Model Exploration Module

Tutorial

Prepared by

Róbert Mészáros

And

Márton Iványi

Multi-Agent Simulation Suite

AITIA

August 2007, Budapest

Table of content

Table of content 1

1 Introduction 2

1.1 Iterated Prisoner's Dilemma 2

2 Importing 3

3 Creating a view table 6

3.1 Computations 7

3.1.1 Aggregative operations 7

3.1.2 Scripting 8

3.2 Sorting 8

3.3 View name and description 9

4 Visualization 9

4.1 Creating a bar chart 10

4.2 Creating a line chart 11

4.3 Zooming 12

4.4 Exporting and formatting charts 13

4.5 Saving the chart collection 13

5 Notes 14

6 References 14

7 Appendix 15

7.1 PrisonersAgent.java 15

7.2 PrisonersModel.java 15

7.3 PrisonersModelBatch.java 17

7.4 PrisonersModelGUI.java 18

1 Introduction

This tutorial introduces you to the MEME program through an example model. You will be shown how MEME works and how it can help you to store, sort and visualize data from multiple runs of simulations.

MEME is a tool for running simulations in batch mode, using various parameter sets and handling the thus created data. It enables the user to store and systemize the raw data in databases(s), create proper datasets and visualize them.

Please consult the MEME User Manual (*MEME_User_Manual.pdf*) for general and installation information.

We will use an example model, called Iterated Prisoner’s Dilemma to present the use and advantages of MEME. This tutorial guides you through the import data created by the batch running of a Repast model and the analysis and visualization of the data extracted from the batch mode runs.

1.1 Iterated Prisoner’s Dilemma

In game theory, the prisoner's dilemma (PD) is a type of non-zero-sum game in which two players can "cooperate" with or "defect" (i.e. betray) the other player. In this game, as in all game theory, the only concern of each individual player ("prisoner") is maximizing his/her own payoff, without any concern for the other player's payoff. In the classic form of this game, cooperating is strictly dominated by defecting, so that the only possible equilibrium for the game is for all players to defect. In simpler terms, no matter what the other player does, one player will always gain a greater payoff by playing defect. Since in any situation playing defect is more beneficial than cooperating, all rational players will play defect.

The unique equilibrium for this game is a Pareto-suboptimal solution—that is, rational choice leads the two players to both play defect even though each player's individual reward would be greater if they both played cooperate. In equilibrium, each prisoner chooses to defect even though both would be better off by cooperating, hence the dilemma.

In the iterated prisoner's dilemma the game is played repeatedly. Thus each player has an opportunity to "punish" the other player for previous non-cooperative play. Cooperation may then arise as an equilibrium outcome. The incentive to defect is overcome by the threat of punishment, leading to the possibility of a cooperative outcome. If the game result is infinitely repeated, cooperation may be Nash equilibrium although both players defecting always remain equilibrium.

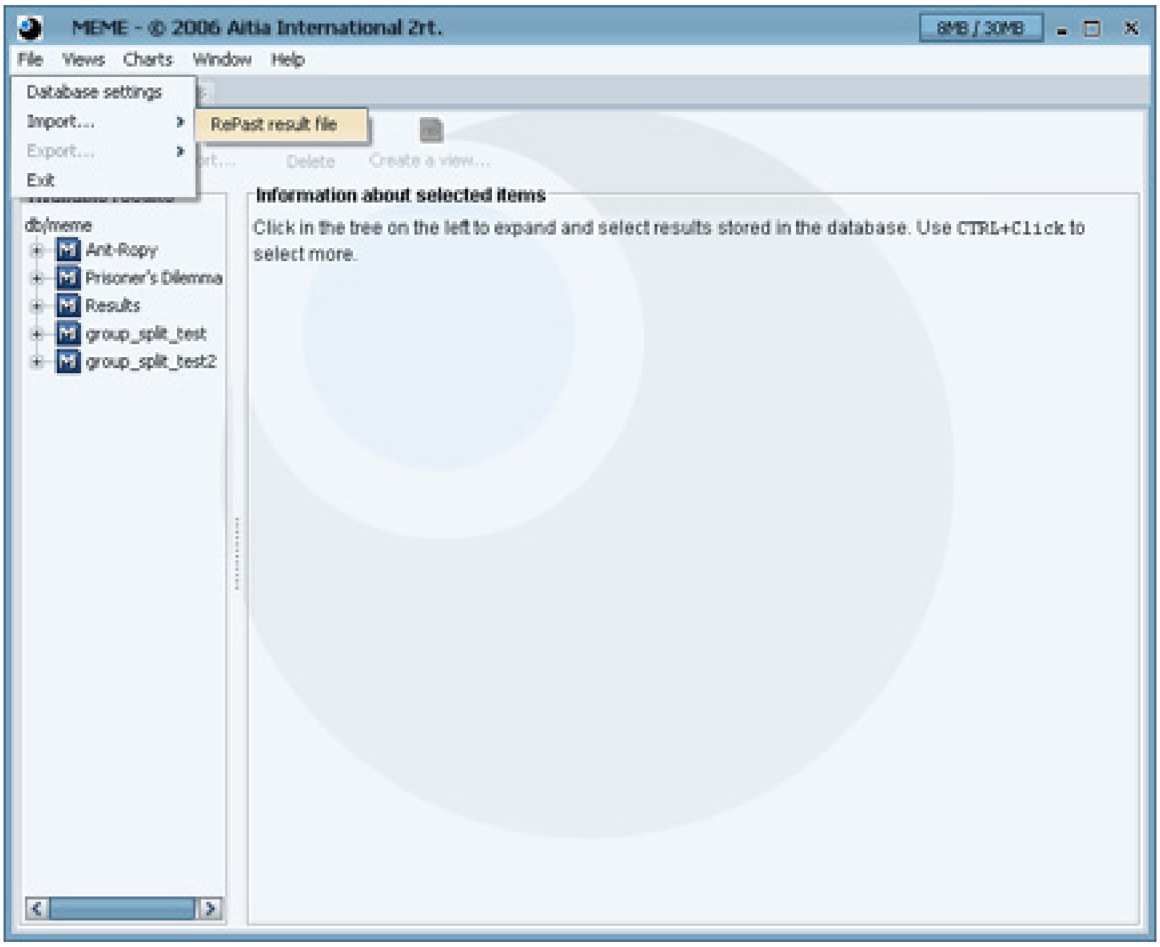
In his book The Evolution of Cooperation (1984), Robert Axelrod explored an extension to the classical PD scenario, which he called the iterated prisoner's dilemma (IPD). In this, participants have to choose their mutual strategy again and again, and have memory of their previous encounters. Axelrod invited academic colleagues all over the world to devise computer strategies to compete in an IPD tournament. The programs that were entered varied widely in algorithmic complexity; initial hostility; capacity for forgiveness; and so forth.

Axelrod discovered that when these encounters were repeated over a long period of time with many players, each with different strategies, "greedy" strategies tended to do very poorly in the long run while more "altruistic" strategies did better, as judged purely by self-interest. He used this to show a possible mechanism for the evolution of altruistic behavior from mechanisms that are initially purely selfish, by natural selection.

The best deterministic strategy was found to be "Tit for Tat", which Anatol Rapoport developed and entered into the tournament. It was the simplest of any program entered, containing only four lines of BASIC, and won the contest. The strategy is simply to cooperate on the first iteration of the game; after that, the player does what his opponent did on the previous move. A slightly better strategy is "Tit for Tat with forgiveness". When the opponent defects, on the next move, the player sometimes cooperates anyway, with a small probability (around 1%-5%). This allows for occasional recovery from getting trapped in a cycle of defections. The exact probability depends on the line-up of opponents. "Tit for Tat with forgiveness" is best when miscommunication is introduced to the game — when one's move is incorrectly reported to the opponent.

2 Importing

Start MEME and select the *File/Import/RePast result file* menu (see below).



Click in the tree on the left to expand and select results stored in the database. Use CTRL+Click to select more.

Figure 1 - Import > Repast result file

Select the Repast result file included with the MEME installer. The default destination of the file is C:\Program Files\MASS\MEME\Documents\Tutorial\IPD.out. If you chose a different installation folder it is implicitly <...>\Documents\Tutorial\IPD.out.

