

# Lecture 3 – Introduction to JIDT

Dr. Joseph Lizier



**JIDT**

# COMMONWEALTH OF AUSTRALIA

Copyright Regulations 1969

## WARNING

This material has been reproduced and communicated to you by or on behalf of the **University of Sydney** pursuant to Part VB of the Copyright Act 1968 (the Act).

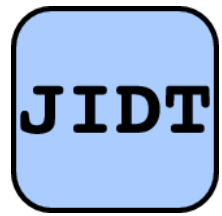
The material in this communication may be subject to copyright under the Act. Any further reproduction or communication of this material by you may be the subject of copyright protection under the Act.

Do not remove this notice

# Introduction to JIDT: session outcomes

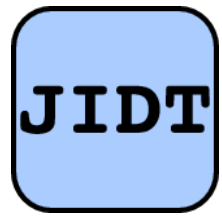
- Understanding of what JIDT offers for information-theoretic calculations;
  - Ability to obtain and install JIDT distribution;
  - Ability to use JIDT AutoAnalyser to make simple information-theoretic calculations on discrete data;
  - Understand how and where to seek further information on JIDT.
- 
- 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/ilizier/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](https://arxiv.org/abs/1408.3270)
  - A full tutorial and exercises;
  - 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?



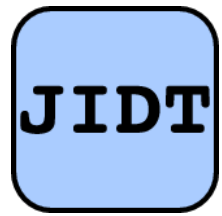
- JIDT is unique in the combination of features it provides:
  - Large array of measures, including all conditional/multivariate forms of the transfer entropy, and complementary measures such as active information storage.
  - Wide variety of estimator types and applicability to both discrete and continuous data

# Measure-estimator combinations

– As of version 1.2:

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

# 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/jlazier/jidt/wiki>



## Why implement in Java?

- Platform agnostic, requiring only a JVM;
- 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

## — Beginners:

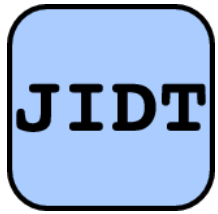
1. Download the latest full distribution by following the Download link at <https://github.com/jlazier/jidt/>
2. Unzip it to your preferred location for the distribution

## — Advanced users:

1. Take a git fork/clone at <https://github.com/jlazier/jidt/>

— 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-JType 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.
3. Additional preparation may be required to use JIDT in GNU Octave or Python ...

# Check that your environment works

## – 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

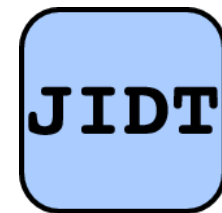
- Python2: `pip install jpype` to connect Python to Java
- Python3: `pip install jpype1` to connect Python to Java
- Run `demos/python/example1TeBinaryData.py`

