Acknowledgment: Some of the slides below are taken from lecture slides by Prof. Axtell and Prof. Tesfatsion.

How?

Giorgio Fagiolo

Sant'Anna School of Advanced Studies, Pisa (Italy)

giorgio.fagiolo@sssup.it

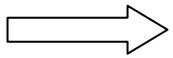
https://mail.sssup.it/~fagiolo

The Structure of Agent-Based Models

- Main ingredients (to cook an ABM)
  - Bottom-up (agent-based) Philosophy (Tesfatsion, 1997)
  - Agents live in complex systems evolving through time (Kirman, 1998)
  - Agents might be heterogeneous in almost all their characteristics
  - “Hyper-rationality” not viable (Dosi et al., 1996)
  - Agents as boundedly rational entities with adaptive expectations
  - “True” dynamics: Systems are typically non-reversible
  - Agents interact directly, networks change over time (Fagiolo, 1997)
  - Endogenous and persistent novelty: open-ended spaces
  - Selection-based market mechanisms (Nelson & Winter, 1982)

The Outcomes of ACE/EV Models (1/2)

Micro-Dynamics
(induced by decision rules, interactions and expectations)



Macro-Dynamics
(obtained as aggregation of individual behaviors)

- Stochastic components in decision rules, expectations, interactions imply that the dynamics of micro and macro variables can be described by some (Markovian) stochastic process parameterized by $(\underline{\theta}_i)$, Θ :

$$(\underline{x}_{i,t}) \mid (\underline{x}_{i,t-1}), (\underline{x}_{i,t-2}), \ldots ; (\underline{\theta}_i), \Theta$$

$$\underline{X}_t \mid (\underline{X}_{t-1}, \underline{X}_{t-2}, \ldots ; (\underline{\theta}_i), \Theta)$$

- Non-linearities in decision rules, expectations, interactions **may** imply that it is **hard** to analytically derive laws of motion, kernel distributions, time-*t* probability distributions, etc.

An important distinction

- **Analytically-solvable ABMs**
  - Very simple, toy models
  - Little microeconomics
- **ABM delivering NO analytical solution in their full-fledged version**
  - More micro-founded
  - Over parameterized?
  - Model selection: KISS vs. KIDS vs. TAPAS
- **In what follows: some examples of analytically-solvable ABMs**
  - Simple dynamic coordination games
  - Boundedly-rational players
  - Global (population) interactions

➢ Kirman (1993): Ants, rationality and recruitment

- Agents: $i = 1, \ldots, N$
- Time: $t = 0, 1, 2, \ldots$
- Choices: $A = \{-1, +1\}$
  - State of the system: $k_t \in \{0, 1, 2, \ldots, N\}$ (# agents choosing +1)

- Individual Dynamics:
  - Suppose there are $k_t = k$ (resp. $N-k$) agents choosing +1 (resp. -1) at t
  - An agent is drawn at random and makes a "phone call" to another randomly drawn agent in the population
  - The agent is converted to other's choice with probability $1-\delta$ (NB: $1-\delta$ measures the strength of interactions)
  - There is a small probability $\varepsilon > 0$ that the agent switches **without doing** the "phone call" (NB: ε measures the strength of idiosyncrasy)

- Aggregate Dynamics:
  - Given $k_t = k$ then we will only have $k_{t+1} \in \{k-1, k, k+1\}$.
  - K_t is a Markov Chain
  - Transition probability matrix $P(k_{t+1} | k_t)$ has entries different from zero only if $k_{t+1} \in \{ k_t - 1, k_t, k_t + 1\}$
  - Must compute only:
    - $Prob\{k+1 | k\}$, $Prob\{k-1 | k\}$
    - $Prob\{k | k\} = 1 - Prob\{k+1 | k\} - Prob\{k-1 | k\})$

$$p_{k+1,k} = Prob\{k+1 | k\} = \text{Prob\{A } (-1) \text{ player is drawn}\} \cdot$$
$$\text{Prob\{The player changes his mind\}} =$$

$$= \text{Prob\{A } (-1) \text{ player is drawn}\} \cdot$$
$$[\ \text{Prob\{Changes independently of the phone call}\} +$$
$$\text{Prob\{Meets a } (+1) \text{ player}\} \cdot \text{Prob\{Changes his idea}\}\ ] =$$

$$= \left(1 - \frac{k}{N}\right) \cdot \left(\varepsilon + \frac{k}{N-1}(1-\delta)\right)$$

$p_{k-1,k}$ = *Prob{k–1| k}* = Prob{A (+1) player is drawn}·
Prob{The player changes his mind}=

= Prob{A (+1) player is drawn}·
[Prob{Changes independently of the phone call} +
Prob{Meets a (–1) player} · Prob{Changes his idea}] =

$$= \frac{k}{N} \cdot \left(\varepsilon + \frac{N-k}{N-1}(1-\delta) \right)$$

Results

- K_t is a Markov chain
- The transition probability matrix reads:

k	0	1	2	…	k-1	k	k+1	…	N-2	N-1	N
0	$1-\varepsilon$	ε	0	…	0	0	0	…	0	0	0
…	…	…	…	…	…	…	…	…	…	…	…
k	0	0	0	…	$p_{k-1,k}$	$p_{k,k}$	$p_{k+1,k}$	…	0	0	0
…	…	…	…	…	…	…	…	…	…	…	…
N	0	0	0	…	0	0	0	…	0	ε	$1-\varepsilon$

- The chain is aperiodic and irreducible. Therefore there exists a unique invariant (limit) distribution μ that satisfies:

$$\mu_k \cdot p_{k+1,k} = \mu_{k+1} \cdot p_{k,k+1}$$

$$\frac{\mu_{k+1}}{\mu_k} = \frac{p_{k+1,k}}{p_{k,k+1}} = \frac{(1-\frac{k}{N})(\varepsilon+(1-\delta)\frac{k}{N-1})}{\frac{k+1}{N}(\varepsilon+(1-\delta)\frac{N-k-1}{N-1})}$$

- Shape of invariant distribution:

  - Depends on $\theta = \varepsilon / (1-\delta)$

  - Meaning of θ
    - Large θ: Weak interactions, strong idiosyncrasy
    - Small θ: Strong interactions, weak idiosyncrasy

  - If $\theta = \theta^* = (N-1)^{-1} \Rightarrow \mu_k/\mu_{k+1}=1 \Rightarrow \mu$ is UNIFORM

  - If $\theta>\theta^* \Rightarrow \mu$ is $\cap$-shaped (unimodal) around N/2

  - If $\theta<\theta^* \Rightarrow \mu$ is $\cup$-shaped (bimodal) w/ peaks {0,N}

Arthur *et al.* (1994): Polya-Urn Schemes

- Agents: $i = 1, 2, \ldots$ (infinitely enumerable)
- Time: $t = 0, 1, 2, \ldots$
- Choices: $A = \{-1, +1\}$
- State of the system: $x_t \in [0,1]$ (share of agents choosing +1)

  or: $y_t = 0,1,2,\ldots$ (# of firms choosing +1)

- Individual Dynamics:
  - Suppose at time $t=0$ there are N_0 firms of which a share x_0 chooses +1 (i.e. $y_0 = N_0 x_0$ firms in total)
  - Let $p(x)$ the probability that a firm chooses +1 given existing adoption shares; $p(x)$ increasing in x due to positive network externalities
  - At any t=1,2,… a new firm enters the industry and <u>irreversibly</u> chooses +1 with probability $p(x_t)$ [and of course -1 with probability $1-p(x_t)$]
  - NB: At time t there will be N_0+t firms in the industry

- Aggregate Dynamics:

  - Given $y_0 = N_0 x_0$, then:

$$y_{t+1} = y_t + \begin{cases} 1 \text{ with } p = p(x_t) \\ \\ 0 \text{ with } p = 1 - p(x_t) \end{cases}$$

  - Therefore:

$$y_{t+1} = y_t + z(x_t)$$

where $z(x_t) = \begin{cases} 1 & \text{with prob} = p(x_t) \\ 0 & \text{with prob} = 1 - p(x_t) \end{cases}$

NB: $\mathbf{E}[z(x_t)] = 1 \cdot p(x_t) + 0 \cdot (1 - p(x_t)) = p(x_t)$

$(N_0+t+1)x_{t+1} = (N_0+t)x_t + z(x_t)$

$$x_{t+1} = x_t + \frac{1}{N_0 + t + 1}\left(z(x_t) - x_t\right)$$

$$\boxed{x_{t+1} = x_t + \frac{1}{N_0 + t + 1}\left(p(x_t) - x_t\right) + \frac{1}{N_0 + t + 1}\varepsilon(x_t)}$$

where: $\varepsilon(x_t) = z(x_t) - p(x_t) = z(x_t) - E[z(x_t)]$

$\Rightarrow$ $\boldsymbol{E}[\varepsilon(x_t)]=0$

NB: The equivalent deterministic system reads:

$$\mathbf{E}(x_{t+1}) = x_t + \frac{1}{N_0 + t + 1}\left(p(x_t) - x_t\right)$$

Results

- Theorem (Arthur, 1988)

Define V=$\{x \in [0,1]: p(x)=x\}$

- If $p(x_t)$ is a CONTINUOUS function from [0,1] to [0,1] and:
- The equivalent deterministic system has a Lyapunov function (i.e. it is globally stable);

Then: x_t converges with probability one to a STABLE point in V, i.e. a point $x^* \in V$ s.t. for any small $\varepsilon>0$:

- *If* $x^* \in (0,1) \Rightarrow p(x^*+\varepsilon) \leq x^*+\varepsilon$

  $p(x^*-\varepsilon) \geq x^*-\varepsilon$
- *If* $x^*=0 \Rightarrow p(\varepsilon) \leq \varepsilon$
- *If* $x^*=1 \Rightarrow p(1-\varepsilon) \geq 1-\varepsilon$

- **Key Points**

  - The system always LOCKS-IN into a stable frequency because noise introduced by entrants later on becomes negligible.

  - The process is PATH-DEPENDENT: Initial conditions and history matter !

  - Necessity (structural conditions embodied in *p*) determines possible long-run behavior

  - Chance (small early entrants) determines actual long-run behavior

  - Dynamic interplay between chance and necessity implies non predictability and possible inefficiency (QWERTY).

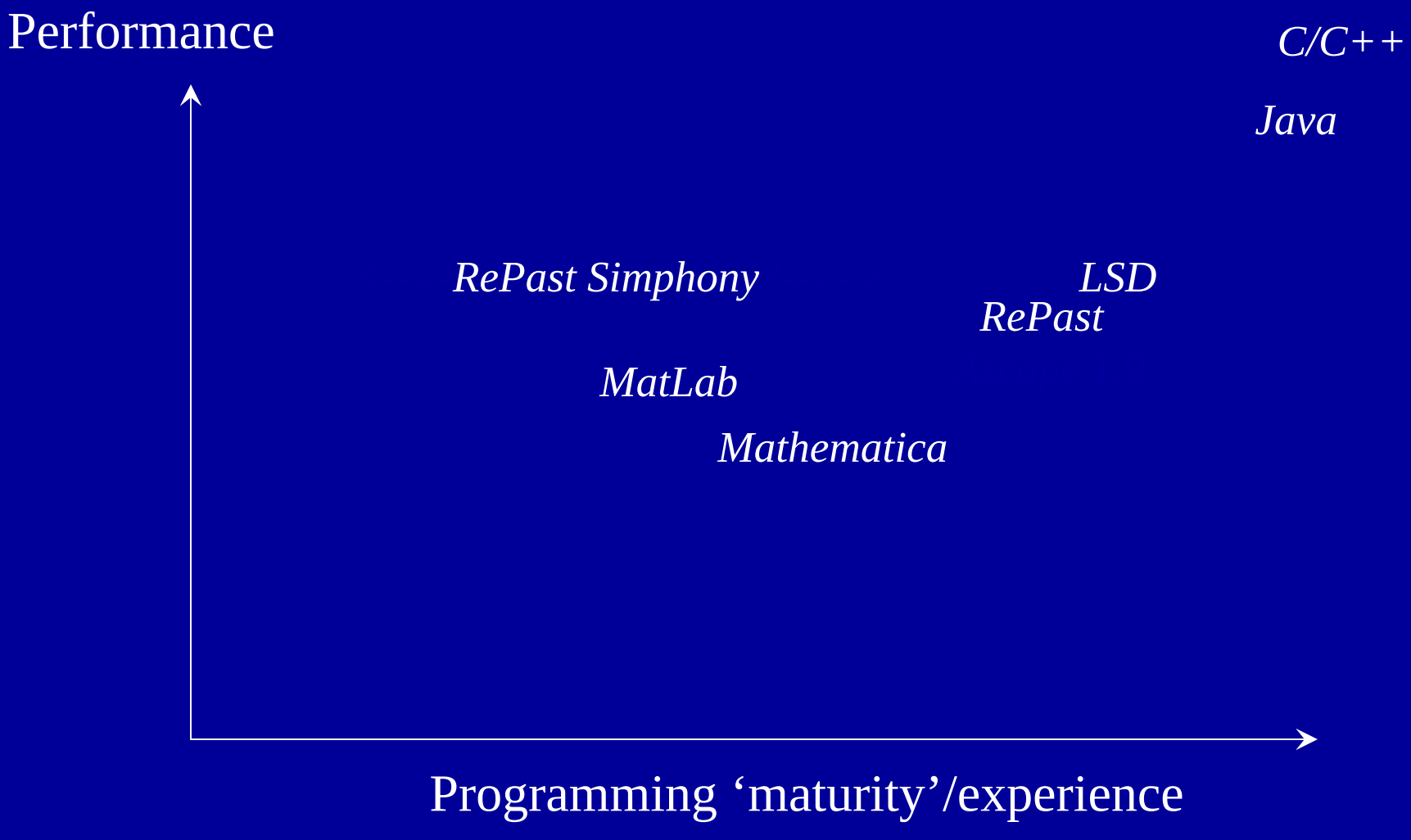
What if the model is not analytically solvable?