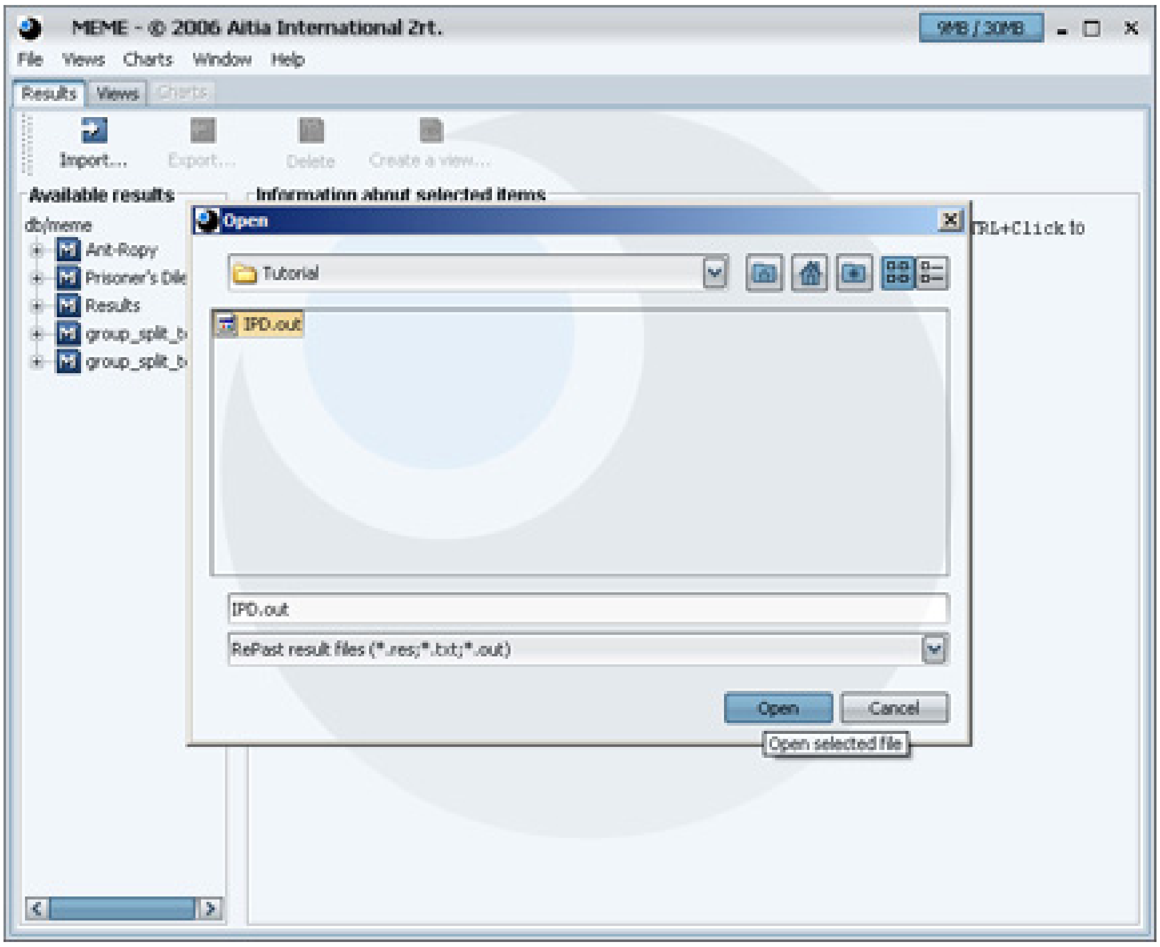
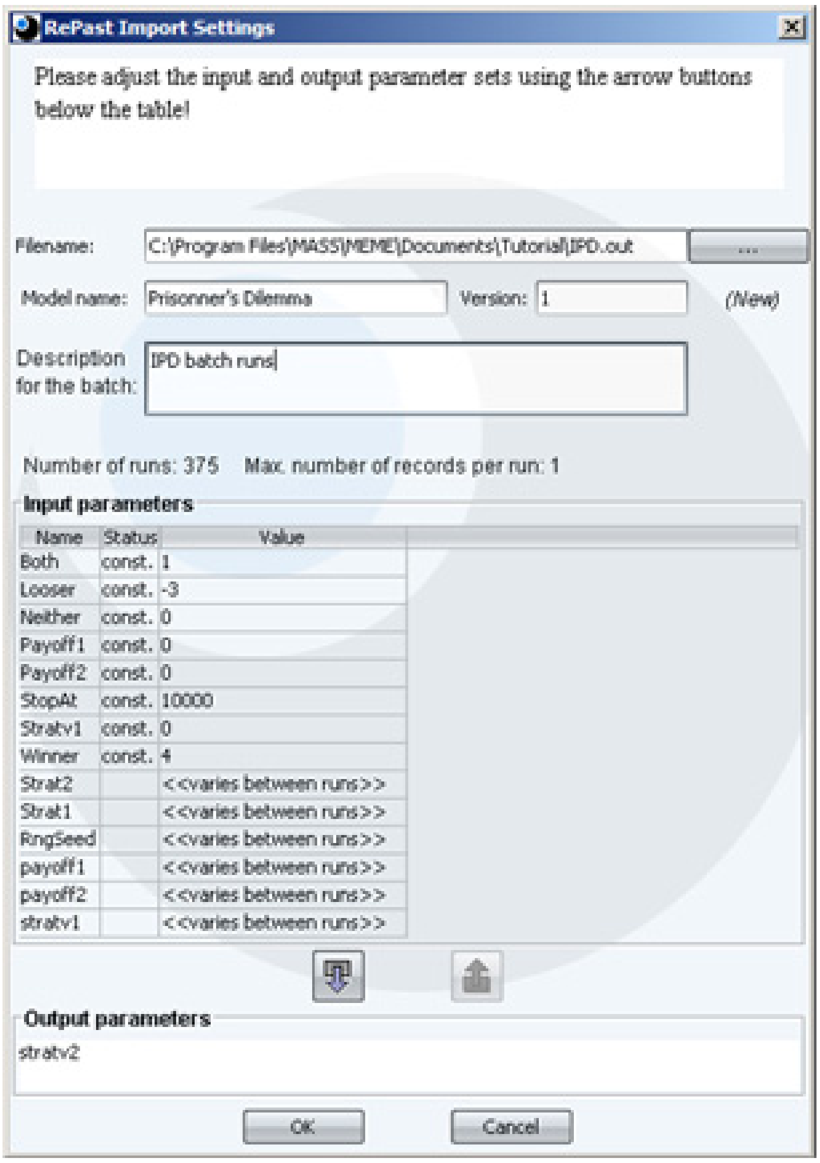

MEME - © 2006 Aitia International Zrt.
9MB / 30MB
File Views Charts Window Help
Results Views Charts
Import... Export... Delete Create a view...
Available results
Information about selected items
db/meme
Ant-Ropy
Prisoner's Dile
Results
group_split_b
group_split_b
Open
Tutorial
IPD.out
IPD.out
RePast result files (*.res;*.txt;*.out)
Open
Cancel
Open selected file
TRL+Click to

Figure 2 - Locate IPD.out

Upon pressing the *Open* button the Repast Import Settings dialogue appears (see below).



RePast Import Settings

Please adjust the input and output parameter sets using the arrow buttons below the table!