- 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



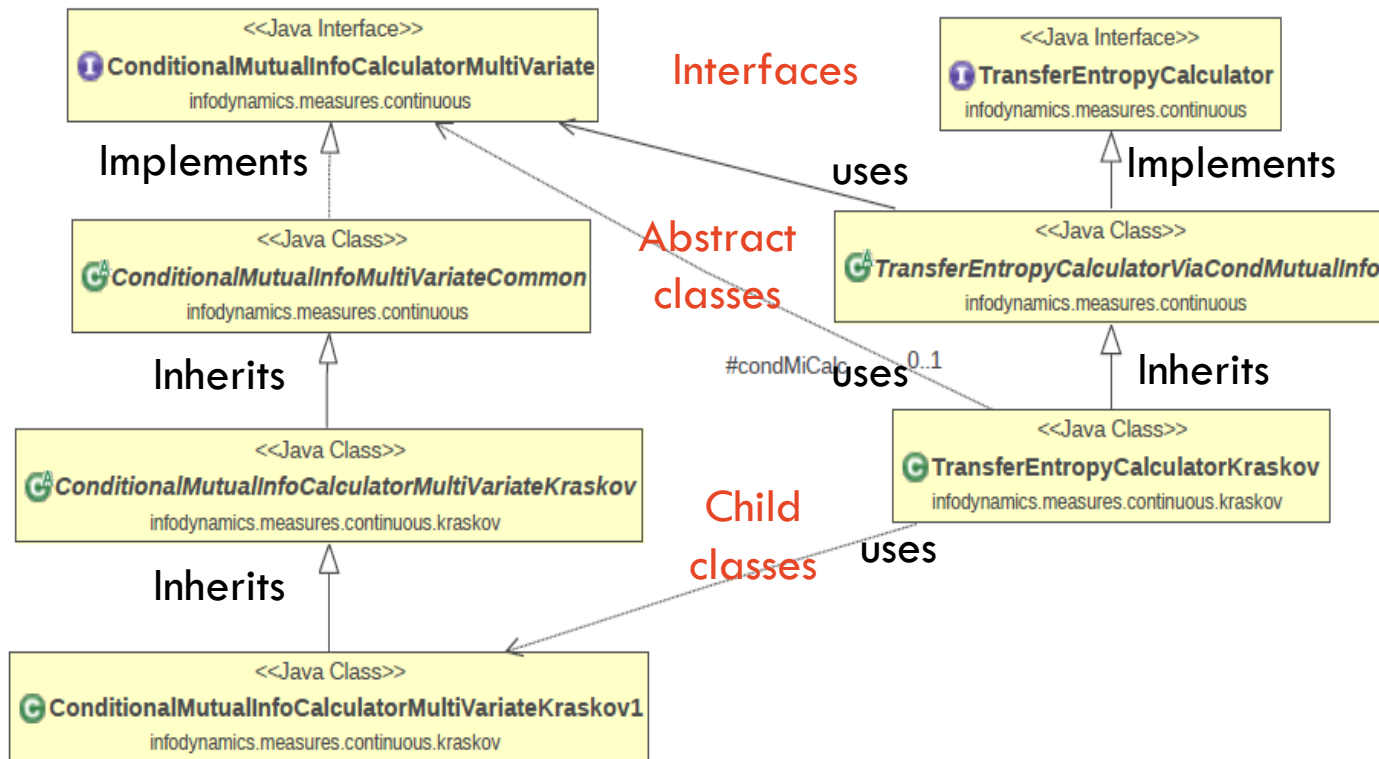
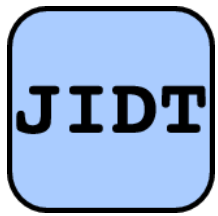
- Included in the distribution:
  - `readme.txt`;
  - `InfoDynamicsToolkit.pdf` – a pre-print of the publication introducing JIDT;
  - `tutorial` folder (a full tutorial presentation and sample exercises – also via [JIDT wiki](#))
  - `javadocs` folder (documents the methods and various options for each 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



- 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



Used under CC-BY license from: Lizier, Frontiers in Robotics & AI, 1:11, 2014



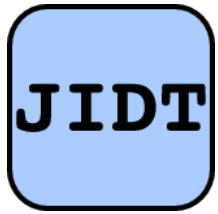
# Demonstrations

- 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)

- A GUI application to:
  - Make simple calculations for you (MI and TE);
  - Create code for you.
- For other measures, note that the general coding paradigm for all calculators is the same.
- Scripts to start the apps are in the `demos/AutoAnalyser` folder:
  - `runMIAutoAnalyser.sh` (and `.bat`) for MI, and
  - `runTEAutoAnalyser.sh` (and `.bat`) for TE.
- **Run:** `runMIAutoAnalyser.sh` (or `.bat`)

# Auto Analyser GUI (Code Generator)



- Computing MI could not be easier:

The screenshot shows the JIDT Mutual Information Auto-Analyser GUI. The window is titled "JIDT Mutual Information Auto-Analyser". It is divided into two main panels: "Calculation parameters" on the left and "Generated code" on the right.

**Calculation parameters panel:**

- Calculator Type:** A dropdown menu set to "Discrete".
- Data file:** A text field containing "objects/JIDT/demos/data/2coupledBinaryColsUseK2.txt" and a "Select" button below it. A note below the button says "Valid data file with 1000 rows and 2 columns".
- All pairs?:** An unchecked checkbox.
- Source column:** A text field containing "0".
- Destination column:** A text field containing "1".
- Add stat. signif.?:** Two unchecked checkboxes labeled "analytically?".
- Properties table:** A table with two columns: "Property name" and "Property value".