- **Different levels of lack of analytical solutions**
  - The model is analytically solvable but the modeler does not figure it out
  - The model allows for numerical solutions (equilibrium)
  - The model is inherently non-solvable
- **Simulating the behavior of the model**
  - Build an algorithmic description of micro and macro dynamics
  - Run the model (for given seed, parameters, initial conditions)
  - Store model's outputs
  - Understand what happens and why
- **It seems plain vanilla but it is not! Critical phases**
  - Writing the code (implementation, bugs, etc.)
  - Analyzing the model (graphical and statistical tools)
  - Understanding the outcomes (dependence on seeds, initial conditions, parameters)

Writing the code

- **Some important decisions**
  - Choosing a language vs. choosing a platform
  - Choosing the programming approach (standard vs. OOP)
  - Choosing how to analyze the output (built-in vs. external code/software)
- **Language**
  - Many available: C/C++, Java, Python
  - Compiled: faster
  - Flexible, powerful, control over code, self-development, availability of functions and classes, can handle any ABMs
  - Huge initial investment, poor results in the short-run, needs more work
- **Platform**
  - Many available: Repast, Swarm, CAS, **LSD**, Matlab, etc.
  - Interpreted: slower
  - Less flexible (its own language), less control, cannot handle any ABMs
  - Small initial investment, good results in the short-run, needs less work

Summary


Performance
C/C++
Java
RePast Simphony
LSD
RePast
MatLab
Mathematica
Programming ‘maturity’/experience

Implementing a 2-dim binary Cellular Automata

- S=2 (# of states)
- SP={0,1}
- K = # of neighbors
- # of all possible local configurations = 2^(K+1)
- # of all possible rules: 2^(2^(K+1))
- How to code a rule? A number from 0 to [2^(2^(K+1))-1]. Why?

Example: S=2, K=2: 8 local configurations, 256 rules

000	0
001	1
010	0
011	1
100	1
101	0
110	1
111	0

Bin-to-dec representation of local configurations

000	0	=0*2^2+0*2^1+0*2^0
001	1	=0*2^2+0*2^1+1*2^0
010	2	=0*2^2+1*2^1+0*2^0
011	3	...
100	4	...
101	5	...
110	6	...
111	7	=1*2^2+1*2^1+1*2^0

- We can order local configurations in a unique way using their decimal representation
- Each local configuration can be associated to a number from 0 to 2^(K+1)-1

Bin-to-dec representation of rules

- Given decimally-ordered 2^(K+1) configurations, a rule is a 2^(K+1)-digit binary number
- EX (K=2): A rule is a 8-digit binary number
- A 2^(K+1)-digit binary number can be univocally associated to a decimal number from 0 to [2^(2^(K+1))-1]
- EX (K=2): A rule can be associated to a decimal number from 0 to 255
- EX: Rule 90

000	0	0	0*2^0=0
001	1	1	1*2^1=2
010	2	0	0*2^2=0
011	3	1	1*2^3=8
100	4	1	1*2^4=16
101	5	0	0*2^5=0
110	6	1	1*2^6=64
111	7	0	0*2^7=0
			sum=90

Therefore:

0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

Fully characterized by →

0
1
0
1
1
0
1
0

1-to-1 with ↔ Rule number

Given any r in {0,...,255}: Apply DEC2BIN(r,8) to get a 8-digit string **a**={a0 a1 a2 ... a7} (binary equivalent of the rule number)

Given any 3-digit binary local configuration **b**={b0 b1 b2} apply BIN2DEC(**b**) to get the decimal equivalent of the local configuration, i.e. **the entry in the look-up table of the rule (on a 0 to 7 scale)**

EX1: DEC2BIN(90,8)={01011010}
EX2: BIN2DEC(010)=2
EX3: How to find the output associated to {010}? Look in the 3rd row (2+1, REM: 0-7 scale) in the look up table (i.e. in the 3rd position of the output string {01011010})

1dim CAs: Structure of the code

1. Set number of cells (N), number of states (K), rule (dec) number, number of neighbors, number of iterations (T)
2. Define structure of interactions (ring)
3. Convert rule using dec2bin
4. Set initial conditions (possibly using different options)

5. Begin cycle for t=1:T
   a. Copy current system state into temp configuration
   b. Begin cycle for i=1:N
      - Read neighborhood state of i
      - Convert neighborhood state into decimal representation and employ it to get new output
      - Update new output into temp system state
   c. Copy temp system state into new current system state

Platform Example: 1-DIM Cellular Automata

- Code to simulate a 1-DIM cellular automata in Matlab (1/3)

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% CA in 1-dimensional space with periodic boundaries  %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all;

% Number of cells
N=100;

% Number of iterations and output matrix
MAXT=100;
PLOTMAT=zeros(MAXT+1,N);

% Number of states
S=2;

% Number of neighbors (excluding agent)
% NB: Must be even
K=2;

% Defining Radius of Interaction
INTRAD=K/2;

% State space
SP=0:S-1;

% Define rule
RNUM=33;
```

Platform Example: 1-DIM Cellular Automata

- **Code to simulate a 1-DIM cellular automata in Matlab (2/3)**

```
% Convert rule into output vector
% NB: Second parameter = number of bits required to define a lookup table
% It is equal to the number of possible configurations
NCONF=S^(K+1);
TEMPOUT=str2num(transpose(dec2bin(RNUM,NCONF)));
OUT=TEMPOUT(NCONF:-1:1);                                    % Neded to correctly match I/O


% Define neighboring structure
NEIGH=zeros(N,K+1);             % Rows: Cells; Columns: Labels of K+1 Neighbors
LABELS=[[1:N] [1:N] [1:N]];     % Needed in order to allow for boundary conditions

for i=1:N
    h=1;
    for j=-INTRAD:INTRAD
            NEIGH(i,h)=LABELS(N+j+i);
            h=h+1;
    end
end

% Define initial configuration

% Random
%INICONF=unidrnd(2,1,N)-1;

% ONLY MID CELL ON
INICONF=zeros(1,N);
INICONF(N/2)=1;
```

Platform Example: 1-DIM Cellular Automata

- **Code to simulate a 1-DIM cellular automata in Matlab (3/3)**

```
% Set iteration
time=1;

% Set current configuration (the one agents look to adjust)
CURRCONF=INICONF;
PLOTMAT(1,:)=CURRCONF;

% Begin iteration cycle

while(time<=MAXT)
    time
    TEMPCONF=zeros(1,N);                                  % Create temp configuration
    for i=1:N
        NEIGH_STATUS=CURRCONF(NEIGH(i,:));                % Copy status of neighborhood
        CELL_INP=bin2dec(num2str(NEIGH_STATUS))+1;        % Compute decimal value of
                                                          % neighborhood
        CELL_OUT=OUT(CELL_INP);                           % Lookup output value in out
                                                          % matrix
        TEMPCONF(i)=CELL_OUT;
    end
    CURRCONF=TEMPCONF;
    PLOTMAT(time+1,:)=CURRCONF;
    time=time+1;
end

imagesc(PLOTMAT,[0 1])
tit=strcat('Rule = ',num2str(RNUM));
title(tit);
xlabel('Cells');
ylabel('Generations');
colormap(1-gray)
```

1dim CAs: Take-Home Messages

1. Allow for generality in your code, but not too much: generality comes at a cost

2. Encoding a model requires finding shortcuts, inventing tricks, learning about the model and sometimes even changing it

3. Always comment the code as much as possible

4. Employ in-built routines to solve your problems, but always understand well how do they work before using them (otherwise build your own version)

Writing the code in C/C++

- **What do you need to get started**
  - A C++ compiler/debugger: Borland C++ Compiler 5.5, GCC, g++
  - A good text editor: Emacs, SourceEdit, Winedt, Dev C++ (IDE), XCode (Mac)
- **A fundamental choice: plain or OOP approach**
  - Rule of thumb: It mainly depends on ABM sophistication
  - Constant population, single type of agents
    - No need for OOP, agents encoded in vectors and arrays
  - Many type of agents, population grows/shrinks in time
    - OOP is better
- **What does it mean encoding agents in vectors and arrays**
  - Population: I={1,2,…,N} constant through time
  - {1,2,…,N} are the labels of the entries of an KxN array
    - Columns: Agents
    - Rows: Variables

Agents, Vectors and Arrays: Example

Time t: Encoding of microeconomic variables

Agents

	1	2	…	N-1	N
Production	Q(1,1)	Q(1,2)	…	Q(1,N-1)	Q(1,N)
Profits	Q(2,1)	Q(2,2)	…	Q(2, N-1)	Q(2,N)
	…	…	…	…	…
Price	Q(K,1)	Q(K,2)	…	Q(K,N-1)	Q(K,N)

A generic time step t

Agents

	1	2	…	N-1	N
Production	Q(1,1)	Q(1,2)	…	Q(1,N-1)	Q(1,N)
Profits	Q(2,1)	Q(2,2)	…	Q(2, N-1)	Q(2,N)
	…	…	…	…	…
Price	Q(K,1)	Q(K,2)	…	Q(K,N-1)	Q(K,N)

Inheriting matrix from time t-1

Agents adjust microeconomic variables

Store time t matrix

Matrix is updated and passed to time t+1

Agents

	1	2	…	N-1	N
Production	Q(1,1)	Q(1,2)	…	Q(1,N-1)	Q(1,N)
Profits	Q(2,1)	Q(2,2)	…	Q(2, N-1)	Q(2,N)
	…	…	…	…	…
Price	Q(K,1)	Q(K,2)	…	Q(K,N-1)	Q(K,N)

C/C++ : Getting started

- **Basic Data Types (signed, unsigned)**
  - Integer: **`int, long`**
  - Floating point: **`float, double`**

C/C++ : Getting started

- **Basic Data Types (signed, unsigned)**
  - Integer: **`int, long`**
  - Floating point: **`float, double`**

- **The simplest C++ program**

```
#include <iostream.h>

int main (void) {
    cout << "Hello world!" << endl;

    return 0;
}
```

C/C++ : Getting started

- **Basic Data Types (signed, unsigned)**
  - Integer: **`int, long`**
  - Floating point: **`float, double`**

- **The simplest C++ program**


```
#include <iostream.h>

int main (void) {
    cout << "Hello world!" << endl;

    return 0;
}
```

C/C++ : Getting started

- **Basic Data Types (signed, unsigned)**
  - Integer: **`int, long`**
  - Floating point: **`float, double`**

- **The simplest C++ program**

{ } : Block

```
#include <iostream.h>

int main (void) {
    cout << "Hello world!" << endl;

    return 0;
}
```

C/C++ : Getting started

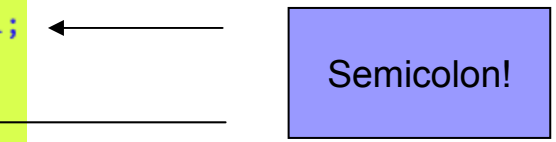
- **Basic Data Types (signed, unsigned)**
  - Integer: **`int, long`**
  - Floating point: **`float, double`**

- **The simplest C++ program**

