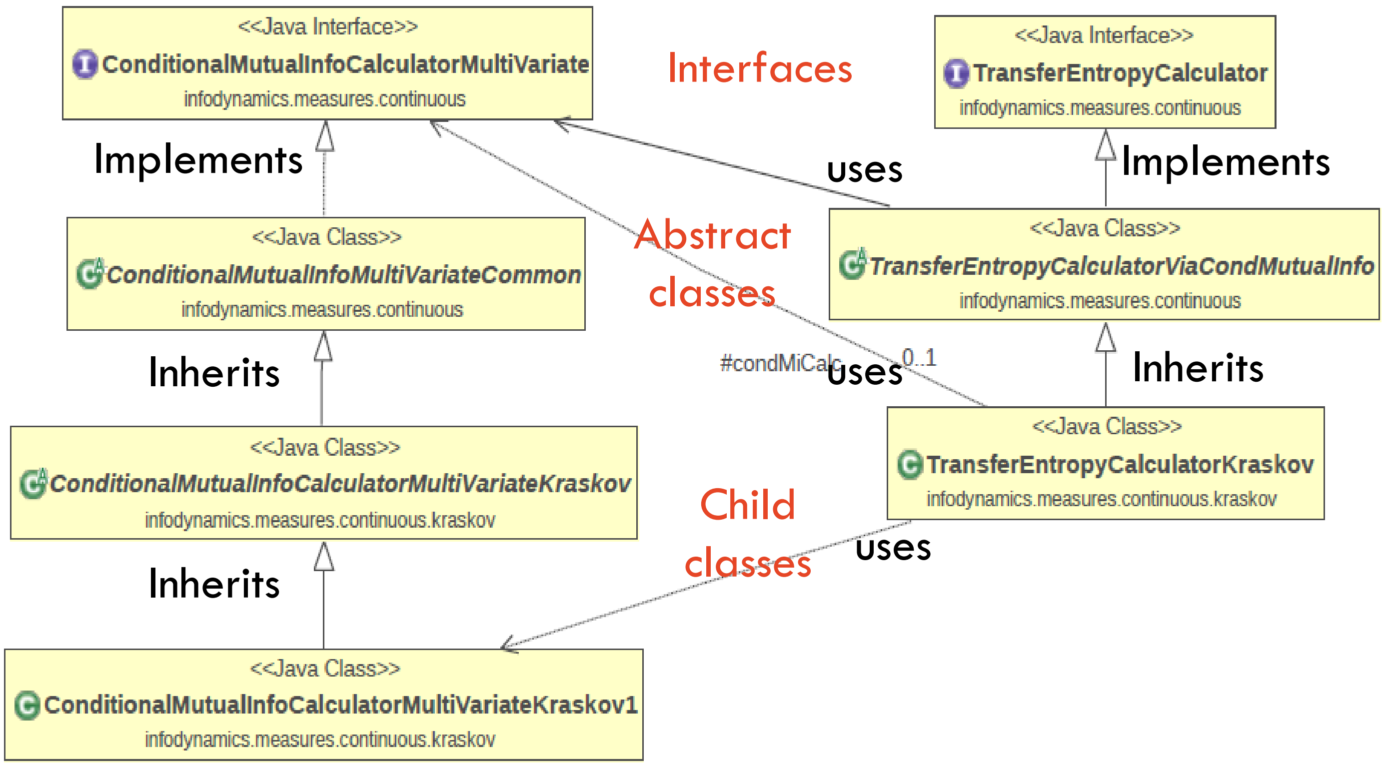
- Also see:
  - The wiki pages on the JIDT website
  - Our email discussion list `jidt-discuss` on Google groups

Source code structure

JIDT

- 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



Demonstrations

- JIDT is distributed with the following demos:
  - Auto-analyser GUI (code generator)
  - 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 (PDF and wiki pages) provided to help run them.

Auto Analyser GUI (Code Generator)

JIDT

- A GUI application to:
  - Make simple calculations for you (Entropy, MI, TE, etc);
  - Create code for you.