Filename: C:\Program Files\MASS\MEME\Documents\Tutorial\IPD.out

Model name: Prisonner's Dilemma Version: 1 (New)

Description for the batch: IPD batch runs

Number of runs: 375 Max. number of records per run: 1

Input parameters

Name	Status	Value
Both	const.	1
Looser	const.	-3
Neither	const.	0
Payoff1	const.	0
Payoff2	const.	0
StopAt	const.	10000
Stratv1	const.	0
Winner	const.	4
Strat2		<<varies between runs>>
Strat1		<<varies between runs>>
RngSeed		<<varies between runs>>
payoff1		<<varies between runs>>
payoff2		<<varies between runs>>
stratv1		<<varies between runs>>

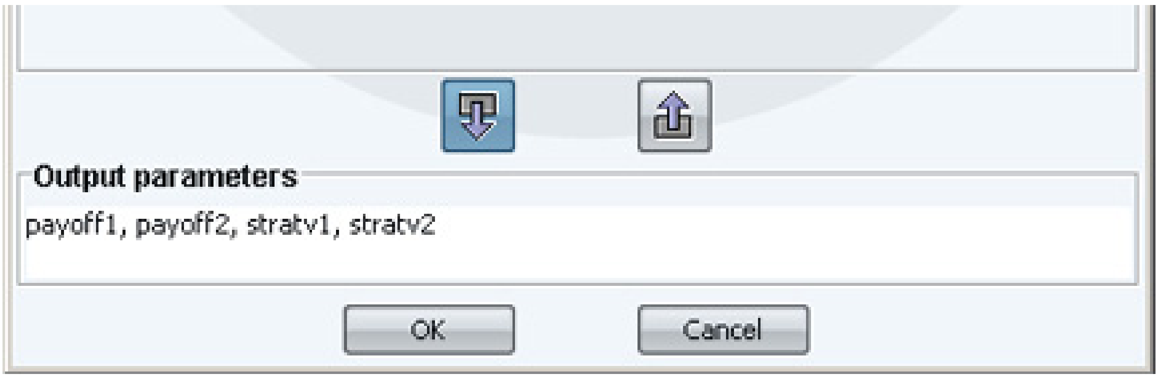
Output parameters

stratv2

OK Cancel

Figure 3 – Import settings

In the *Model name* field enter Prisoner's Dilemma as shown in the above picture. You can also provide (optional) a description in the *Description of the batch* field. Click the Add button between the *Output parameters* and *Input parameters* boxes six times to have RngSeed, Strat1, Strat2, payoff1, payoff2, stratv1 and startv2 added as output parameters.



Output parameters

payoff1, payoff2, stratv1, stratv2

OK Cancel

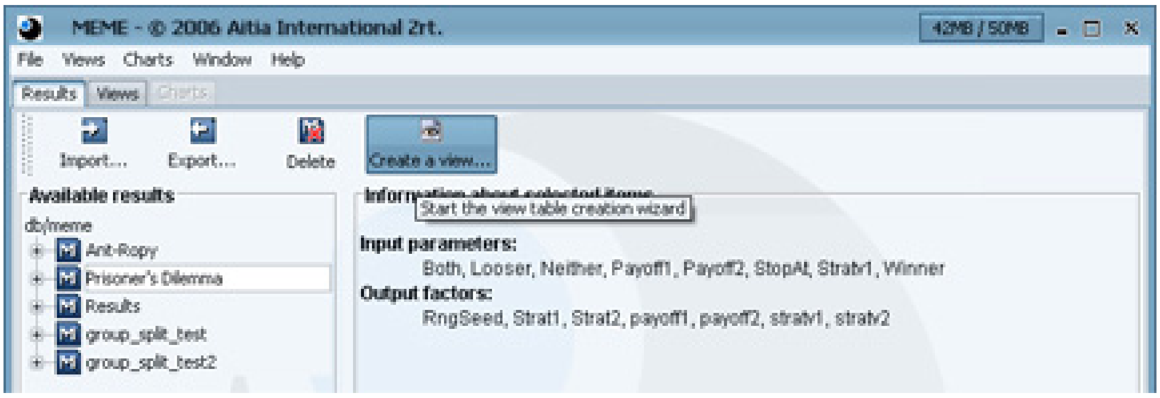
Figure 4 - Move to output

Press the *OK* button thus adding the Prisoner’s Dilemma to the MEME database.

3 Creating a view table

Before displaying a chart a view table must be created from the imported data. When creating a view table the imported data is processed and sorted according to the needs of the modeler.

Select the *Prisoner’s Dilemma* model on the *Results* tab in MEME.

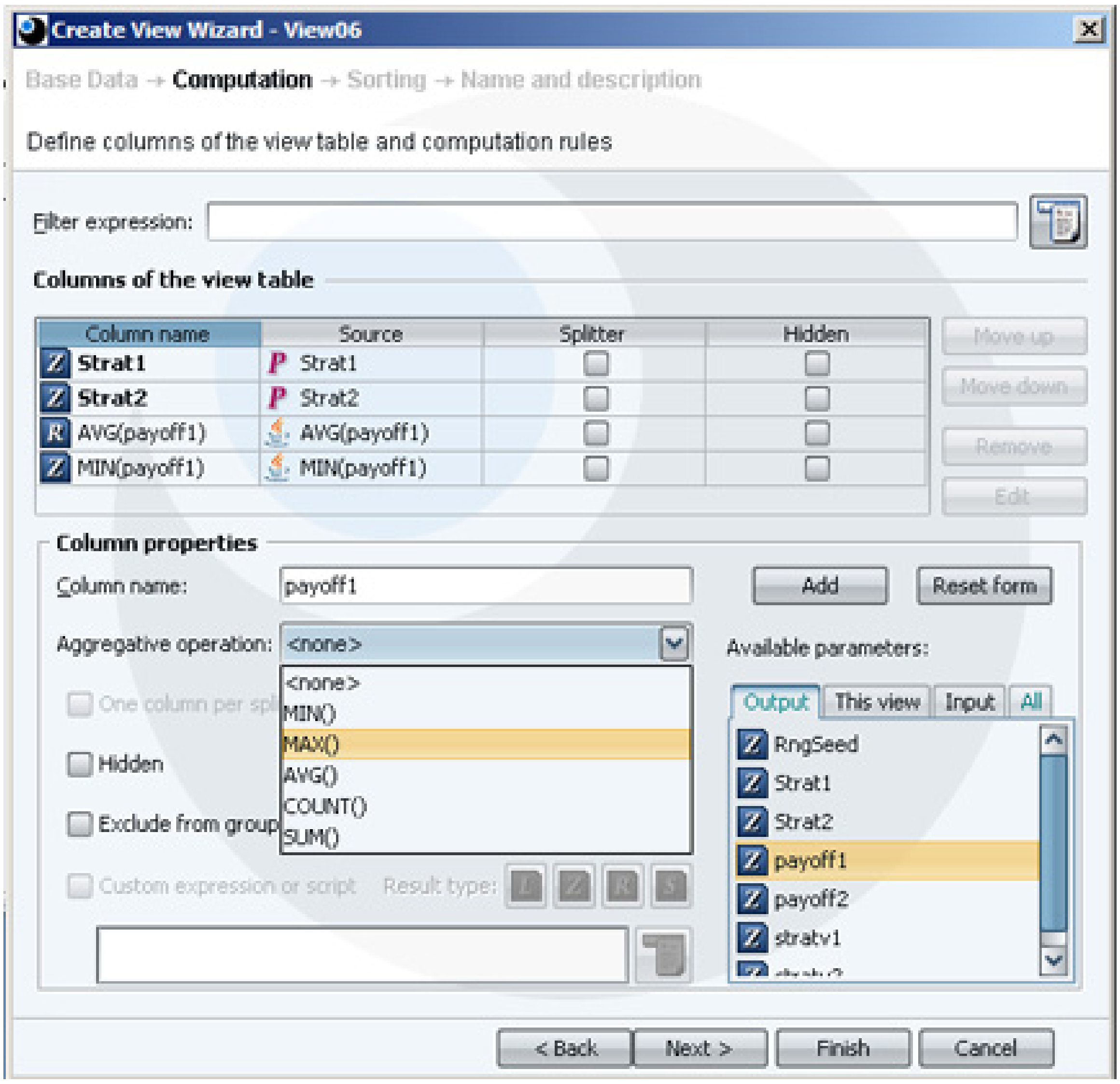


Input parameters:
Both, Looser, Neither, Payoff1, Payoff2, StopAt, Stratv1, Winner
Output factors:
RngSeed, Strat1, Strat2, payoff1, payoff2, stratv1, stratv2

Figure 5 - Select batch

Press Create a view... button on the toolbar or Views > Create a view… in the menu to invoke the Create View Wizard.

3.1 Computations


Create View Wizard - View06

Base Data → **Computation** → Sorting → Name and description

Define columns of the view table and computation rules

Filter expression:

Columns of the view table

Column name	Source	Splitter	Hidden
Strat1	Strat1	☐	☐
Strat2	Strat2	☐	☐
AVG(payoff1)	AVG(payoff1)	☐	☐
MIN(payoff1)	MIN(payoff1)	☐	☐

Column properties

Column name: payoff1

Aggregative operation: <none>

Figure 6 - Create view wizard

On the *Computation* page (see above) you can set the columns of the view table. Note that the parameters selected as output when importing are listed on the *Output* tab of the *Available parameters* list.