```
#include <iostream.h>

int main (void) {
    cout << "Hello world!" << endl;

    return 0;
}
```



C/C++ : Getting started

- **Basic Data Types (signed, unsigned)**
  - Integer: **`int, long`**
  - Floating point: **`float, double`**

- **The simplest C++ program**

```
#include <iostream.h>

int main (void) {
    cout << "Hello world!" << endl;

    return 0;
}
```

- **C++ Functions**
  - Always: **`main()`**
  - Structure of function body:

    `[data type out] FunctionName ([data type in]) { }`
  - Important: `return [output data];`
  - Function definition (prior to function body):

    `[data type out] FunctionName ([data type in]);`

C/C++ : Getting started

- **Constants initialization**
  - `const int M=100;`
  - `const double a=1.5;`
- **Scalar variable initialization**
  - `int M=100;`
  - `double a=1.5;`
- **Vectors and arrays (variable) initialization (N must be a constant)**
  - `int M[100];`
  - `double a[N];`
  - `int M[100][100];`
  - `double a[N][N];`

C/C++ : Getting started

- **“Conditional if” statement**

```
if (grades [student] >= passing_grade)
    congratulate_student (student);
else {
    call_rector (student);
    call_parents (student);
}
```

- **“While” loop**

```
while (input_is_valid) {
    congratulate_user ();
    process_input ();
    get_more_input ();
}
```

C/C++ : Getting started

- **“For” loop**

```
int mysum;
int i;
const int size=10;
int vect[size];