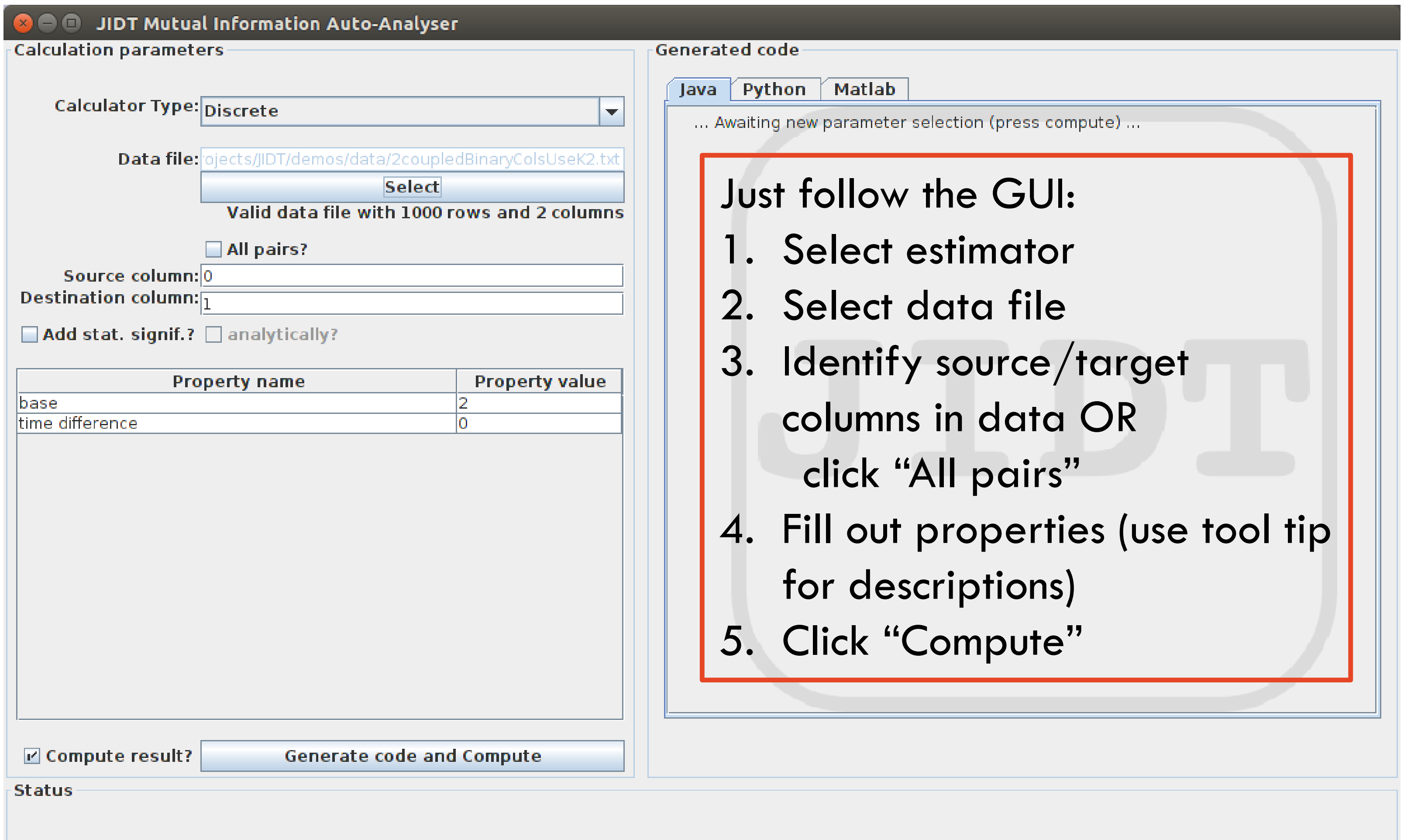
- For other measures, note that the general coding paradigm for all calculators is the same.

- To start the Auto Analyser, either:
  a. Double click the `infodynamics.jar` file, OR
  b. Scripts to start the apps are in the `demos/AutoAnalyser` folder:
     - `launchAutoAnalyser.sh` (and `.bat`)
     - (if on linux, need to `chmod u+x *.sh` in this folder)