Add Strat1 and Strat2 by simply selecting them (two at a time if you wish) from the available parameters list and pressing the *Add* button.

3.1.1 Aggregative operations

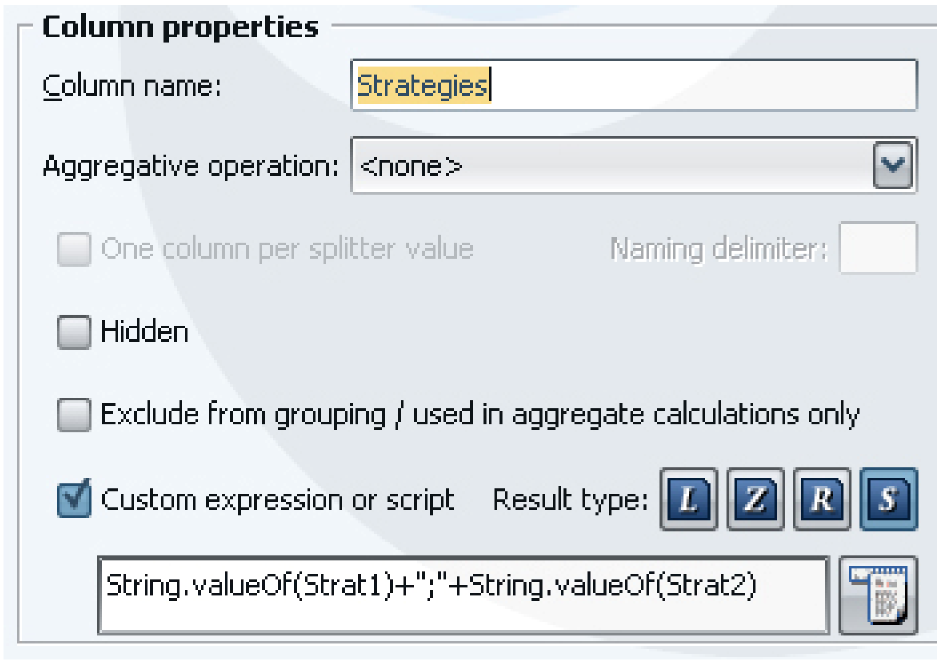
Select payoff1 and set *AVG()* in the *Aggregative operations* field and press *Add*. Due to the use of randomization in strategy 0 the model was executed ten times for every (strat1, strat2) pair thus the use of aggregative operations of the payoff values is advisable in order to analyze the data. Do the same with *MIN()* and *MAX()* operations for the payoff1 parameter and repeat the whole thing with the payoff2 values.

The program provides a default name for every column derived from the parameter it was created from. In our case it is useful to set a custom name for the payoff columns hence avoiding having columns named payoff1, payoff2, …, payoff6. In the example above the column names are made out of the name of the operation and the parameter it was done on.

3.1.2 Scripting

Now select *Custom expression or script* from the *Column properties*, press S to declare the new column values as string and type in:

`String.valueOf(Strat1)+”;“String.valueOf(Strat2)`



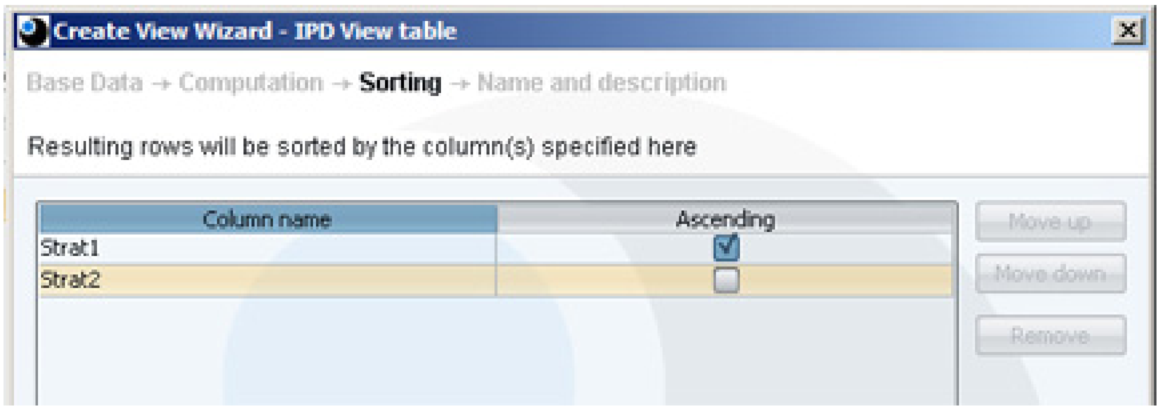
Column name: Strategies
Aggregative operation: <none>
[✓] Custom expression or script
String.valueOf(Strat1)+";"+String.valueOf(Strat2)

Figure 7 - Scripting

This little script creates the Strat1;Strat2 values we will need later on when creating charts. Provide a name (Strategies) for the column and press *Add*.

If you wish to modify an already added column select it and use the *Edit* button or simply double click the column. Don’t forget to press *Modify* when done with the changes. When done press *Next*.

3.2 Sorting


Create View Wizard - IPD View table
Base Data → Computation → Sorting → Name and description
Move up
Move down
Remove

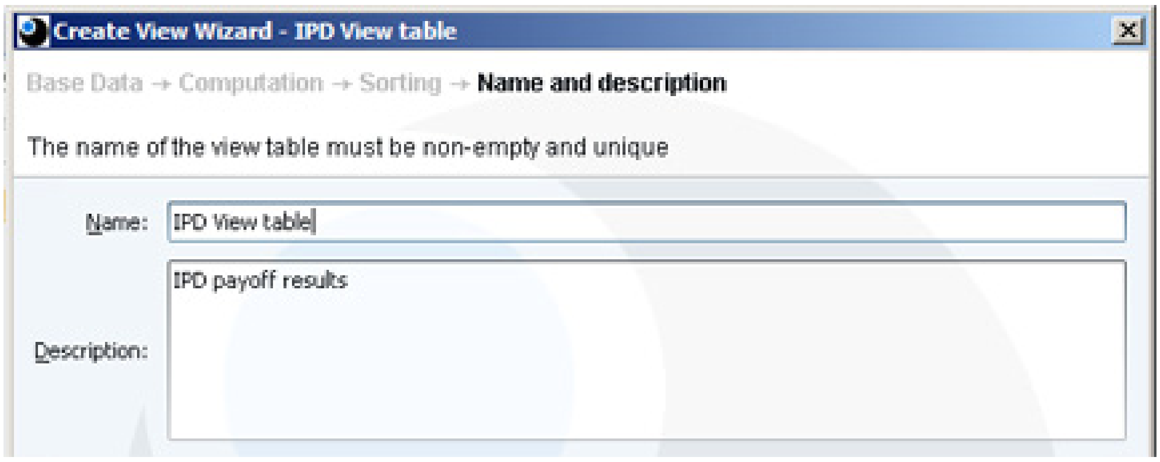
Resulting rows will be sorted by the column(s) specified here

Column name	Ascending
Strat1	✓
Strat2	☐

Figure 8 - Sorting

In order to create transparent view table sort the rows of the table by the columns Strat1 and 2, both in *Ascending* order. When done press *Next* to continue.