for (i=0; i<size; i++)
{
        mysum += vect[i];
}
```

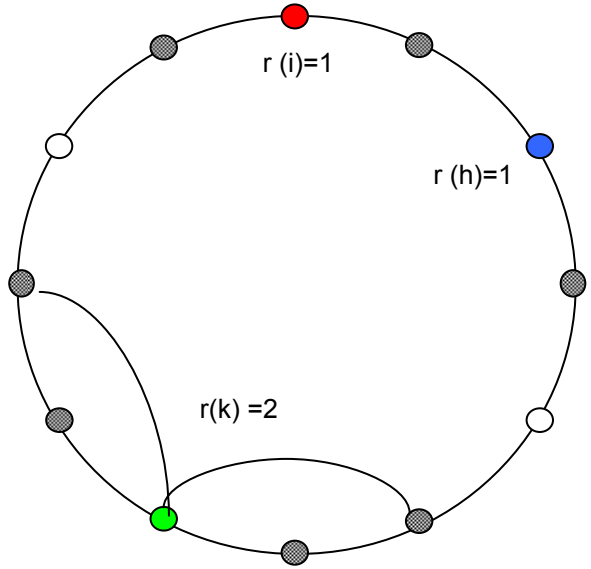
- **Things to know**
  - **a++** means a=a+1; **a+=b** means a=a+b; many abbreviations available…
  - Passing a vector to a function: **float myFun(int[]);**
  - Global and local variables
  - Importance of commenting the code using ****
  - Input/output: **<iostream.h>** and **<fstream.h>**

Example: Dynamic Games

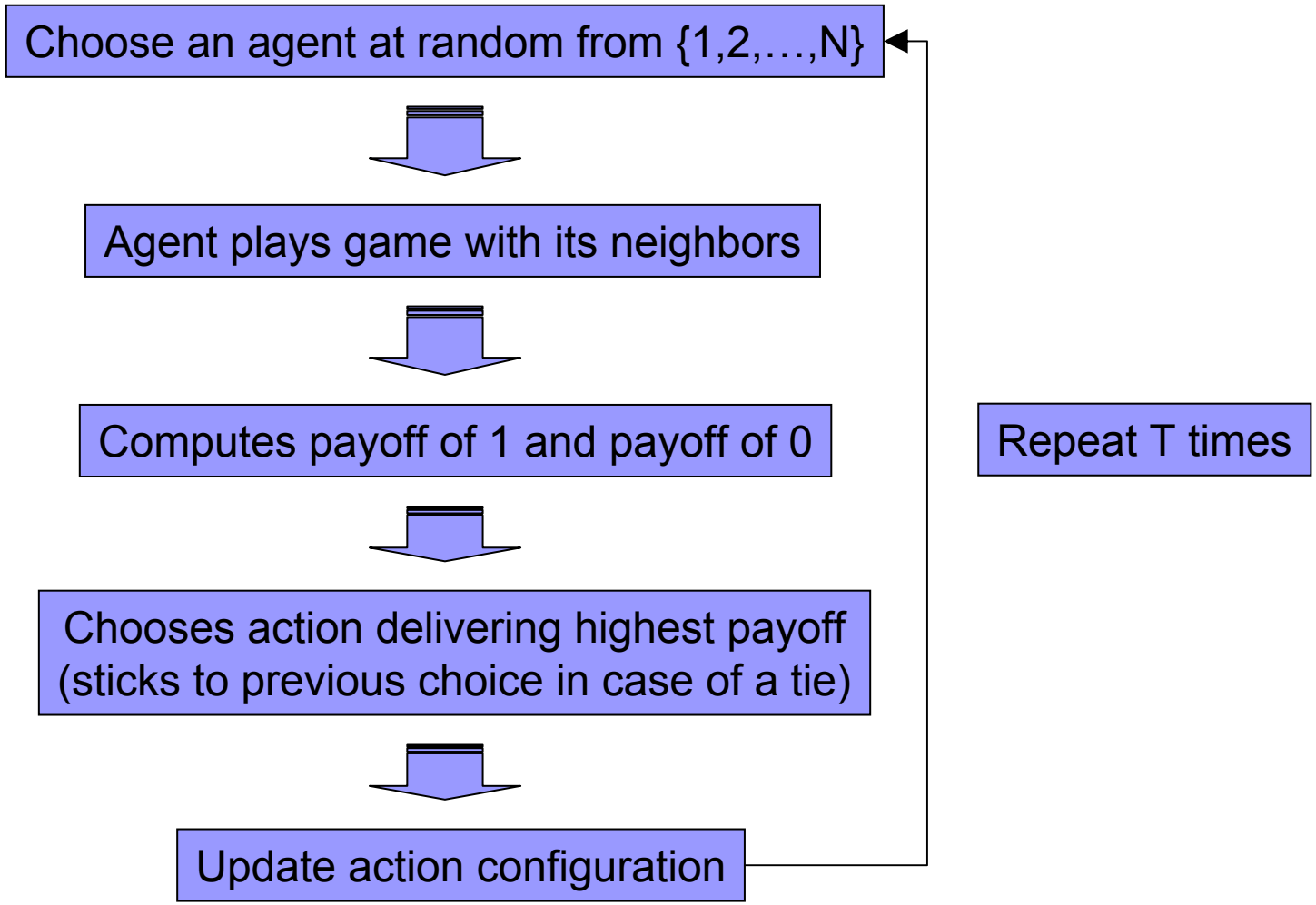
• Time	$t = 0, 1, 2, \ldots$	
• Sets of Agents	$I = \{1, 2, \ldots, N\}$	Players
• Sets of Micro States	$i \rightarrow \{1,0\}$	Pure strategies
• Interaction Structures	G_t = 1-Dim Lattice	Circle
• Micro-Parameters	r(i)	Interaction Radius
• Vector of Macro-Parameters	$\begin{bmatrix} a & b \\ c & d \end{bmatrix}$	Stage-Game Payoffs
• Micro Decision Rules	BR Rule	Strategy Updating
• Aggregate variables	Mean/Var Action	Coordination Level

Example: Dynamic Games

- Time — $t = 0, 1, 2, \ldots$
- Sets of Agents — $I = \{1, 2, \ldots, N\}$
- Sets of Micro States — $i \rightarrow \{1,0\}$
- Interaction Structures — G_t = 1-Dim Lattice
- Micro-Parameters — r(i) — Interaction Radius
- Vector of Macro-Parameters — $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$ — Stage-Game Payoffs
- Micro Decision Rules — BR Rule — Strategy Updating
- Aggregate variables — Mean/Var Action — Coordination Level



Time-t Loop


Choose an agent at random from {1,2,…,N}
Agent plays game with its neighbors
Computes payoff of 1 and payoff of 0
Chooses action delivering highest payoff
(sticks to previous choice in case of a tie)
Update action configuration
Repeat T times

Best Reply Rules w/ Risk Aversion

- Let
  - n the total number of agent neighbors: n=2*r
  - n1 the number of an agent neighbors playing 1
  - n0 the number of an agent neighbors playing 0
  - We have: n0 = n – n1
- Payoffs
  - 1: a*n1+ [n – n1]*b
  - 0: c*n1+ [n – n1]*d
  - The agent chooses 1 if:

$$n1 > \frac{d-b}{(a-c)+(d-b)}n$$

  - In case of = the agent sticks to current choice

Dynamic Games: Writing the Code

- C++ Code
  - See <IntSource.cpp>

- User-friendly VB Platform
  - See <locint.exe>

Object-Oriented Programming (OOP)

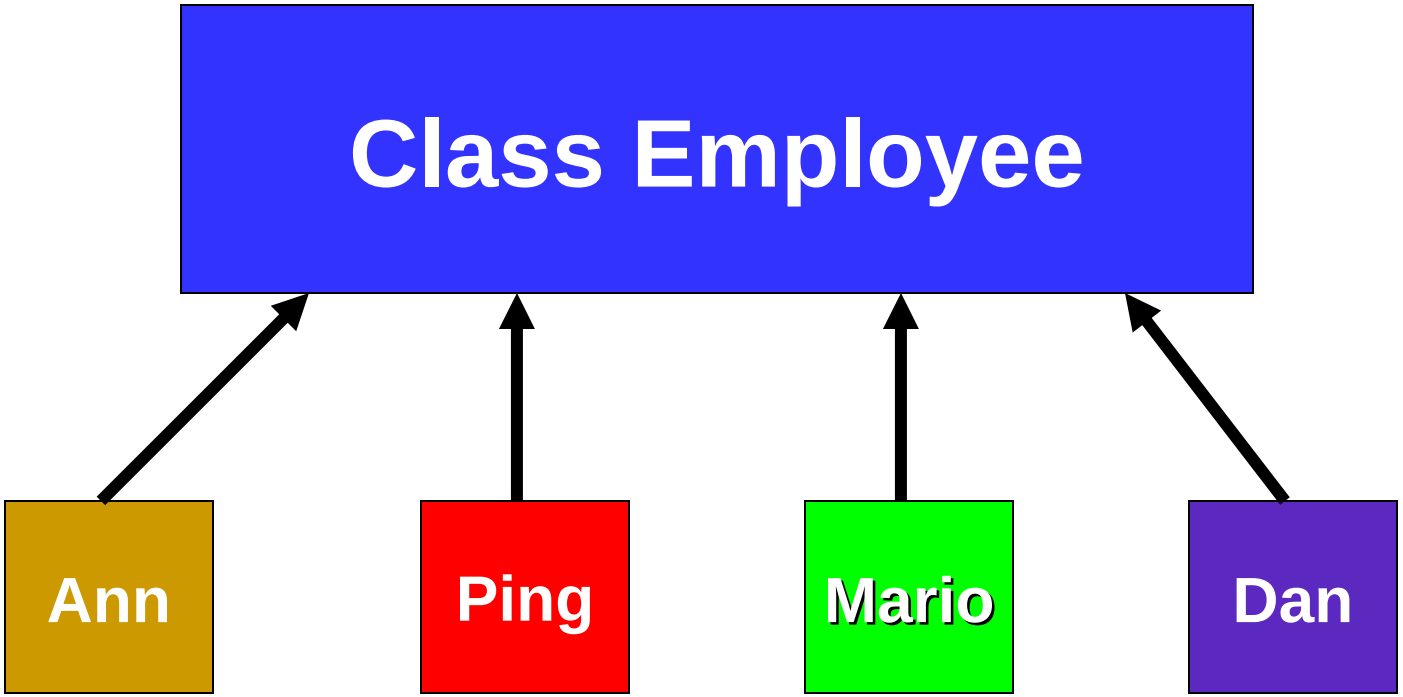
KEY CONCEPTS:

- **Object**
  - Methods (behaviors, functions, procedures,…)
  - Attributes (data, state information,...)
  - Access: public, private, or protected
- **Class**
- **Interface**
- **Encapsulation**
- **Inheritance (subclass, superclass)**
- **Composition**

Object-Oriented Programming (OOP)

- An ***object*** is a software entity containing ***attributes*** plus ***methods*** that act on these attributes.
- An object controls ***access*** to its attributes and methods by declaring them
  - public (accessible to all other objects);
  - private (inaccessible to all other objects);
  - or protected (accessible only to certain designated other objects).
- A ***class*** is a blueprint for an object, i.e., a template used to create ("instantiate") an object.

Class = Object Template



Employee Objects (Instances of Employee)

OOP … Continued

- The public methods and public attributes of an object are called the ***interface*** of the object.
- Objects ***communicate*** with each other via their public methods, i.e., by activating ("invoking") the public methods of other objects.

Illustration: Payroll Class