- Run the AutoAnalyser and select a Mutual Information calculation

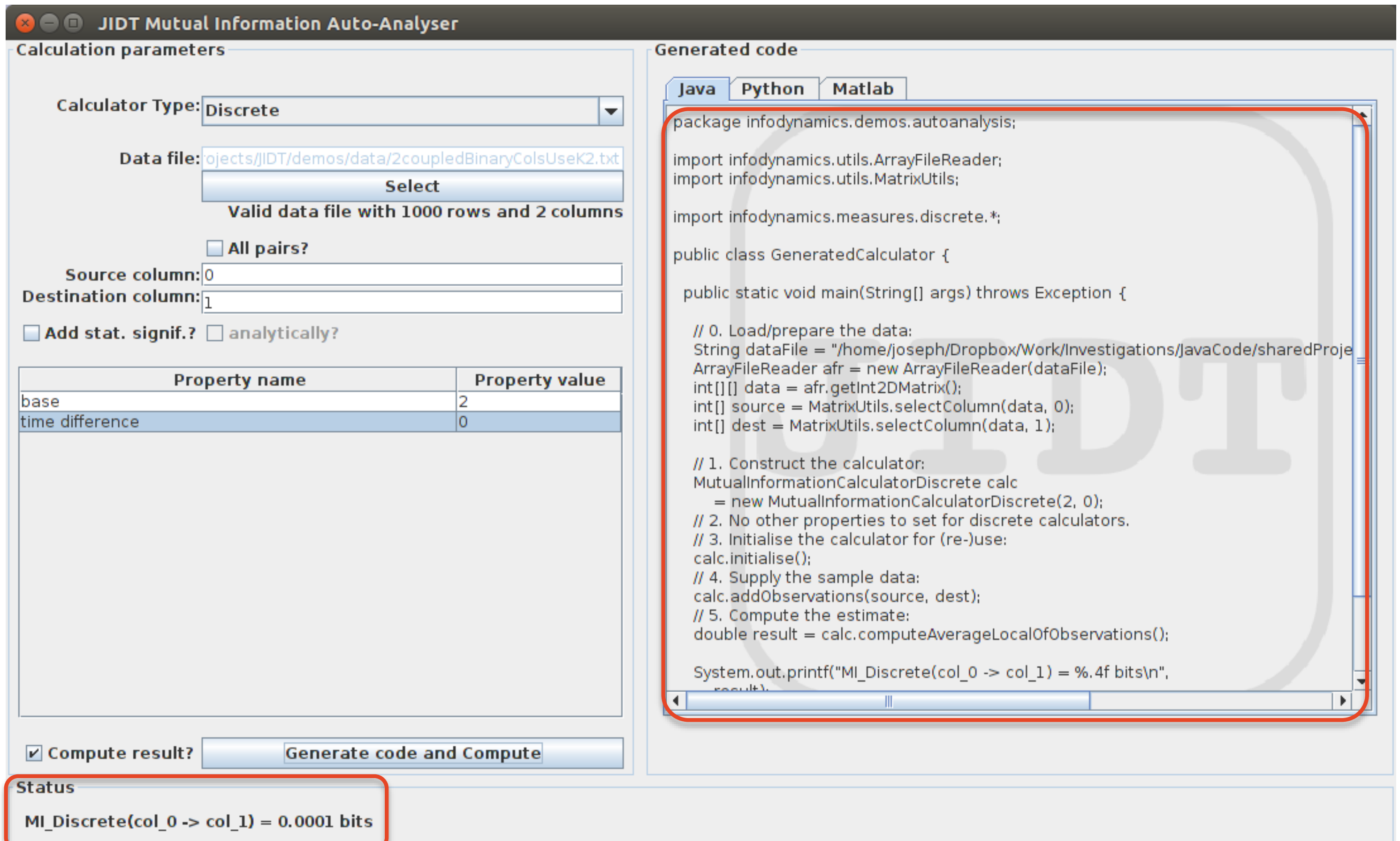
Auto Analyser GUI (Code Generator)

– Computing MI could not be easier:



Auto Analyser GUI (Code Generator)

– Clicking "Compute" then gives you:
  1. The estimated result of the measure
  2. Code to generate this calculation in Java, Python and Matlab



Auto Analyser GUI (Code Generator) – discrete

JIDT

- Let's generate a sample MI calculation on discrete data:
  - Select *Discrete* estimator
  - Select the data file `2coupledDiscreteCols.txt` from the default directory in the *Select* popup (`demos/data`).
    - Note: the GUI checks validity of the file
    - Valid file format is described when you hover on *Data file*
  - Leave source and destination columns
  - Set `base` to 4 (data range is 0…3). (*Try what happens without this*)
  - Click *Compute*

- Did everyone get the result 0.0007 bits?
- *Hover on the property names to see the description for them*
- Try changing "time difference" property to 1
  - Did you get 1.0002 bits? (The 2 bit variables have 1 coupled bit at lag 1)

Auto Analyser GUI – discrete – code analysis

– Let's examine the code that was generated:

– Either:

1. Click on the panel for the language you want to work in, OR
2. Open the file that was automatically generated for you:
   - `demos/java/infodynamics/demos/autoanalysis/GeneratedCalculator.java` OR
   - `demos/AutoAnalyser/GeneratedCalculator.m` OR
   - `demos/AutoAnalyser/GeneratedCalculator.py`

– You can run the automatically generated code (in the `demos/AutoAnalyser` folder) by:

  – Java: `shell> ./runAutoGenerated.sh` (or `.bat`)
  – Matlab: `matlab> GeneratedCalculator`
  – Python: `shell> python GeneratedCalculator.py`

Auto Analyser GUI – discrete – locating library

JIDT

- Observe how the classpath is pointed to `infodynamics.jar`:
  - Java: java command line in `demos/AutoAnalyser/runAutoGenerated.sh/.bat` (or in your IDE);
  - Matlab/Octave: `javaaddpath()` statement;
  - Python: `startJVM()` statement.

Auto Analyser GUI – discrete – 0. data format

- Discrete data (source and dest) represented as integer arrays:
  - Java: `int[]` array
  - (Python/Matlab use native array formats, conversion comes later)
- Data values in the range 0…*base*-1, where e.g. *base*=2 for binary data.
- Array is indexed by sample/observation number (for time-series measures the array is indexed by time).
- For multivariate time-series, we use 2D integer arrays, indexed first by sample index (or time) then variable number.
- Files can have comment lines starting with '%'
- Open the file `demos/data/2coupledDiscreteCols.txt` to inspect

Data for one variable

0
1
1
1
0
0

Sample index / time (row)

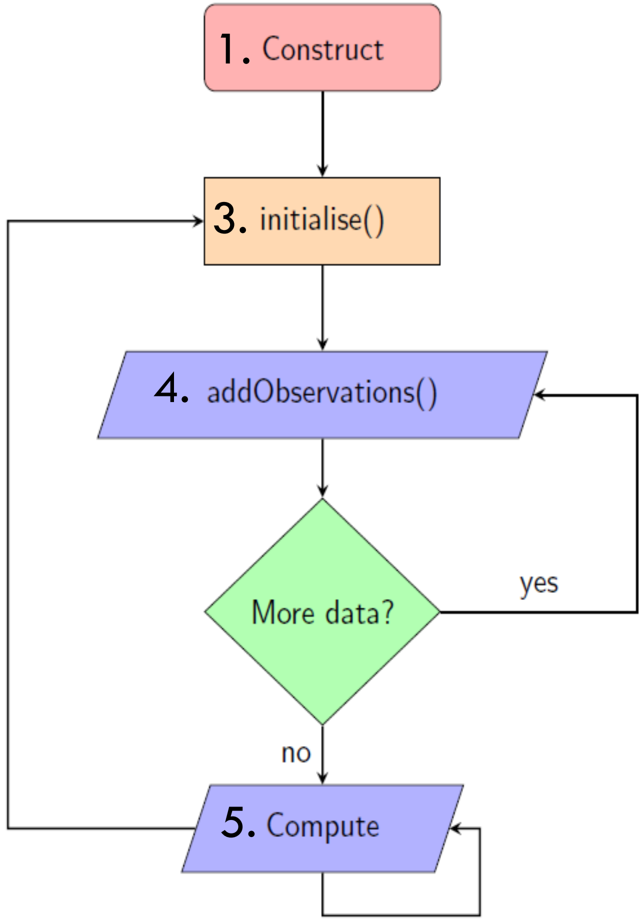
0	1	0	0	1	1	0	1	0
1	0	0	1	0	1	1	1	0
1	0	1	0	0	1	1	0	0
1	0	1	1	1	0	1	1	1
0	1	0	1	0	1	1	0	0
0	0	1	1	1	0	1	0	0

Multivariate data

Variable index (column)

Discrete data – usage paradigm

JIDT



Auto Analyser GUI – discrete – usage paradigm

JIDT

```
// int [] source and dest defined and loaded earlier
MutualInformationCalculatorDiscrete calc = new MutualInformationCalculatorDiscrete(4, 1);
calc.initialise();
calc.addObservations(source, dest);
double result = calc.computeAverageLocalOfObservations();
```

Java code

1. Construct the calculator, providing parameters
   – AutoAnalyser fills out any you need.
     - If not using AutoAnalyser always check Javadocs for which parameters are required.
   – Here the parameters are the number of possible discrete symbols per sample (4, binary), and source-target lag to compute MI across (1).
   – Constructor syntax is different for Matlab/Octave/Python – see generated code.