3.3 View name and description



Create View Wizard - IPD View table

Base Data → Computation → Sorting → **Name and description**

The name of the view table must be non-empty and unique

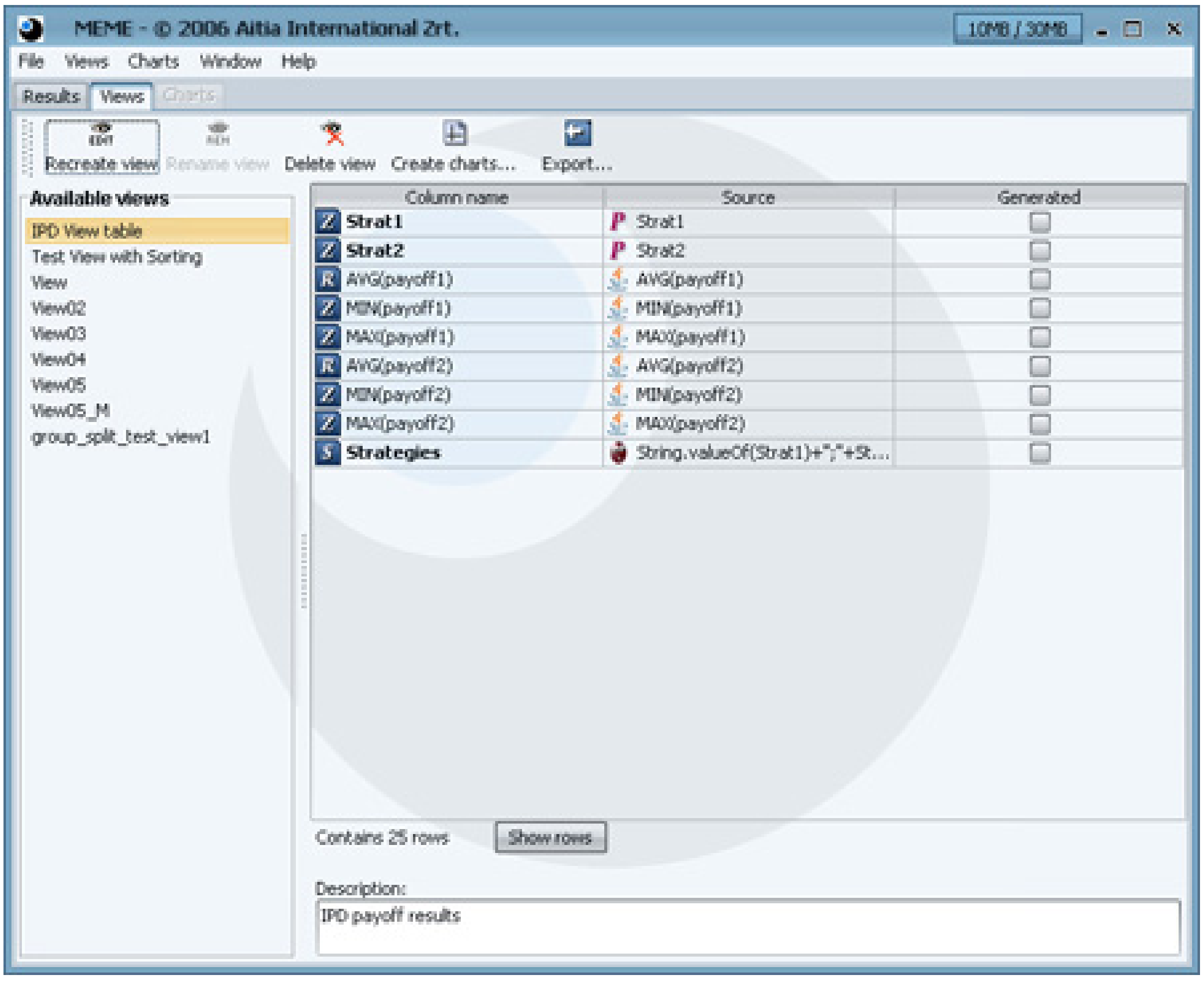
Name: IPD View table

Description: IPD payoff results

Figure 9 - Name and description

Give a unique name for the view – if there is already a view with the same name it will be overwritten – not longer than 64 characters and provide a description without limitations of length if you feel necessary. Press the Finish button when done thus adding the view table to the database. Now you can create charts in MEME or export the view table (File > Export…) as a CSV file.

4 Visualization



Available views

- IPD View table
- Test View with Sorting
- View
- View02
- View03
- View04
- View05
- View05_M
- group_split_test_view1

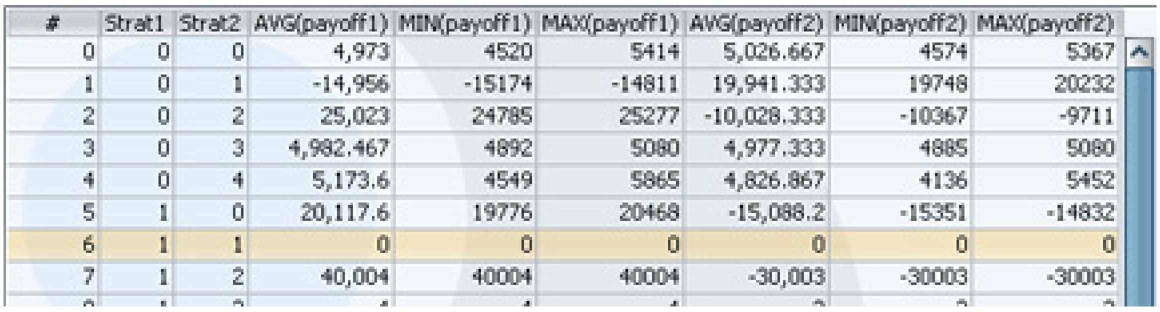
Column name	Source	Generated
Strat1	Strat1	☐
Strat2	Strat2	☐
AVG(payoff1)	AVG(payoff1)	☐
MIN(payoff1)	MIN(payoff1)	☐
MAX(payoff1)	MAX(payoff1)	☐
AVG(payoff2)	AVG(payoff2)	☐
MIN(payoff2)	MIN(payoff2)	☐
MAX(payoff2)	MAX(payoff2)	☐
Strategies	String.valueOf(Strat1)+";"+St…	☐

Contains 25 rows

Description: IPD payoff results

Figure 10 - Available views

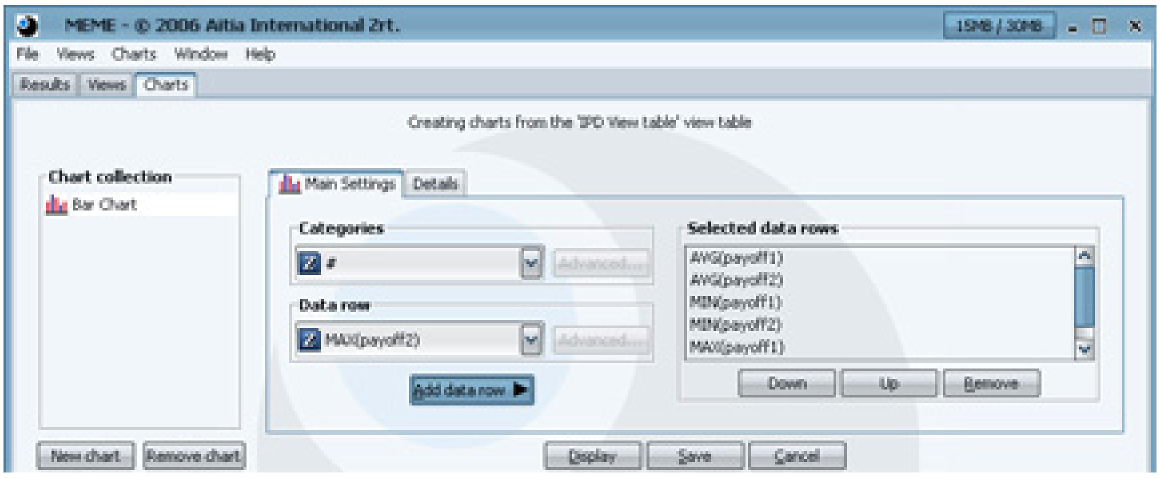
The freshly created view table is now listed under *Available views*. The column names and properties are listed by default but you can observe the actual values in the table by pressing *Show rows* on the bottom of the screen.



#	Strat1	Strat2	AVG(payoff1)	MIN(payoff1)	MAX(payoff1)	AVG(payoff2)	MIN(payoff2)	MAX(payoff2)
0	0	0	4,973	4520	5414	5,026.667	4574	5367
1	0	1	-14,956	-15174	-14811	19,941.333	19748	20232
2	0	2	25,023	24785	25277	-10,028.333	-10367	-9711
3	0	3	4,982.467	4892	5080	4,977.333	4885	5080
4	0	4	5,173.6	4549	5865	4,826.867	4136	5452
5	1	0	20,117.6	19776	20468	-15,088.2	-15351	-14832
6	1	1	0	0	0	0	0	0
7	1	2	40,004	40004	40004	-30,003	-30003	-30003

Figure 11 - Show rows

Having selected the *IPD View table* press Create charts... in the toolbar or select Charts > Create charts... from the menu to switch MEME to chart creating mode. Select *Bar Chart* from the *Available chart types* menu and press *Create*.