(invokes public methods in Employee class)

Class **PAYROLL**
{
Public Access:
Methods:
getEmployeeSSN( ) ;
getEmployeeGender( ) ;
getEmployeeDateOfBirth( ) ;
calculateEmployeePay( ) ;
payEmployee( ) ;
Private Access Only:
Attributes:
CurrentProfits ;
EmployeePayoll ;
}

OOP … Continued

- ***Encapsulation*** is the process of determining which aspects of a class are not needed by other classes, and hiding these aspects from other classes.
- More precisely, encapsulation is the process of dividing each class of a program into two distinct parts:

  (1) (public) interface;

  (2) private (or protected) stuff that other classes do not need to know about.

Class Inheritance

- A class C can ***inherit*** the attributes and methods of another class B.
- The class C is then called the ***subclass*** of class B, and class B is called the ***superclass*** of class C.
- A subclass can also include specialized attributes and methods that are not present in the superclass.

Class Inheritance: Example

Superclass of Buyer and Seller

TradeBot

data Price ;
method trade( ) ;

Subclass of TradeBot

Buyer

Price = BidPrice ;
trade( ) = buy( ) ;
calculateUtility( ) ;

Subclass of TradeBot

Seller

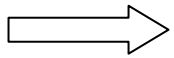
Price = AskPrice ;
trade( ) = sell( ) ;
calculateProfits( ) ;

OOP and C++ : An Example

- OOP-C++ Code for the 2D-CA
  - See <CellAuto.zip> in DevCpp

The Outcomes of ACE/EV Models (1/2)

Micro-Dynamics
(induced by decision rules, interactions and expectations)



Macro-Dynamics
(obtained as aggregation of individual behaviors)

- Stochastic components in decision rules, expectations, interactions imply that the dynamics of micro and macro variables can be described by some (Markovian) stochastic process parametrized by $(\underline{\theta}_i)$, Θ :

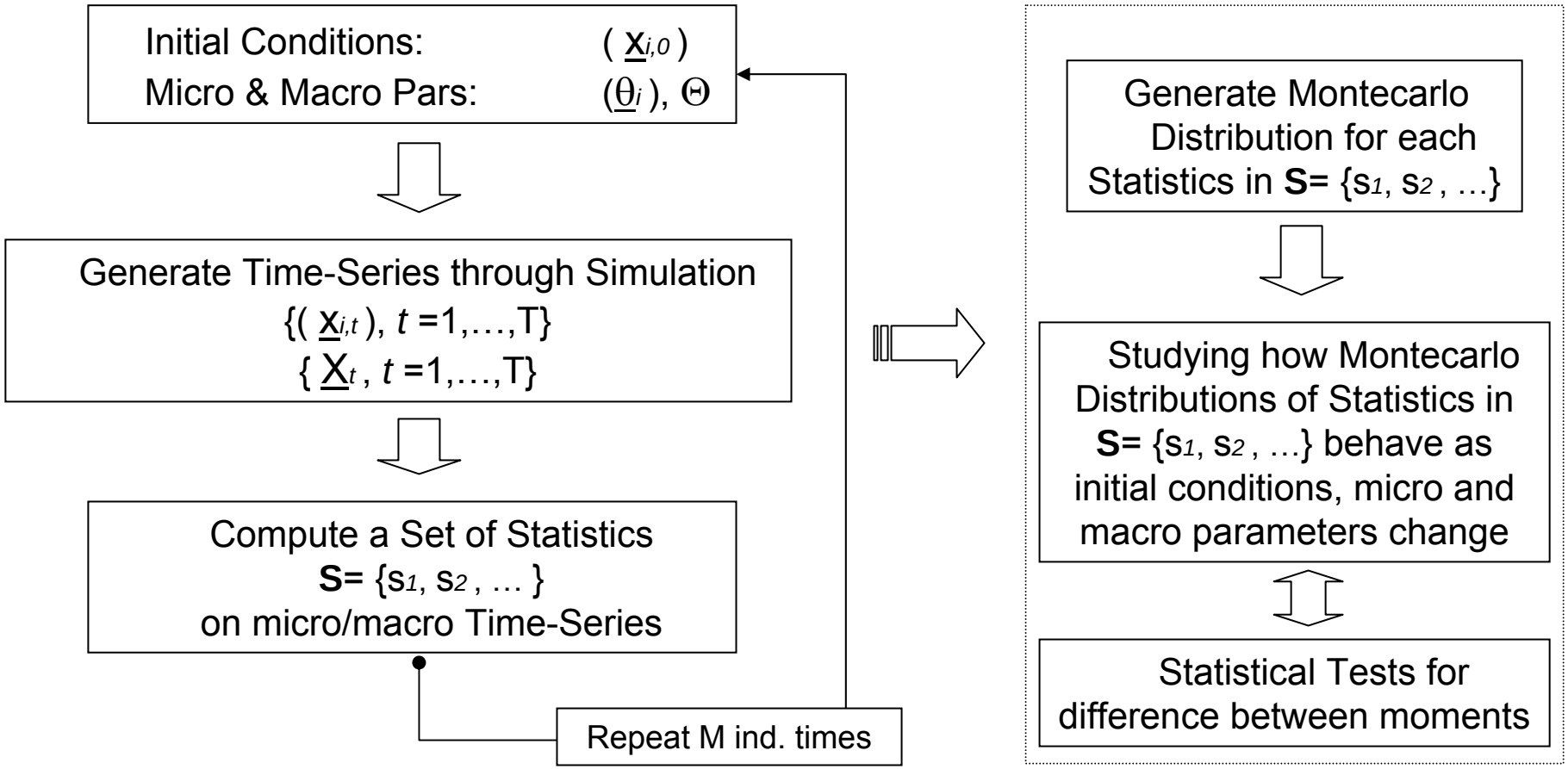
$$(\underline{x}_{i,t}) \mid (\underline{x}_{i,t-1}), (\underline{x}_{i,t-2}), \ldots ; (\underline{\theta}_i), \Theta$$

$$\underline{X}_t \mid (\underline{X}_{t-1}, \underline{X}_{t-2}, \ldots ; (\underline{\theta}_i), \Theta)$$

- Non-linearities in decision rules, expectations, interactions **may** imply that it is **hard** to analytically derive laws of motion, kernel distributions, time-*t* probability distributions, etc.

- Need to resort to **computer simulation** as tool of analysis to study the properties of (stochastic) processes describing $\underline{x}_{i,t}$ and $\underline{X}_t$

Analyzing ABMs: A Recipe



Example: Dynamic Games (1/5)

- Time $t = 0, 1, 2, \ldots$
- Sets of Agents $I = \{1, 2, \ldots, N\}$ Players
- Sets of Micro States $i \rightarrow s(i) \in \{-1,+1\}$ Pure strategies
- Strategic Problem: Overall Coordination out of 2-person games (a > 1.5)

	+1	-1
+1	2a	0
-1	3	2

Pareto-Efficient Strategy →

Risk-Efficient Strategy → if a<2.5

EU(+1)=2a•½+0 •½ = a

EU(-1)=3•½+2 •½ = 2.5

A Primer on Coordination Games

- Importance and meaning of coordination among individuals

- Examples of coordination patterns
  - Choice of compatible technologies (new vs. old technologies, existing standards),
  - Evolution of conventions (languages, currencies, codes of dress, accounting standards, etc.)
  - Cf. H.P. Young (1998)

- Standard analysis of a coordination problem. Ingredients:
  - Game among n individuals (n=2)
  - Static: No time involved