Property name	Property value
base	2
time difference	0
- Buttons:** A checked checkbox "Compute result?" and a "Generate code and Compute" button.
- Status:** A section at the bottom of the panel.

**Generated code panel:**

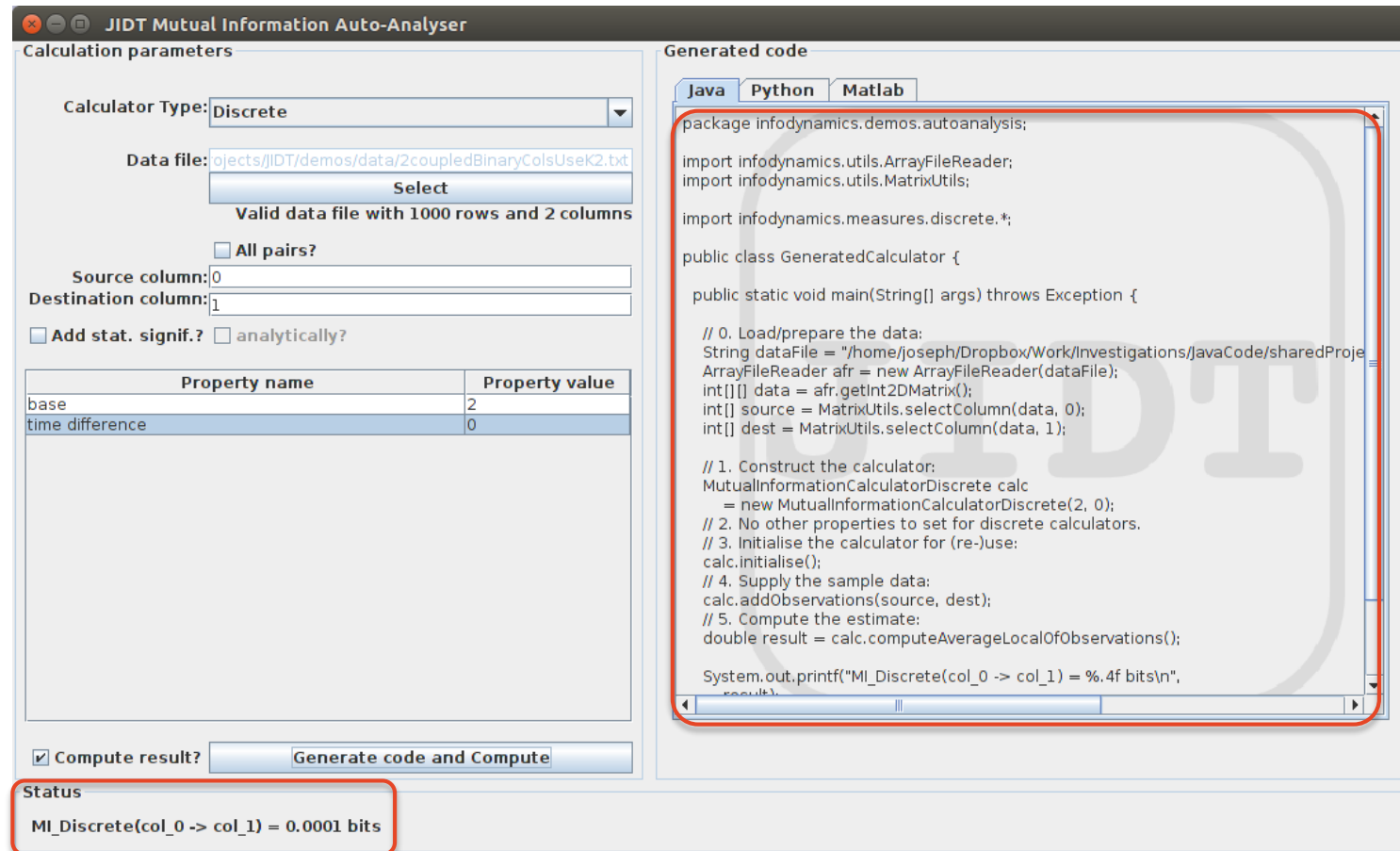
- Three tabs: "Java", "Python", and "Matlab".
- A text area containing the message: "... Awaiting new parameter selection (press compute) ...".