Creating charts from the 'IPD View table' view table

Chart collection: Bar Chart

Categories: #

Data row: MAX(payoff2)

Selected data rows:
AVG(payoff1)
AVG(payoff2)
MIN(payoff1)
MIN(payoff2)
MAX(payoff1)

Figure 12 - Creating a Bar Chart

4.1 Creating a bar chart

On the Main Settings tab set Strategies as the Categories value and add AVG(payoff1), AVG(payoff2), MIN(payoff1), ..., MAX(payoff2) as data rows. It is advisable to keep this order to be able to compare the bars visually. Provide a Title and a Subtitle on the Details tabs and press Display to see the chart.

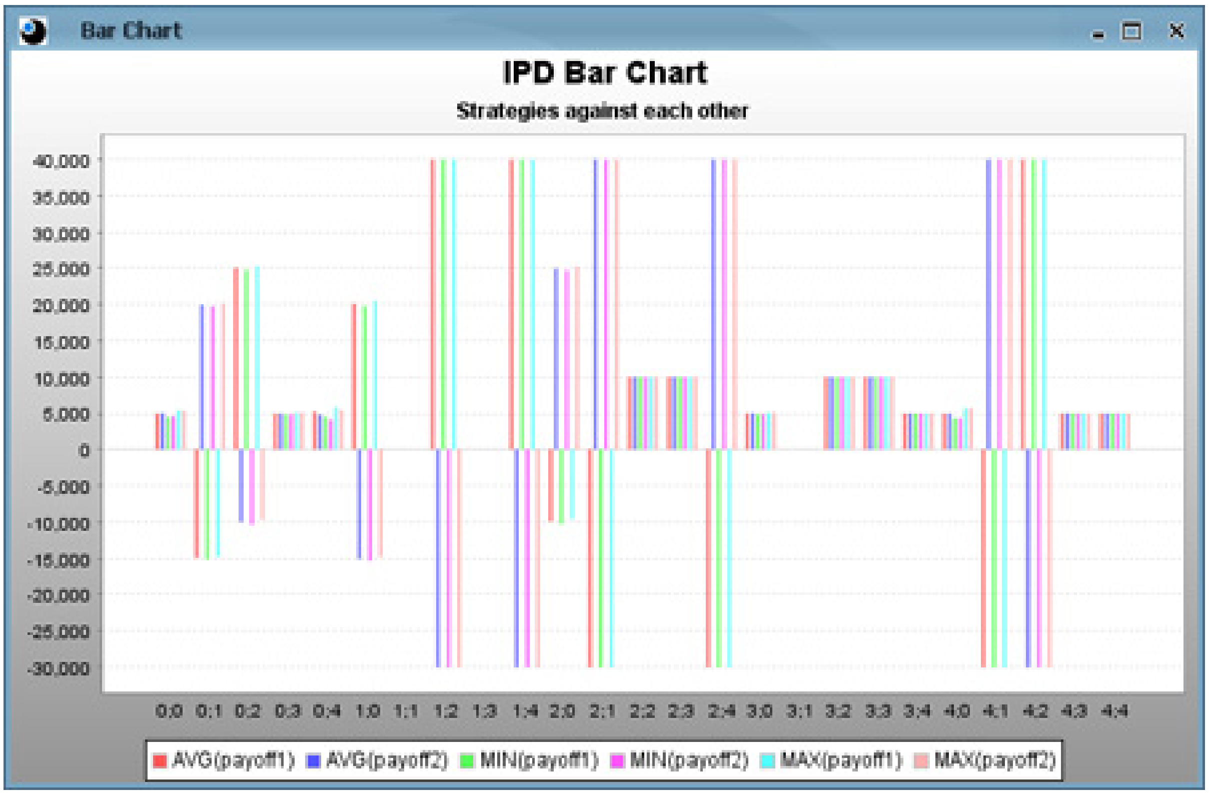


Figure 13 - Bar Chart

As you can see from the resulting bar chart, the average, minimum and maximum values do not differ considerably for the given payoff in one category it is possible to analyze the results picking only one data row for each player's payoff.

4.2 Creating a line chart

Press *New Chart* under the *Chart collection* field, choose XY Line Chart from the *Available chart types* list and press *Create*. In the *X values* field set Strategies*,* select AVG(payoff1) for Y values and press the *Add*▶ button. Repeat this for AVG(payoff2), optionally provide a title, a subtitle, and axis labels for the chart.

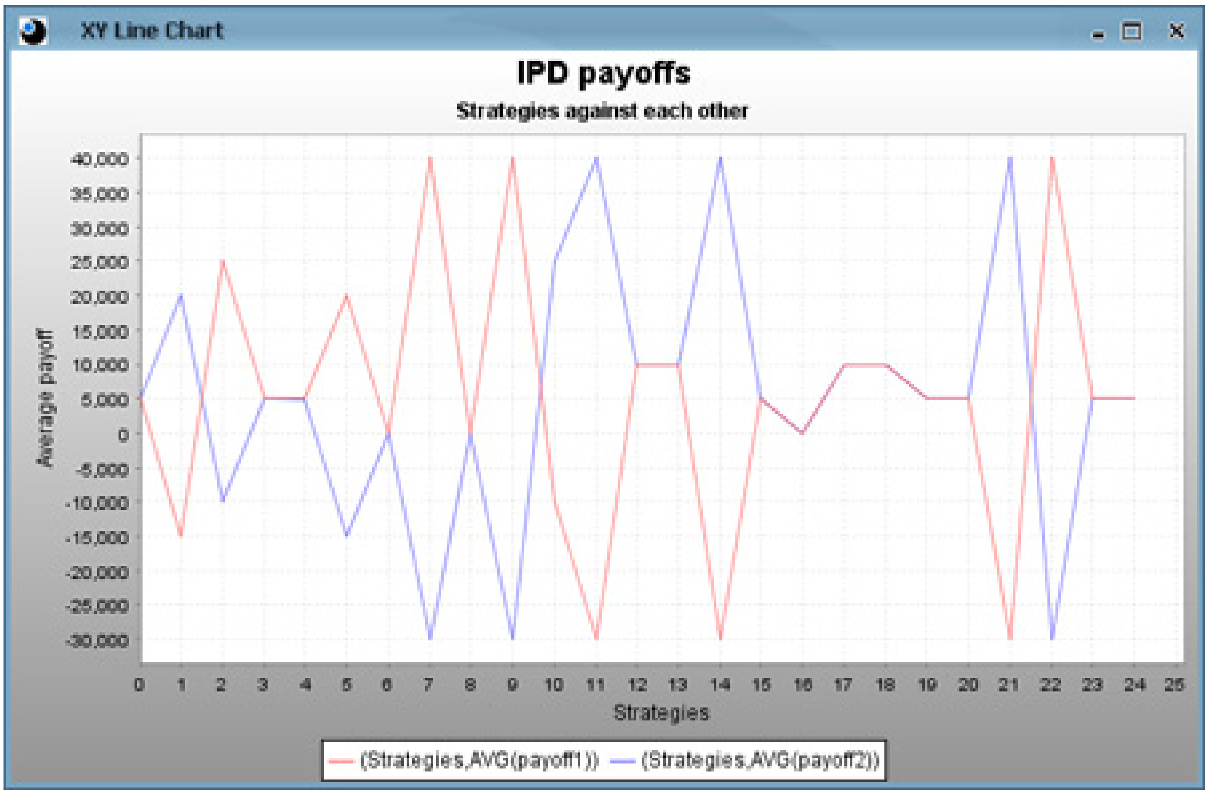


Figure 14 - XY Line Chart

4.3 Zooming

To observe the more interesting parts of the chart hold down the left mouse button and drag it right and down, selecting the area to be zoomed in on with a rectangle.