  - One-Shot: Only one stage of decisions
  - There are $k \geq 2$ options available

- Definition of a coordination game:

> A **coordination game** is a stage game where there is some incentive to choose the strategy that you expect your opponent to play.

- A General Coordination Stage-Game:

A ; B	+1	–1
+1	a ; a	b ; c
–1	c ; b	d ; d

Ass: $a>c$ & $d>b$

- Definition of

1. Nash equilibrium (NE)

   → Which are the NE of the game ?

   → Multiple equilibria !

2. Pareto efficient NE

A NE is Pareto efficient if there are no other NE wherein an individual is strictly better off while the other is no worse off.

- a>d: (+1,+1) PE;
- a<d: (–1,–1) PE;
- a=d: Both NE are Pareto equivalent

- Some examples:

1. Pure coordination game: a=d, b=c

A\B	+1	−1
+1	1	0
−1	0	1

Features: Two NE, both Pareto equivalent

Prediction: (Efficient) coordination will always arise; if the two outcomes can be discriminated, no way of saying which one will be selected.

2. Coordination game with a Pareto Efficient NE : a>d, b=c

A\B	+1	−1
+1	5	0
−1	0	1

Features: Two NE, (+,+) is Pareto efficient

Prediction: Coordination will always arise; no way of saying which NE will be selected; $\Rightarrow$ inefficient NE can be selected but we cannot say when and why !!

- Definition of **Risk-dominant NE**

  - PE is not the only way of ranking NE: Possible loss counts !

5	0
0	1

5	−10
0	1

  - If agents take into account possible losses and they have no idea about opponent move (+1,+1) will be better than (−1, −1) in terms of lower risk iff:

$$a+b > c+d$$

or also iff:

$$p = \frac{(d-b)}{(a-c)+(d-b)} < \frac{1}{2}$$

  - NB: *p* is the probability that player A assigns to the event that his opponent is playing −1 necessary to leave A indifferent between the +1 and −1 strategies. If *p* is smaller than ½ than A is taking a low risk in choosing +1.

3. Coordination game with a PE and a RD equilibrium (a>d but a+b < c+d)

A\B	+1	−1
+1	3	−1
−1	1	2

Features: Two NE, (+,+) is Pareto efficient BUT (−,−) is Risk Dominant:

$E\pi(+1) = 3 \cdot ½ + (-1) \cdot ½ = 1$

$E\pi(-1) = 1 \cdot ½ + (2) \cdot ½ = 1.5$

Prediction: Coordination will always arise; no way of saying which NE will be selected; $\Rightarrow$ inefficient NE is more likely in terms of risk-dominance !!

Example: Dynamic Games (1/5)

- Time $t = 0, 1, 2, \ldots$
- Sets of Agents $I = \{1, 2, \ldots, N\}$ Players
- Sets of Micro States $i \rightarrow s(i) \in \{-1,+1\}$ Pure strategies
- Strategic Problem: Overall Coordination out of 2-person games (a > 1)

	+1	-1
+1	2a	0
-1	3	2

Pareto-Efficient Strategy →

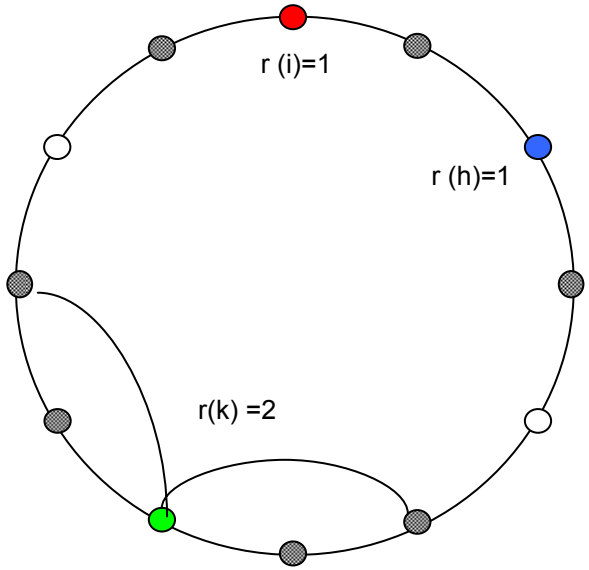
Risk-Efficient Strategy → if a<2.5

EU(+1)=2a•½+0 •½ = a

EU(-1)=3•½+2 •½ = 2.5

Example: Dynamic Games (2/5)

- Interaction Structures G_t = 1-Dim Lattice Circle
- Each agent i interacts with neighbors closer than r(i)



$$V(i) = \{ j : | i - j | \leq r(i) \}$$

Example: Dynamic Games (3/5)

- Micro-Parameters *r*(i) Interaction Radius
- Macro-Parameter *a* Stage-Game payoff of (+1,+1)
- Micro Decision Rules and Dynamics
  - At t=0 random draw of strategies
  - At each t>0 one agent is chosen at random
  - Chooses s_t(i) s.t. max total payoffs given neighbors choices at t-1

$$s_t^*(i) = \arg\max_{s \in \{-1,+1\}} \sum_{j \in V(i)} u(s; s_{t-1}(j))$$

Example: Dynamic Games (4/5)

- Aggregate Variable: LR Coordination Level