Just follow the GUI:

1. Select estimator
2. Select data file
3. Identify source/target columns in data OR click "All pairs"
4. Fill out properties (use tool tip for descriptions)
5. Click "Compute"

# 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



**JIDT Mutual Information Auto-Analyser**

**Calculation parameters**

Calculator Type: **Discrete**

Data file: **objects/JIDT/demos/data/2coupledBinaryColsUseK2.txt**  
 Select  
 Valid data file with 1000 rows and 2 columns

☐ All pairs?

Source column: **0**  
 Destination column: **1**

☐ Add stat. signif. ☐ analytically?

Property name	Property value
base	2
time difference	0

☒ Compute result? **Generate code and Compute**

**Status**  
 MI\_Discrete(col\_0 -> col\_1) = 0.0001 bits

**Generated code**

**Java** **Python** **Matlab**

```
package infodynamics.demos.autoanalysis;

import infodynamics.utils.ArrayFileReader;
import infodynamics.utils.MatrixUtils;

import infodynamics.measures.discrete.*;

public class GeneratedCalculator {

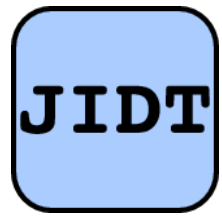
    public static void main(String[] args) throws Exception {

        // 0. Load/prepare the data:
        String dataFile = "/home/joseph/Dropbox/Work/Investigations/JavaCode/sharedProje
        ArrayFileReader afr = new ArrayFileReader(dataFile);
        int[][] data = afr.getInt2DMatrix();
        int[] source = MatrixUtils.selectColumn(data, 0);
        int[] dest = MatrixUtils.selectColumn(data, 1);

        // 1. Construct the calculator:
        MutualInformationCalculatorDiscrete calc
            = new MutualInformationCalculatorDiscrete(2, 0);
        // 2. No other properties to set for discrete calculators.
        // 3. Initialise the calculator for (re-)use:
        calc.initialise();
        // 4. Supply the sample data:
        calc.addObservations(source, dest);
        // 5. Compute the estimate:
        double result = calc.computeAverageLocalOfObservations();

        System.out.printf("MI_Discrete(col_0 -> col_1) = %.4f bits\n",
            result);
    }
}
```

# Auto Analyser GUI (Code Generator) – discrete



- 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



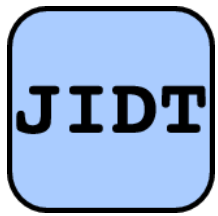
- Let's examine the code that was generated:
- Either:
  1. Click on the panel for the language you want to work in, OR
  2. Open the file that was automatically generated for you:
    - `demos/java/infodynamics/demos/autoanalysis/GeneratedCalculator.java` OR
    - `demos/AutoAnalyser/GeneratedCalculator.m`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