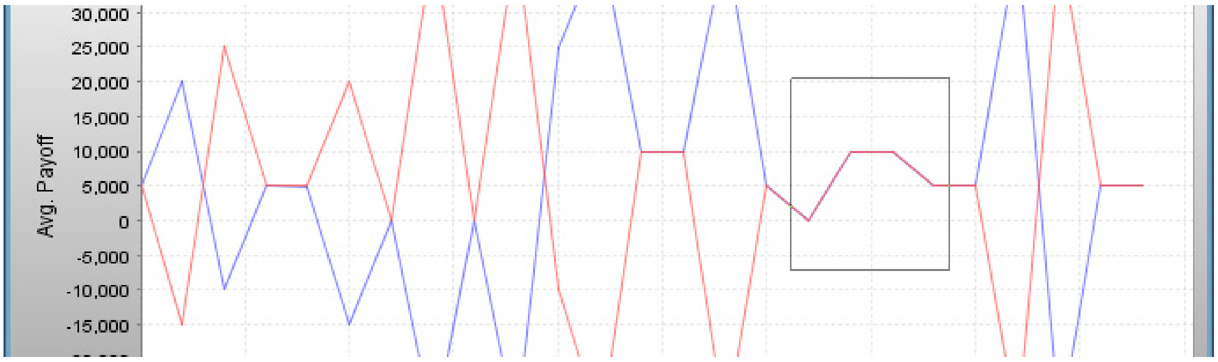


Figure 15 - Select area

After several repetitions it is revealed that strategy 4;0 has somewhat higher payoff2 than payoff1 averages in our results. To zoom out simply drag the mouse left and top while holding down the mouse button.

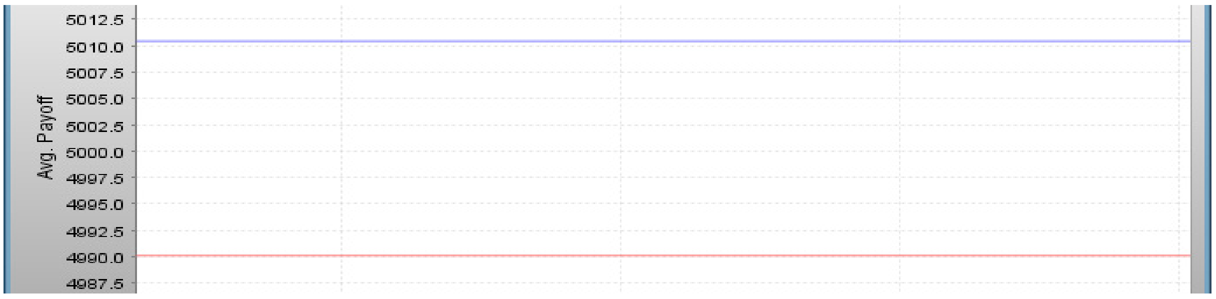
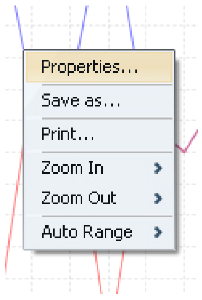


Figure 16 - Small difference

4.4 Exporting and formatting charts



By right clicking on the chart a context-menu pops up. You can control the zooming (Zoom In/Out) from here, export (Save as...) the chart as an EPS or a PNG file, print it (Print) and also format its appearance (Properties...). Through Properties titles and labels can be edited or changed, fonts, border and background colors can be set and axis properties can be customized.

Just make sure you get something nicer than we did in the end:

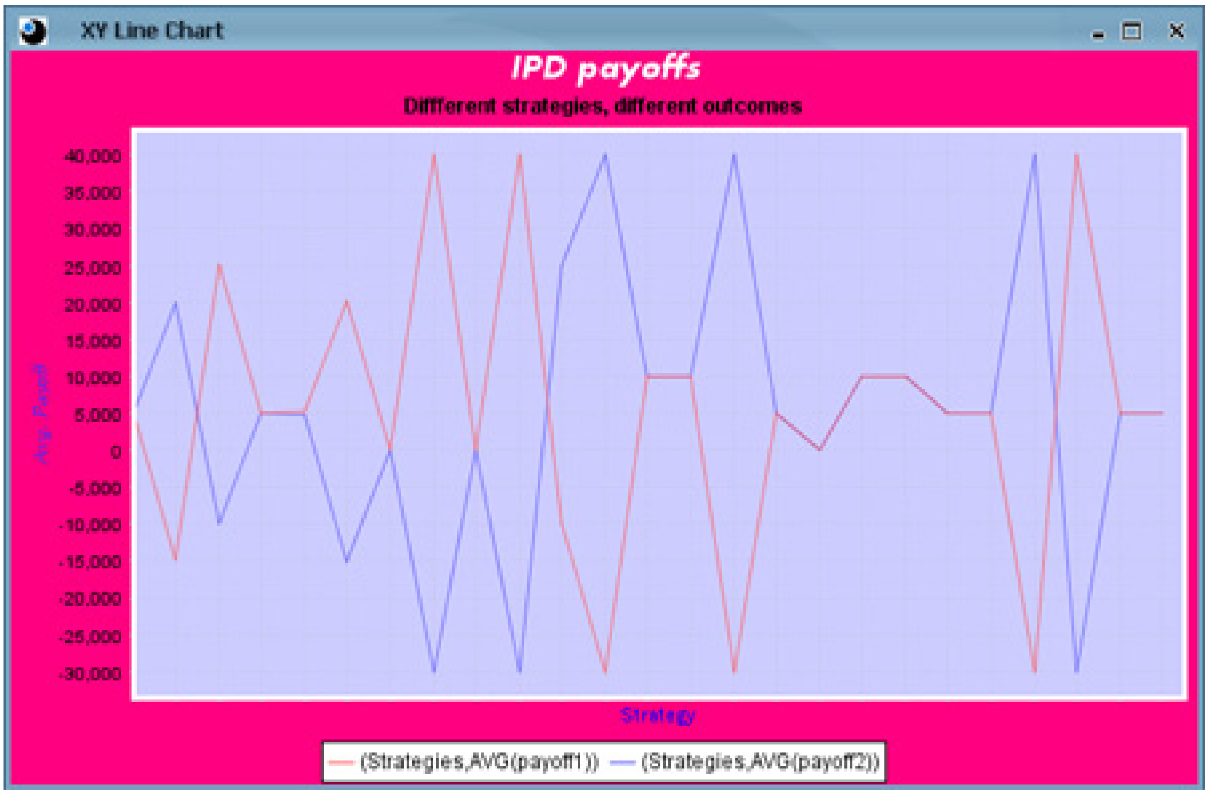


Figure 17 - This one might not win the design award

4.5 Saving the chart collection

When finished save the chart collection by pressing Save. MEME stores the data describing the chart in xml format at a user specified location. When opening (Charts > Open chart) the collection again make sure, that the view table it was created from is still in the database.

5 Notes

MEME is a tool for dealing with batch runs of simulations and the data produced. It allows the user to store and organize the raw data in database(s), and to create distillated (computed) tables which can be visualized on charts. In the near future MEME is expected to be able to run MASS/Repast simulations directly and collect their results internally, and on the other hand, to interoperate with other existing statistical software like R and Matlab. These developments will further simplify the modeling work and allow the users to focus on the models themselves.