$$c = \frac{1}{N} \sum_{i=1}^{N} s_T(i) \in [-1,+1]$$

- Choosing T large enough (stability/convergence of moments)

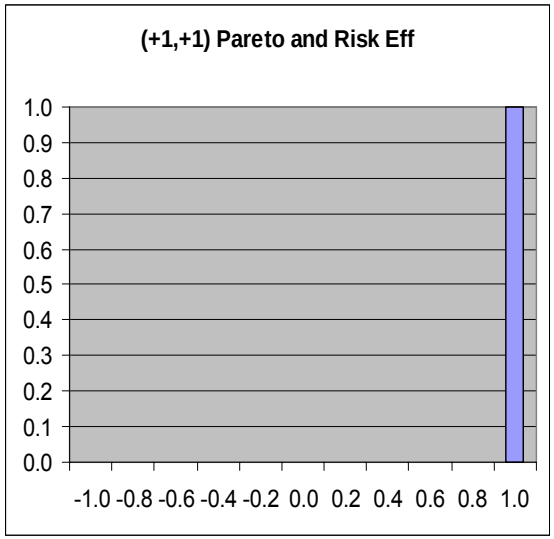
- Goal: Studying MC distributions of LR coordination levels as a function of

    1) Aggregate Parameter (a)
    2) Micro Parameters (e.g. average radius)

Example: Dynamic Games (5/5)

- Results with r(i)=1 all i:

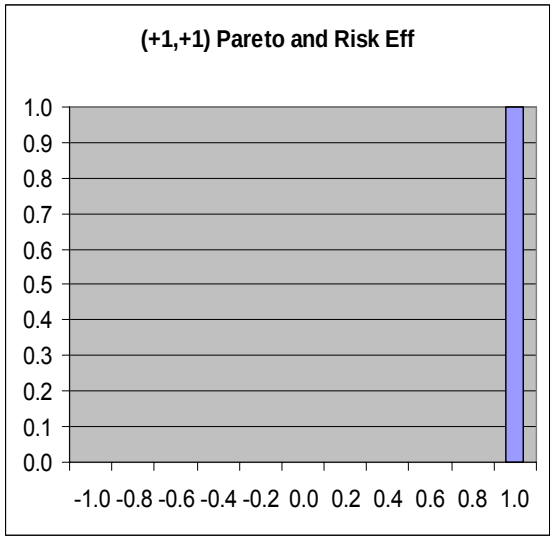
  1) (+1,+1) Pareto-Efficient and Risk Efficient (a > 2.5)
  2) (+1,+1) and (-1,-1) Risk Equivalent (a = 2.5)
  3) (-1,-1) Risk Efficient (a < 2.5)

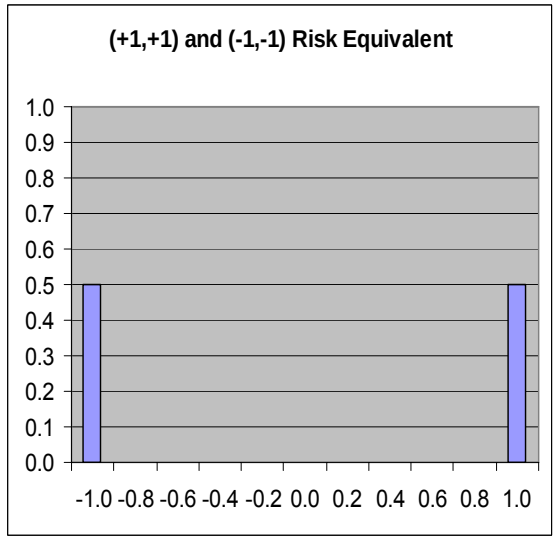


Example: Dynamic Games (5/5)

- Results with r(i)=1 all i:

  1) (+1,+1) Pareto-Efficient and Risk Efficient (a > 2.5)
  2) (+1,+1) and (-1,-1) Risk Equivalent (a = 2.5)
  3) (-1,-1) Risk Efficient (a < 2.5)

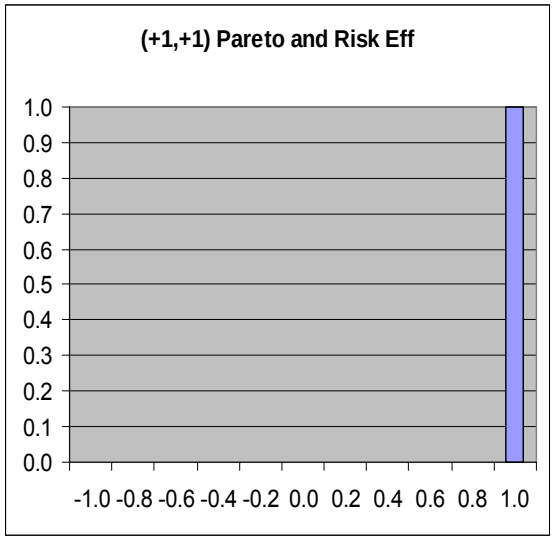


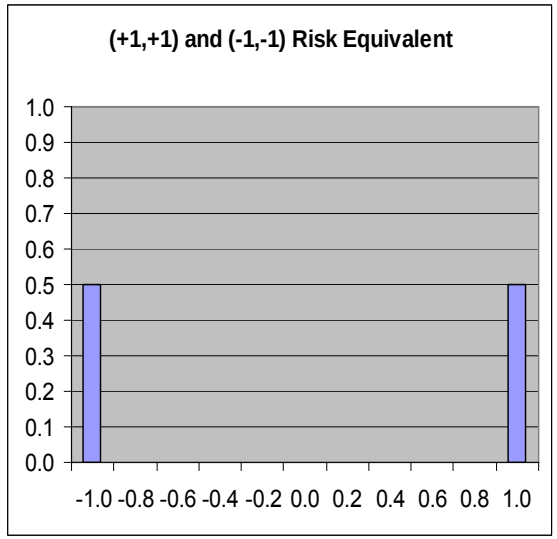


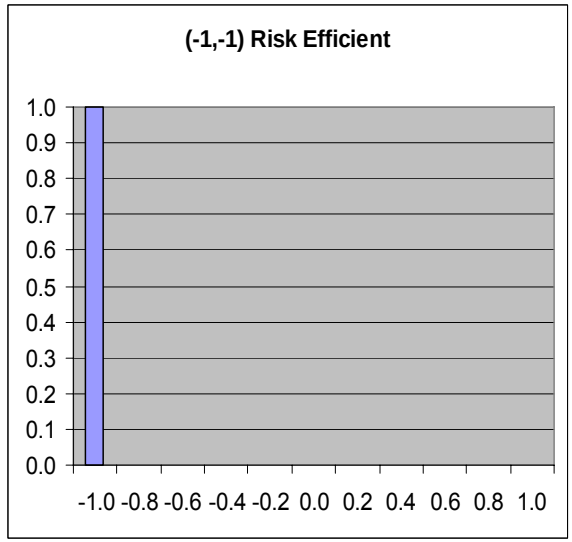
Example: Dynamic Games (5/5)

- Results with r(i)=1 all i:

  1) (+1,+1) Pareto-Efficient and Risk Efficient (a > 2.5)
  2) (+1,+1) and (-1,-1) Risk Equivalent (a = 2.5)
  3) (-1,-1) Risk Efficient (a < 2.5)



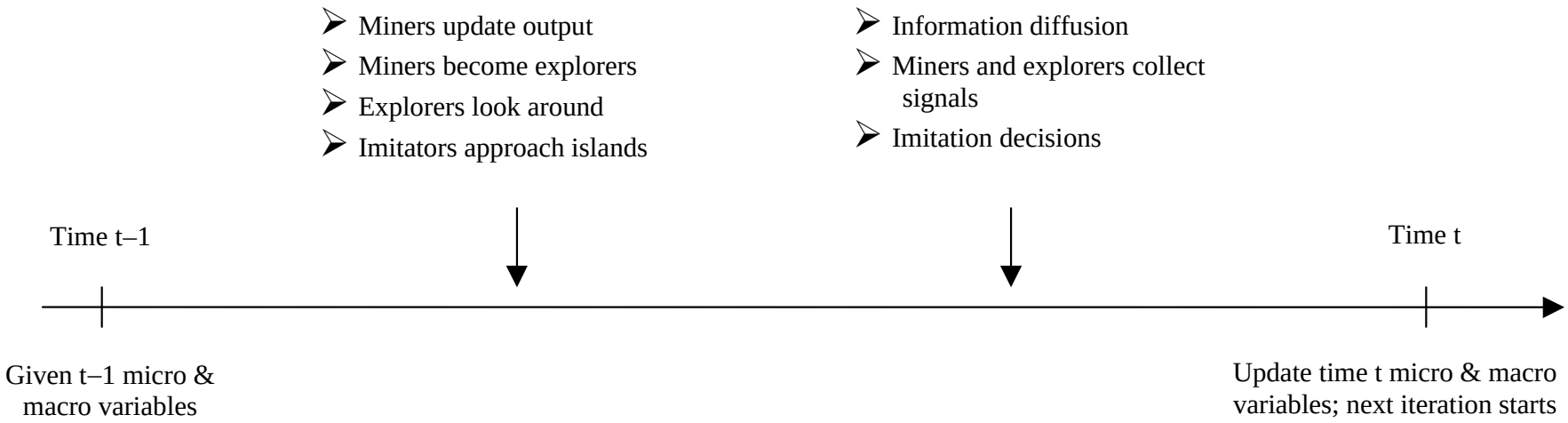




The Islands Model (Fagiolo and Dosi, 2003)

Technological Space	Notionally Unbounded Sea
Technology	Island (‘mine’)
Output	Homogeneous Good
Firms	Stylized Entrepreneurs
Production	Mining/Extracting the Good
Technological Search	Exploration of the Sea
Innovation	Discovering a new island
Technological Diffusion	Spreading knowledge from islands
Imitation	Traveling between already known islands
Technological Difference	Distance between Islands

The Islands Model (Fagiolo and Dosi, 2003)



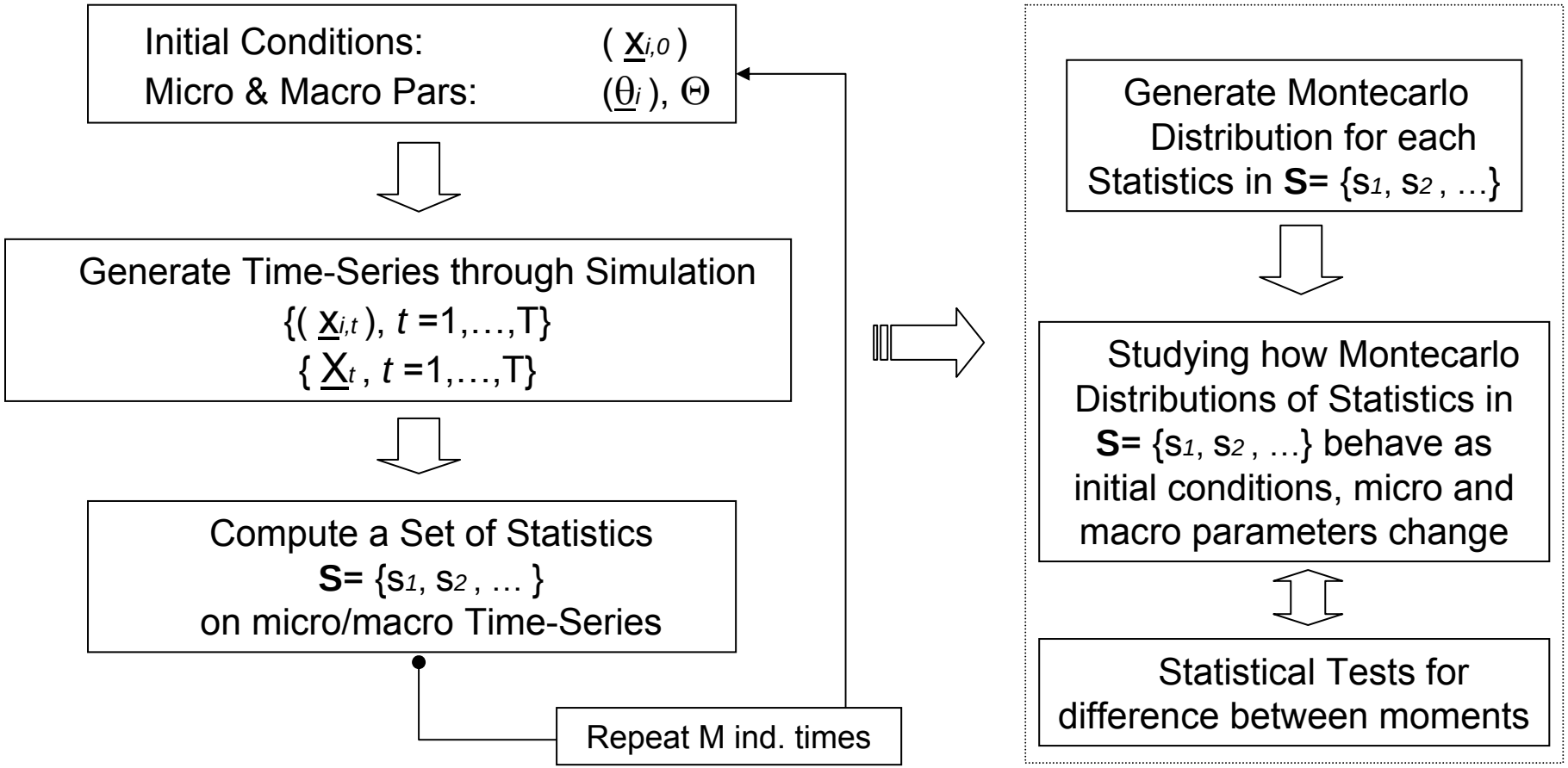
- Focus on
  - Aggregate output (sum of firms' output) and growth rates
  - Number of explorers, imitators, miners

The Islands Model (Fagiolo and Dosi, 2003)

- Model parameters

ρ : globality of information diffusion
φ : path-dependency in learning
λ : likelihood of radical innovations
π : baseline opportunity conditions
α : increasing returns to scale in exploitation
ε : willingness to explore
N : population size
T : time horizon

Analyzing simulation output



A first question…

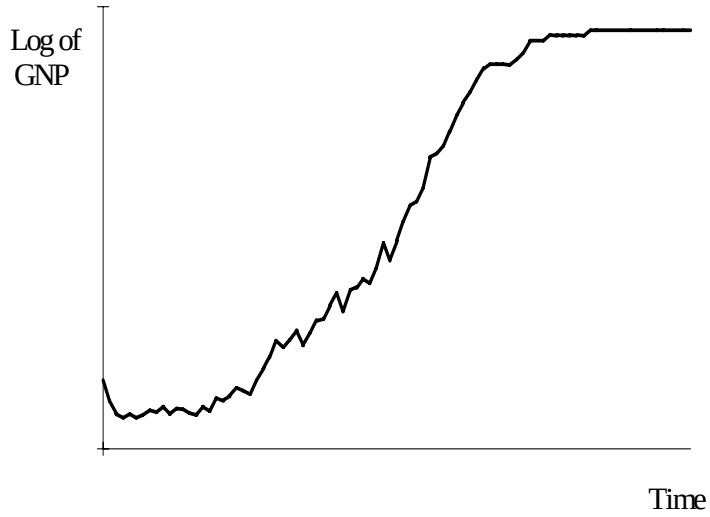
Under which general conditions
is the economy able to generate
self-sustaining growth
as the outcome of the
joint processes of exploitation and exploration ?

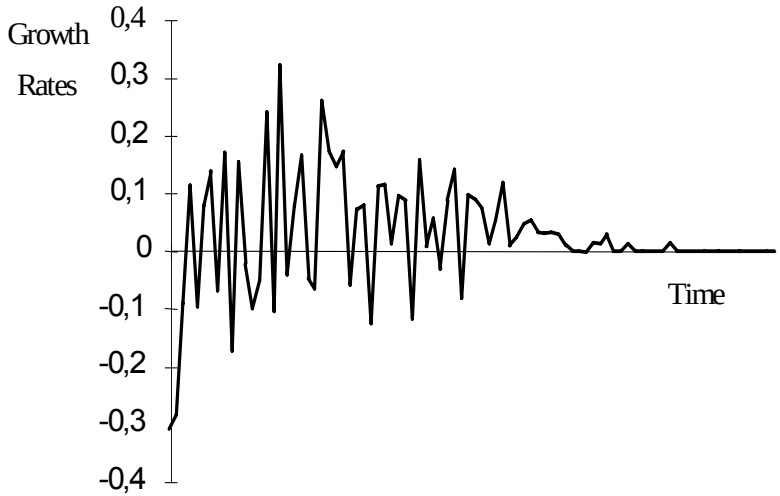
A closed economy **without** exploration (1/2)

- Shutting down exploration and innovation
  - A given initial set of islands (e.g, only 2)
  - Firms initially mining on them (50%, 50%)
  - They can only exchange information among the 2 existing technologies (initial set of islands cannot be expanded)
- Diffusion of information drives growth
  - In this case the model is analytically solvable!
  - Whenever an island manages to capture all agents the growth process stops (growth rates are zero)
  - The process is path-dependent and possibly inefficient (convergence toward an inefficient level of output is a non-zero probability event)

A closed economy **without** exploration (2/2)

- Growth is always a transitory phenomenon





- Lock-in may occur on the ex-ante less efficient island

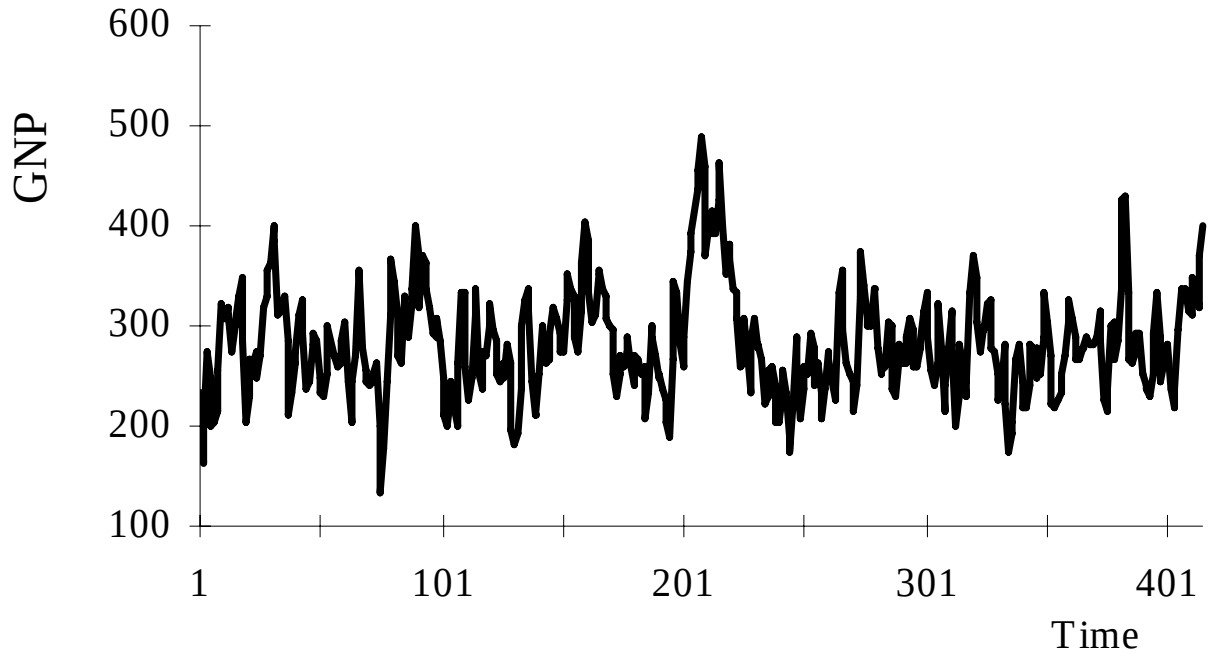
A closed economy **with** exploration (1/4)