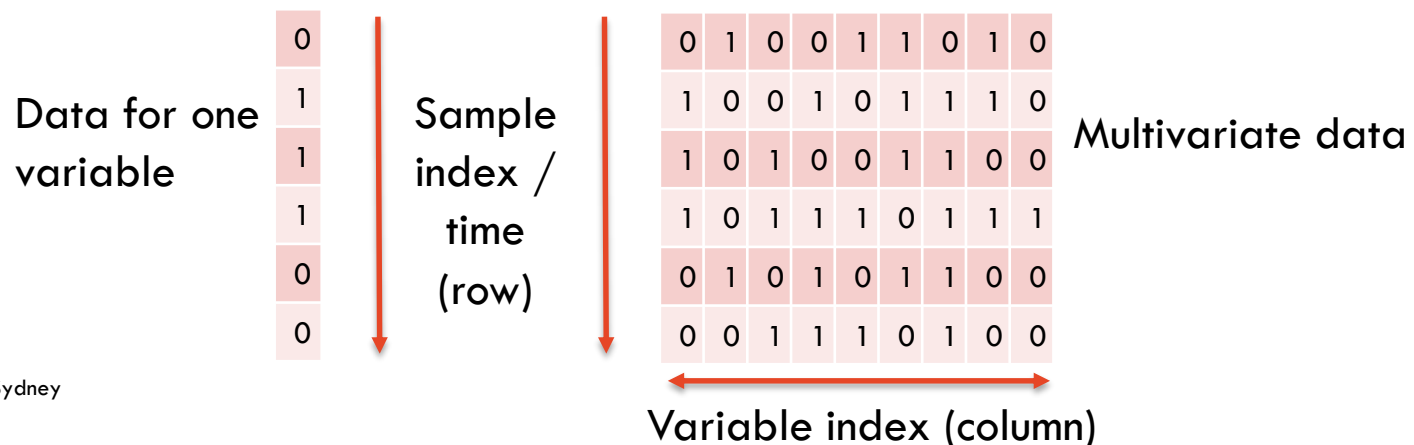


- Observe how the classpath is pointed to `infodynamics.jar`:
  - **Java**: java command line in  
`demos/AutoAnalyser/runAutoGenerated.sh/.bat` (or in your IDE);
  - **Matlab/Octave**: `javaaddpath()` statement;
  - **Python**: `startJVM()` statement.

# Auto Analyser GUI – discrete – 0. data format

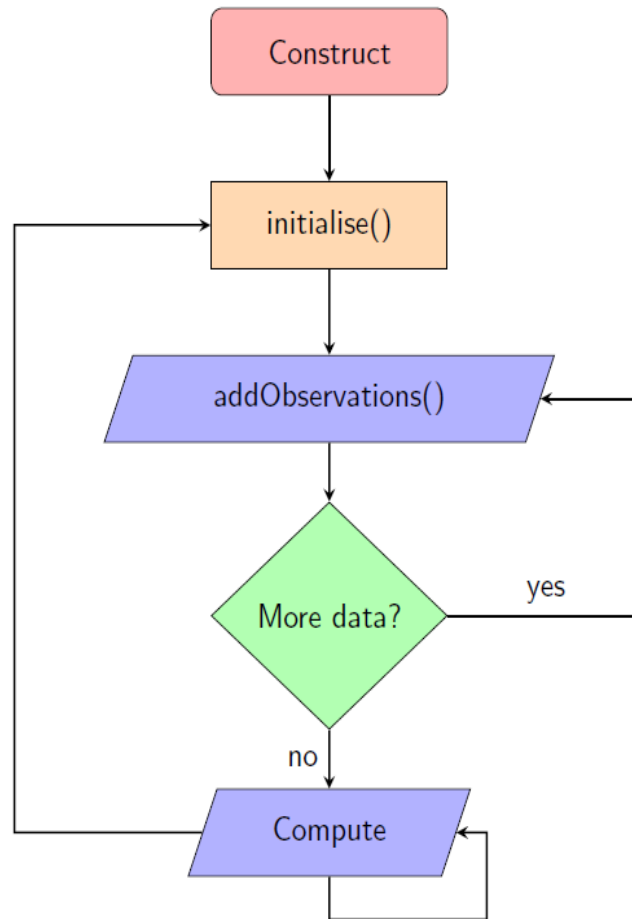
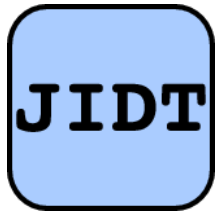


- **Discrete** data (source and dest) represented as **integer arrays**:
  - Java: `int[]` array
  - (Python/Matlab use native array formats, conversion comes later)
- Data values in the range  $0 \dots \text{base}-1$ , where e.g.  $\text{base}=2$  for binary data.
- Array is indexed by sample/observation 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