Auto Analyser GUI – discrete – usage paradigm

```
1 // int [] source and dest defined and loaded earlier
2 MutualInformationCalculatorDiscrete calc = new MutualInformationCalculatorDiscrete(4, 1);
3 calc.initialise();
4 calc.addObservations(source, dest);
5 double result = calc.computeAverageLocalOfObservations();
```

Java code

3. Initialise the calculator prior to:
   - use, or
   - re-use (e.g. looping back from line 5 back to line 3 to examine different data – see code for this by clicking "`All pairs`").
   - This clears PDFs ready for new samples.

Auto Analyser GUI – discrete – usage paradigm

JIDT

```java
// int [] source and dest defined and loaded earlier
MutualInformationCalculatorDiscrete calc = new MutualInformationCalculatorDiscrete(4, 1);
calc.initialise();
calc.addObservations(source, dest);
double result = calc.computeAverageLocalOfObservations();
```

Java code

4. Supply the data to the calculator to construct PDFs:
   - `addObservations()` may be called multiple times;
   - Convert arrays into Java format:
     - From Matlab/Octave using `octaveToJavaIntArray(array)`, etc., scripts (see `demos/octave` folder)
     - From Python using `JArray(JInt, numDims)(array)` for conversion or with numpy arrays via: `JArray(JInt, numDims)(numpyArray.tolist())`.

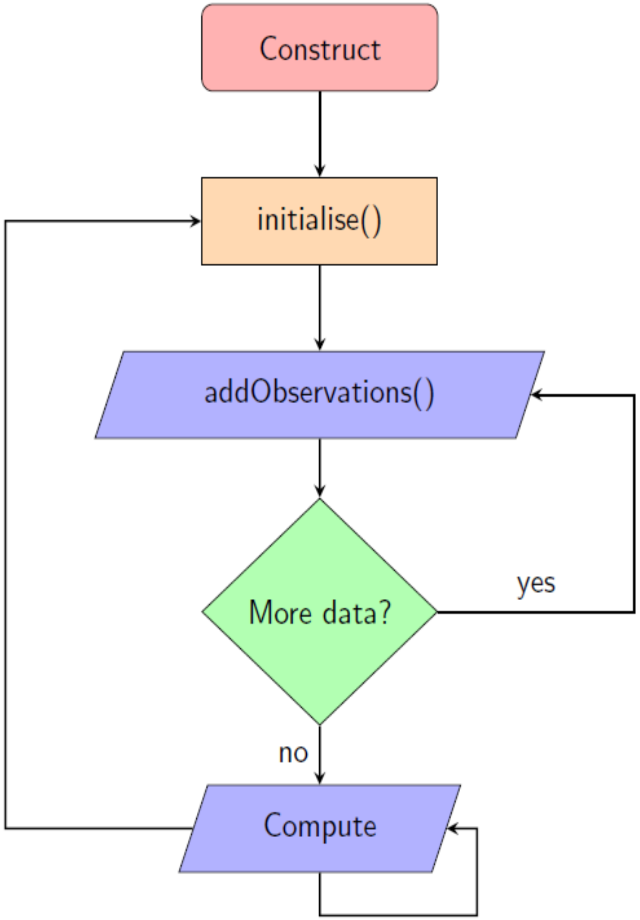
Auto Analyser GUI – discrete – usage paradigm

```java
// int [] source and dest defined and loaded earlier
MutualInformationCalculatorDiscrete calc = new MutualInformationCalculatorDiscrete(4, 1);
calc.initialise();
calc.addObservations(source, dest);
double result = calc.computeAverageLocalOfObservations();
```

Java code

5. Compute the measure:
   – Value is always returned in bits for discrete calculators.
   – Result here approaches 1 bit (destination copies 1 bit of the (random) 2 bit source from 1 time step in the past).
   – Other computations include:
     - `computeLocalOfPreviousObservations()` for local values
     - `computeSignificance()` to compute p-values of measures of predictability (see later session, Appendix A5 of paper).

Discrete data – usage paradigm



Introduction to JIDT: summary

- Our session outcomes were:
  - Understanding of what JIDT offers for information-theoretic calculations;
  - Ability to obtain and install JIDT distribution;
  - Ability to use JIDT AutoAnalyser to make simple information-theoretic calculations on discrete data;
  - Understand how and where to seek further information on JIDT.
- Did we get there?

- *Coming up*: Introduction of estimators for continuous data, and how to use these in JIDT.

Questions