6 References

[1] MASS – Multi-Agent Simulation Suite. © AITIA International Inc.
http://www.aitia.ai/services_and_products/simulation_systems/mass

[2] Repast - Recursive Porus Agent Simulation Toolkit
http://repast.sourceforge.net/

[3] Beanshell Scripting
http://www.beanshell.org/

[4] Iterated Prisoner's Dilemma
http://en.wikipedia.org/wiki/Iterated_Prisoners_Dilemma

7 Appendix

7.1 PrisonersAgent.java

```
package demo.prisoners;

import uchicago.src.sim.util.Random;

/** An Agent class representing a player with none or one-step memory in
Iterated Prisoner's Dilemma game. */
public class PrisonersAgent {

      /** Strategy: Random */
      public static final int RND = 0;
      /** Strategy: Allways defect */
      public static final int ALLD = 1;
      /** Strategy: Allways cooperate */
      public static final int ALLC = 2;
      /** Strategy: Tit-for-tat */
      public static final int TFT = 3;
      /** Strategy: Anti Tit-for-tat */
      public static final int ATFT = 4;

      /** The players current strategy. */
      protected int strategy;
      /** The last step of the other player. */
      protected boolean enemyLast;

      public PrisonersAgent(int strategy) {
            Random.createUniform();
            this.strategy=strategy;
            enemyLast=true;
      }

      /** Memorizes the last step of the other player. */
      public void setEnemyLast(boolean b) {
            enemyLast = b;
      }

      /** Returns true if cooperates. */
      public boolean cooperate() {
            switch (strategy) {
                  case TFT:   return enemyLast;
                  case ATFT:  return !enemyLast;
                  case ALLD:  return false;
                  case ALLC:  return true;
                  case RND:   return //Random.uniform.nextBoolean();
                  uchicago.src.sim.util.Random.uniform.nextBoolean();
            }
            return true;
      }
}
```

7.2 PrisonersModel.java

```
package demo.prisoners;

import uchicago.src.sim.engine.*;

/**Iterated Prisoner's Dilemma game model. */
public class PrisonersModel extends SimpleModel {

      public PrisonersModel() {
            super();
            name = "Prisoner's Dilemma";
      }

      /** The winner's payoff. */
      protected int winner;
      /** Returns winner's payoff. */
      public int getWinner() { return winner; }
      /** Sets winner's payoff. */
      public void setWinner(int winner) { this.winner=winner; }
```

```
/** The loser’s payoff. */
protected int looser;
/** Returns looser's payoff. */
public int getLooser() { return looser; }
/** Sets looser's payoff. */
public void setLooser(int looser) { this.looser=looser; }
/** Both players' payoffs if they're cooperate. */
protected int both;
/** Returns payoffs if both cooperate. */
public int getBoth() { return both; }
/** Sets payoff if both cooperate. */
public void setBoth(int both) { this.both=both; }
/** Both players' payoffs if neither cooperates. */
protected int neither;
/** Returns payoffs if neither cooperates. */
public int getNeither() { return neither; }
/** Sets payoffs if neither cooperates. */
public void setNeither(int neither) { this.neither=neither; }
/** The 1st player’s strategy. */
protected int strat1;
/** Sets 1st player’s strategy. */
public int getStrat1() { return strat1; }
/** Returns the 1st player’s strategy. */
public void setStrat1(int strat1) { this.strat1=strat1; }
/** The 2nd player’s strategy. */
protected int strat2;
/** Sets the 2nd player’s strategy. */
public int getStrat2() { return strat2; }
/** Returns the 2nd player’s strategy. */
public void setStrat2(int strat2) { this.strat2=strat2; }

/** 1st player's payoff.*/
protected int payoff1;
public void setPayoff1(int i) { payoff1 = i; }
public int getPayoff1() { return payoff1; }
/** 2nd player’s payoff. */
protected int payoff2;
public void setPayoff2(int i) { payoff2 = i; }
public int getPayoff2() { return payoff2; }

public String[] getInitParam() {
      String[] params = {"winner","looser","both","neither",
                  "strat1","strat2","payoff1","payoff2"};
    return params;
}

public void setup() {
      super.setup();
      generateNewSeed();
      payoff1 = 0;
      payoff2 = 0;
}

@SuppressWarnings("unchecked")
public void buildModel() {
      PrisonersAgent a = new PrisonersAgent(strat1);
      agentList.add(a);
      PrisonersAgent b = new PrisonersAgent(strat2);
      agentList.add(b);
}

public void step() {
      PrisonersAgent a = (PrisonersAgent)agentList.get(0);
      PrisonersAgent b = (PrisonersAgent)agentList.get(1);
      boolean cA = a.cooperate();
      boolean cB = b.cooperate();
      if (cA && cB) {
            payoff1+=both;
            payoff2+=both;
      }
      if (cA && !cB) {
            payoff1+=looser;
            payoff2+=winner;
      }
      if (!cA && cB) {
            payoff1+=winner;
            payoff2+=looser;
            }
            if (!cA && !cB) {
                  payoff1+=neither;
                  payoff2+=neither;
            }
            a.setEnemyLast(cB);
            b.setEnemyLast(cA);
      }

      public void atEnd() {
            super.atEnd();
      }

}
```

7.3 PrisonersModelBatch.java

```
package demo.prisoners;

import uchicago.src.sim.engine.BasicAction;
import uchicago.src.sim.analysis.*;

/** Iterated Prisoner's Dilemma game for batch mode execution.*/
public class PrisonersModelBatch extends PrisonersModel {

      /** The model stops if the current step reaches this count.*/
      protected int stopAt;
      protected int stratv1;
      protected int stratv2;

      public int    getStopAt()           { return stopAt; }
      public int    getStratv1()          { return stratv1; }
      public int    getStratv2()          { return stratv2; }
      public void   setStopAt(int stopAt) { this.stopAt=stopAt; }
      public void   setStratv1(int i)     { stratv1=i; }
      public void   setStratv2(int i)     { stratv2=i; }

      protected DataRecorder recorder;

      public String[] getInitParam() {
            String[] sparams = super.getInitParam();
            int l = sparams.length;
            String[] params = new String[l+3];
            int i;
            for (i=0; i<l; ++i) {
                  params[i]=sparams[i];
            }
            params[i  ]="stopAt";
            params[i+1]="stratv1";
            params[i+2]="stratv1";
          return params;
      }

      public void setup() {
            stopAt = 1000;
            super.setup();
      }

      public void buildModel() {
            super.buildModel();
            buildRecorder();
            this.run();
      }

      protected void buildRecorder() {
            class NumDataSource implements NumericDataSource {
                  private int parIdx;
                  PrisonersModel model;
                  NumDataSource(int parIdx) {
                        this.parIdx = parIdx;
                        this.model = model;
                  }
                  public double execute() {
                        double ans = 0;
                        switch (parIdx) {
                              case 0 : ans = getPayoff1(); break;
                              case 1 : ans = getPayoff2(); break;
                              case 2 : ans = getStratv1(); break;
                              case 3 : ans = getStratv2(); break;
                        }
                        return ans;
                  }
            }
            recorder = new DataRecorder(this.getName()+".out",this);
            recorder.setDelimeter(" -- ");
            recorder.addNumericDataSource("payoff1", new NumDataSource(0));
            recorder.addNumericDataSource("payoff2", new NumDataSource(1));
            recorder.addNumericDataSource("stratv1", new NumDataSource(2));
            recorder.addNumericDataSource("stratv2", new NumDataSource(3));
      }

      public void buildSchedule() {
            super.buildSchedule();
            class StopAction extends BasicAction {
                  public void execute() {
                        stop();
                  }
            }
            schedule.scheduleActionAt(stopAt, new StopAction());
      }

      public void atEnd() {
            super.atEnd();
            stratv1 = strat1 * stopAt / 2;
            stratv2 = strat2 * stopAt / 2;
            recorder.record();
            recorder.writeToFile();
      }

}
```

7.4 PrisonersModelGUI.java

```
package demo.prisoners;

import uchicago.src.sim.analysis.*;

/** Iterated Prisoner's Dilemma game for gui mode*/
public class PrisonersModelGUI extends PrisonersModel {

      /** Graph in the GUI mode. */
      protected OpenSequenceGraph graph;

      class Payoff implements Sequence {
            private int player;
            public Payoff(int player) { this.player = player; }
            public double getSValue() {
                  return (double)(player==1?payoff1:payoff2);
            }
      }

      public void setup() {
            super.setup();
            winner=4;
            looser=-3;
            both=1;
            neither=0;
            strat1 = PrisonersAgent.RND;
            strat2 = PrisonersAgent.RND;
      }

      public void buildModel() {
            super.buildModel();
            if (graph!=null) graph.dispose();
            graph=new OpenSequenceGraph("Payoff",this);
            graph.setXRange(0, 50);
            graph.setYRange(-30, 170);
            graph.setXViewPolicy(OpenSequenceGraph.SHOW_LAST);
            graph.addSequence("1st player", new Payoff(1));
            graph.addSequence("2nd player", new Payoff(2));
            graph.display();
      }
```

```
	public void step() {
		super.step();
		graph.step();
	}
}
```