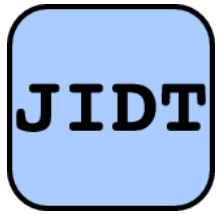




# Discrete data – usage paradigm



# Auto Analyser GUI – discrete – usage paradigm



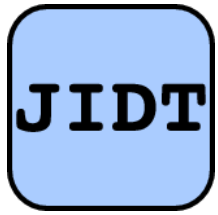
```
1 // int [] source and dest defined and loaded earlier
2 MutualInformationCalculatorDiscrete calc = new MutualInformationCalculatorDiscrete(4, 1);
3 calc.initialise();
4 calc.addObservations(source, dest);
5 double result = calc.computeAverageLocalOfObservations();
```

*Java code*

## 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



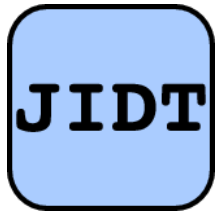
```
1 // int [] source and dest defined and loaded earlier
2 MutualInformationCalculatorDiscrete calc = new MutualInformationCalculatorDiscrete(4, 1);
3 calc.initialise();
4 calc.addObservations(source, dest);
5 double result = calc.computeAverageLocalOfObservations();
```

*Java code*

## 2. 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



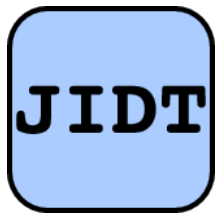
```
1 // int [] source and dest defined and loaded earlier
2 MutualInformationCalculatorDiscrete calc = new MutualInformationCalculatorDiscrete(4, 1);
3 calc.initialise();
4 calc.addObservations(source, dest);
5 double result = calc.computeAverageLocalOfObservations();
```

*Java code*

## 3. 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



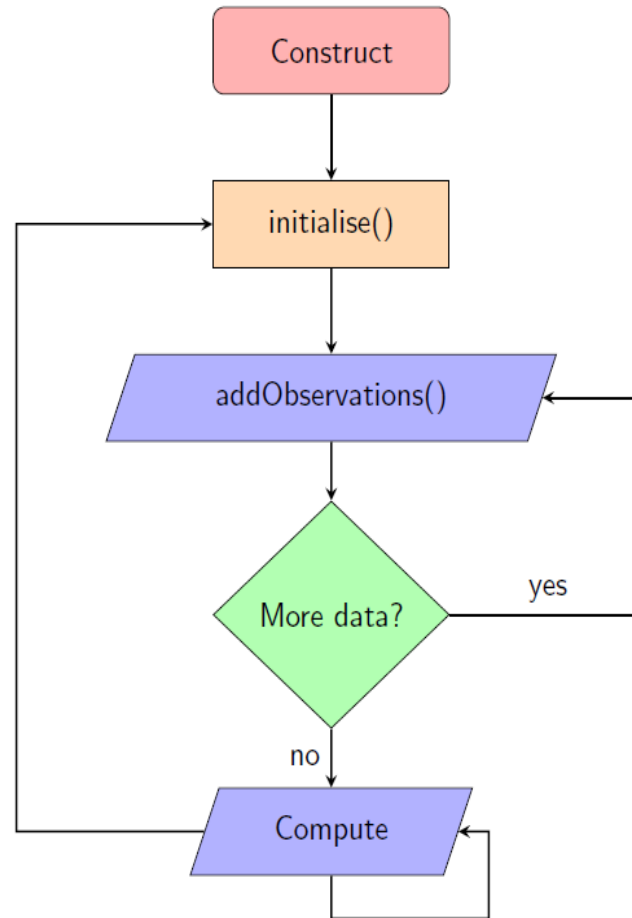
```
1 // int [] source and dest defined and loaded earlier
2 MutualInformationCalculatorDiscrete calc = new MutualInformationCalculatorDiscrete(4, 1);
3 calc.initialise();
4 calc.addObservations(source, dest);
5 double result = calc.computeAverageLocalOfObservations();
```

*Java code*

## 4. 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 Lecture 6, 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?
- *Next lecture:* Introduction of estimators for continuous data, and how to use these in JIDT.

# Questions



THE UNIVERSITY OF  
SYDNEY