- Allowing for exploration in a closed box
  - Initial set of islands cannot be expanded (no innovation)
  - Explorers are allowed to search only inside initial box
  - Imitation still occurs as before

- Diffusion of information still drives growth
  - Process driven by information diffusion
  - Steady states can be destabilized by ‘irrational’ entrepreneurs who decide to leave their island even if everyone is there

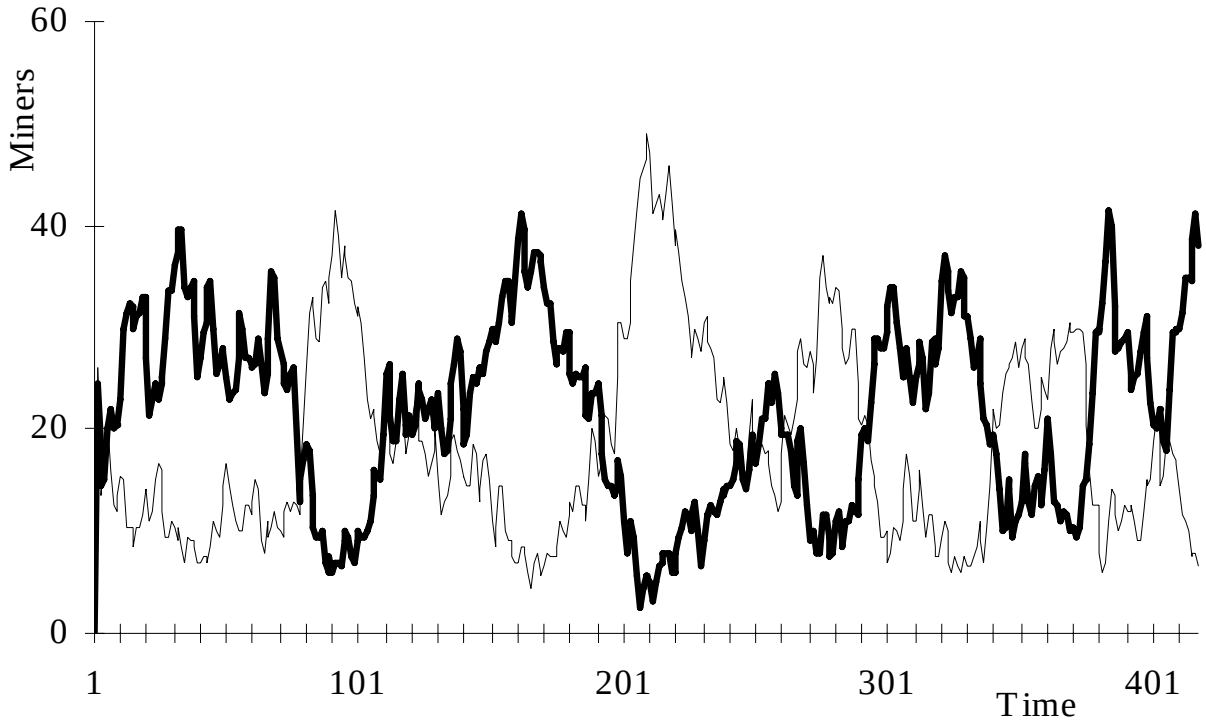
A closed economy **with** exploration (2/4)

- Absorbing states become basins of attraction: growth is a transitory phenomenon but fluctuations can arise



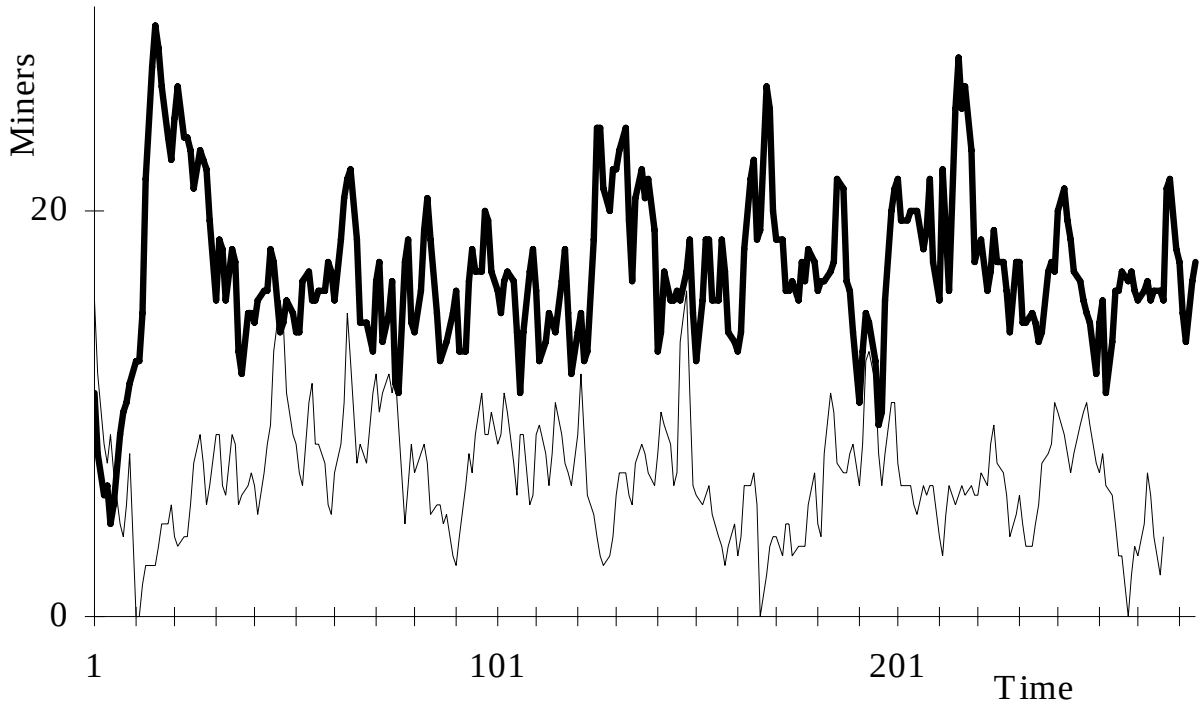
A closed economy **with** exploration (3/4)

- Two ex-ante equally efficient islands



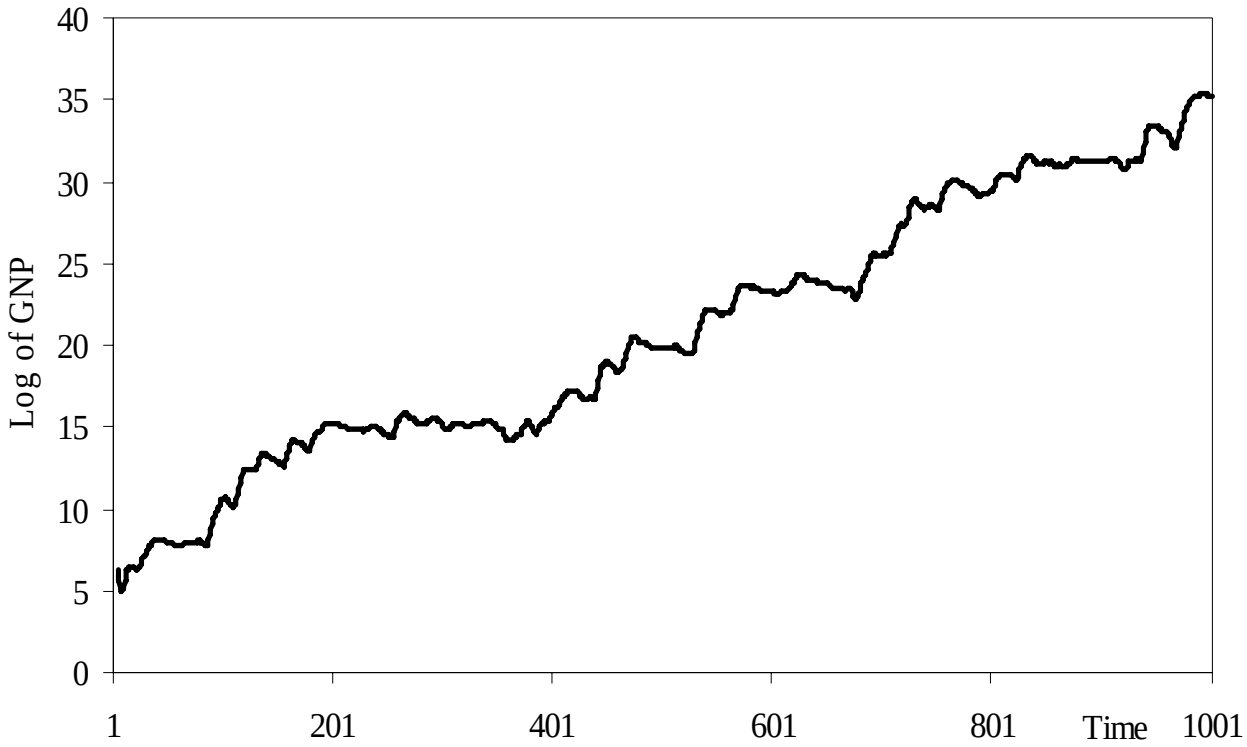
A closed economy **with** exploration (4/4)

- One ex-ante more efficient island: temporary inefficiency may arise



Exploration in a Open-Ended Economy

- In the full-fledged model self-sustaining growth can arise!

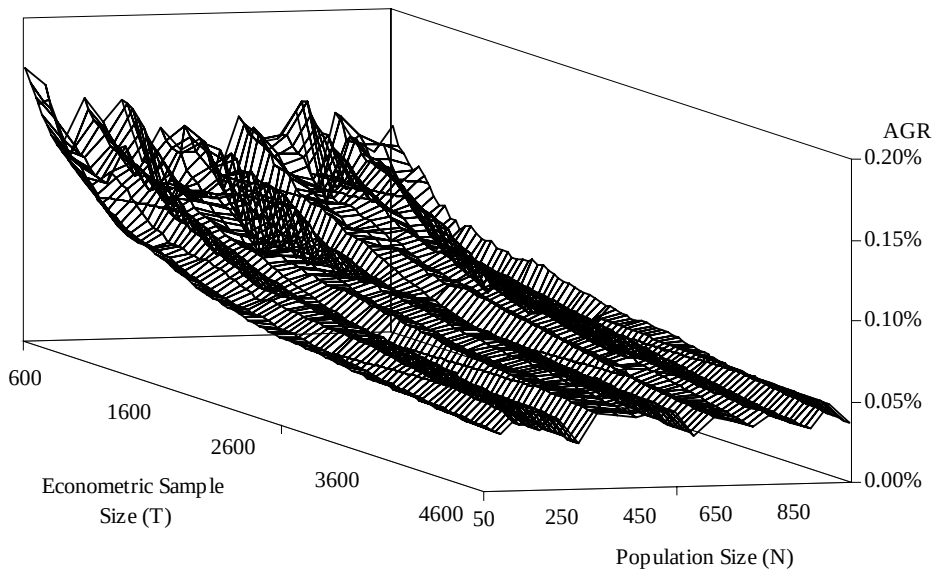


A second question…

When the economy does generate
self-sustaining growth (full-fledged model),
do log(GNP) time-series display empirically observed statistical properties?

Statistical Properties of Simulated GNP Series

- Yes, if self-sustaining growth does emerge
  - log(GNP) time series are I(1), i.e. difference-stationary
  - growth rates are positively correlated over short horizons
  - persistence of shocks are in line with empirical evidence
- Scale-effects are not present
  - As in reality, unlike in many endogenous growth models are!

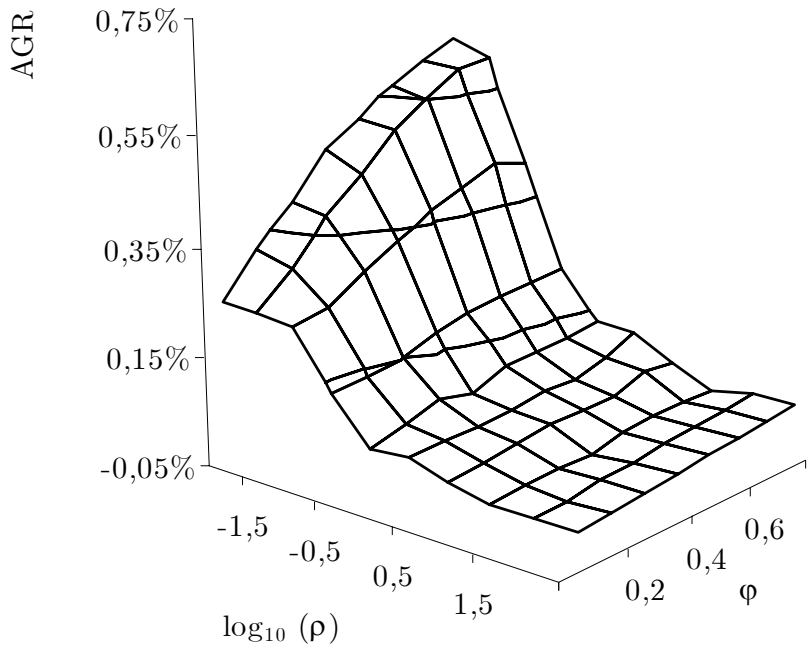


A third question…

When the economy does generate
self-sustaining growth (full-fledged model),
what are the roles played by
system parameters
(i.e. by the sources of growth)?

The Sources of Growth (1/4)

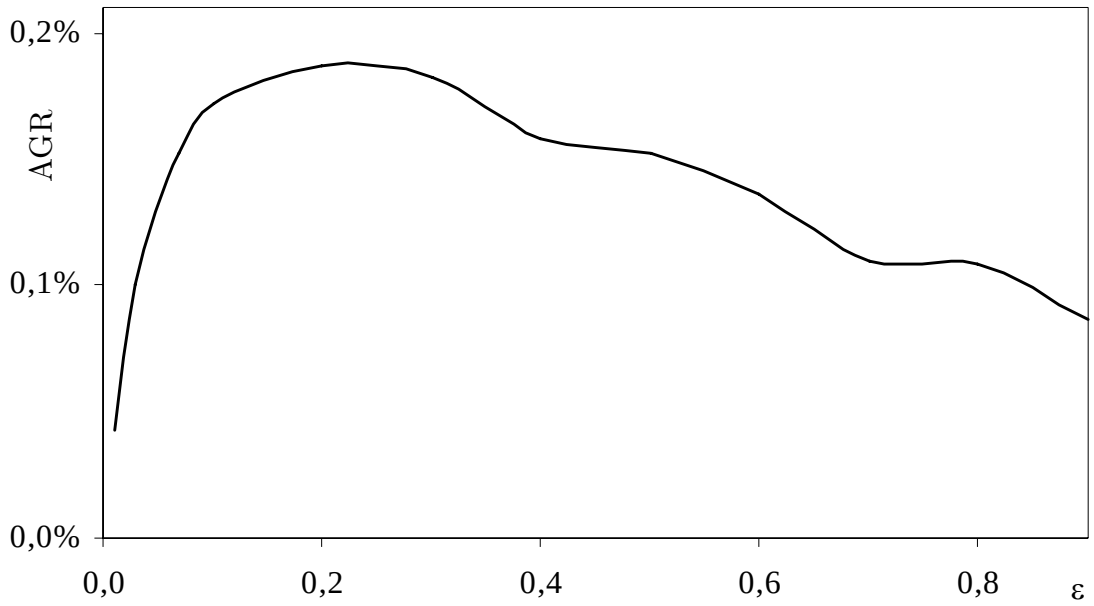
- Average growth rates (AGRs) increasing in
  - path-dependency in knowledge accumulation
  - globality of information diffusion



  - … as well as in returns-to-scale strength and opportunities

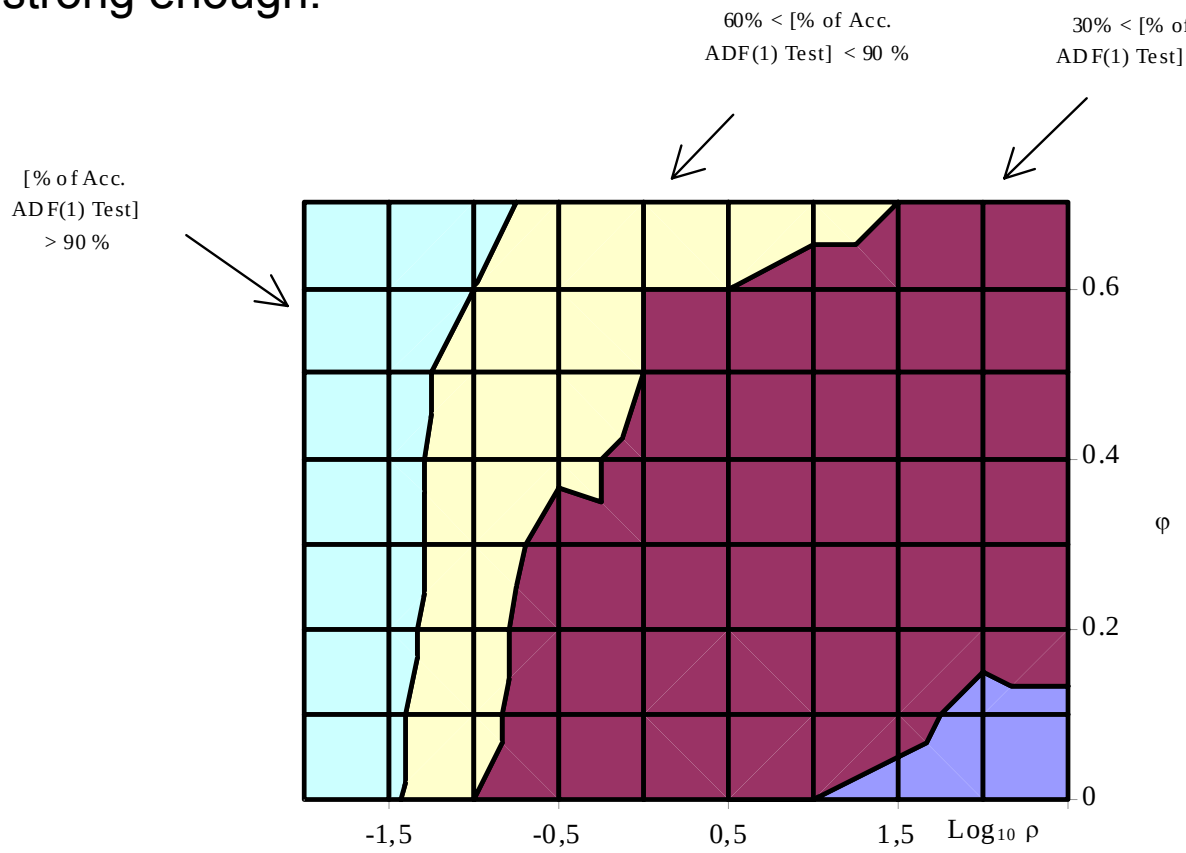
The Sources of Growth (2/4)

- The exploitation-exploration trade-off
  - AGR are maximized only if there is a balance between resources devoted to exploration and resources devoted to exploitation



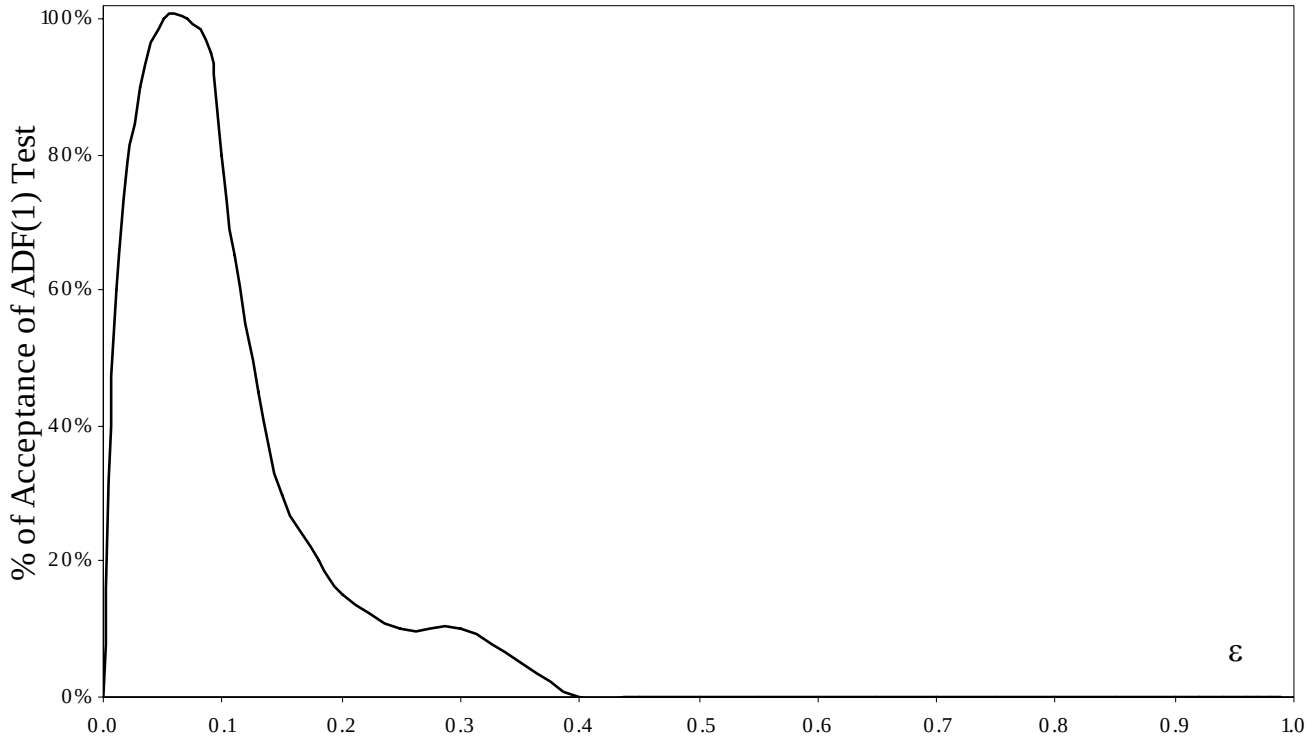
The Sources of Growth (3/4)

- Emergence of thresholds
  - I(1) log(GNP) time-series only emerge if increasing returns to scale, opportunities, path-dependency and globality of information are strong enough!



The Sources of Growth (4/4)

- Emergence of thresholds
  - … and if the exploitation-exploration trade-off is solved



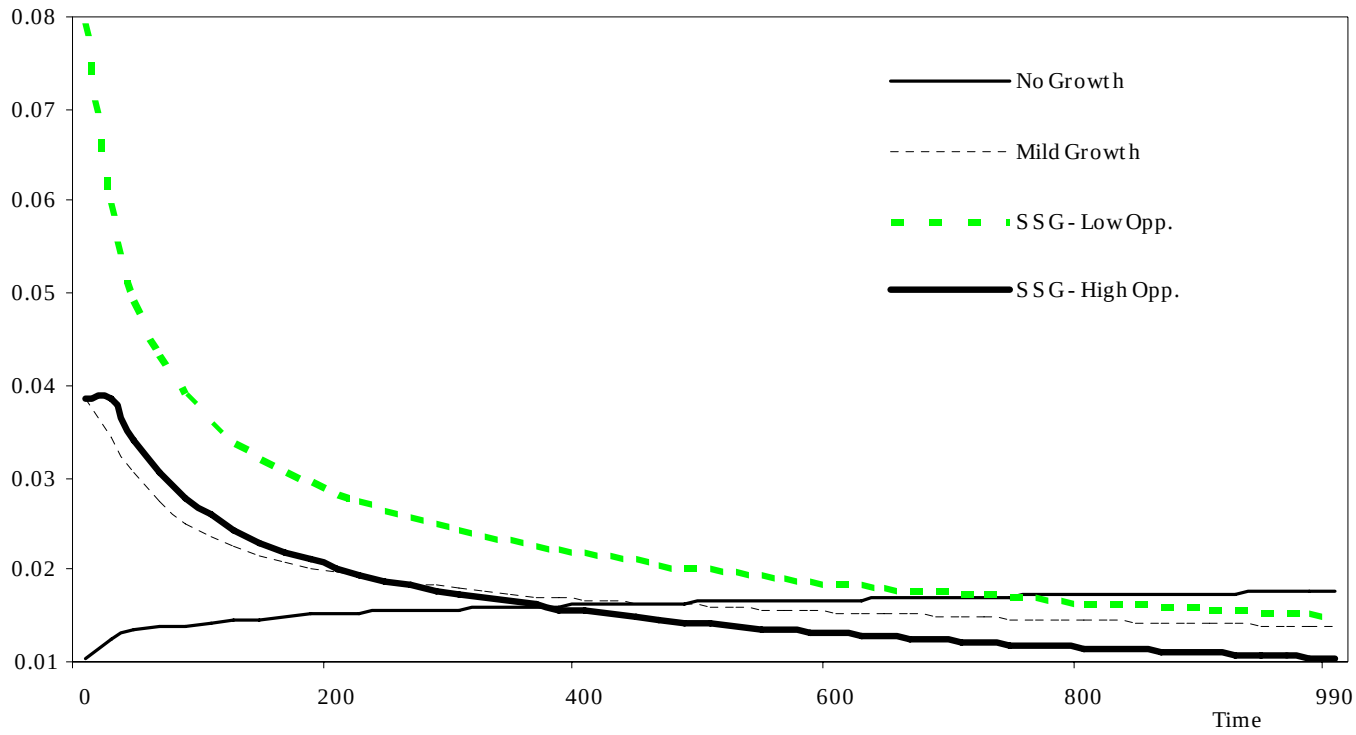
A fourth question…

Does the self-sustaining growth process generated by the model lead to explosive growth patterns?

Does the variability of growth rates increase over time and tends to infinity?

Time Evolution of GNP Growth Rate Variability

- Higher growth is always associated to smaller GR variability!
  - Self sustained growth is a self-organized process leading to ordered growth patterns

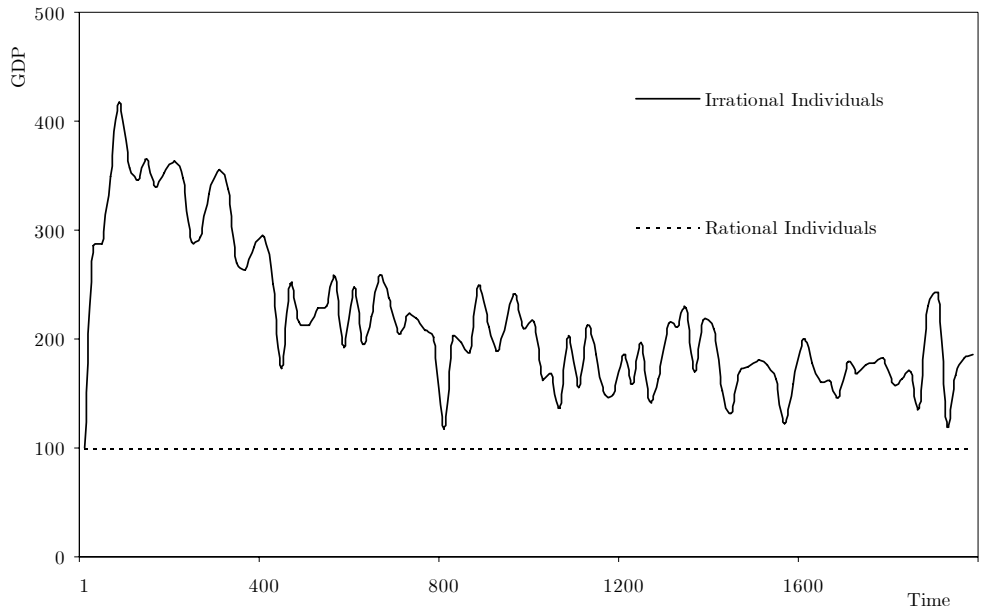


A final question…

What happens in we inject in the economy more rational firms?

Irrationality as a necessary condition for growth

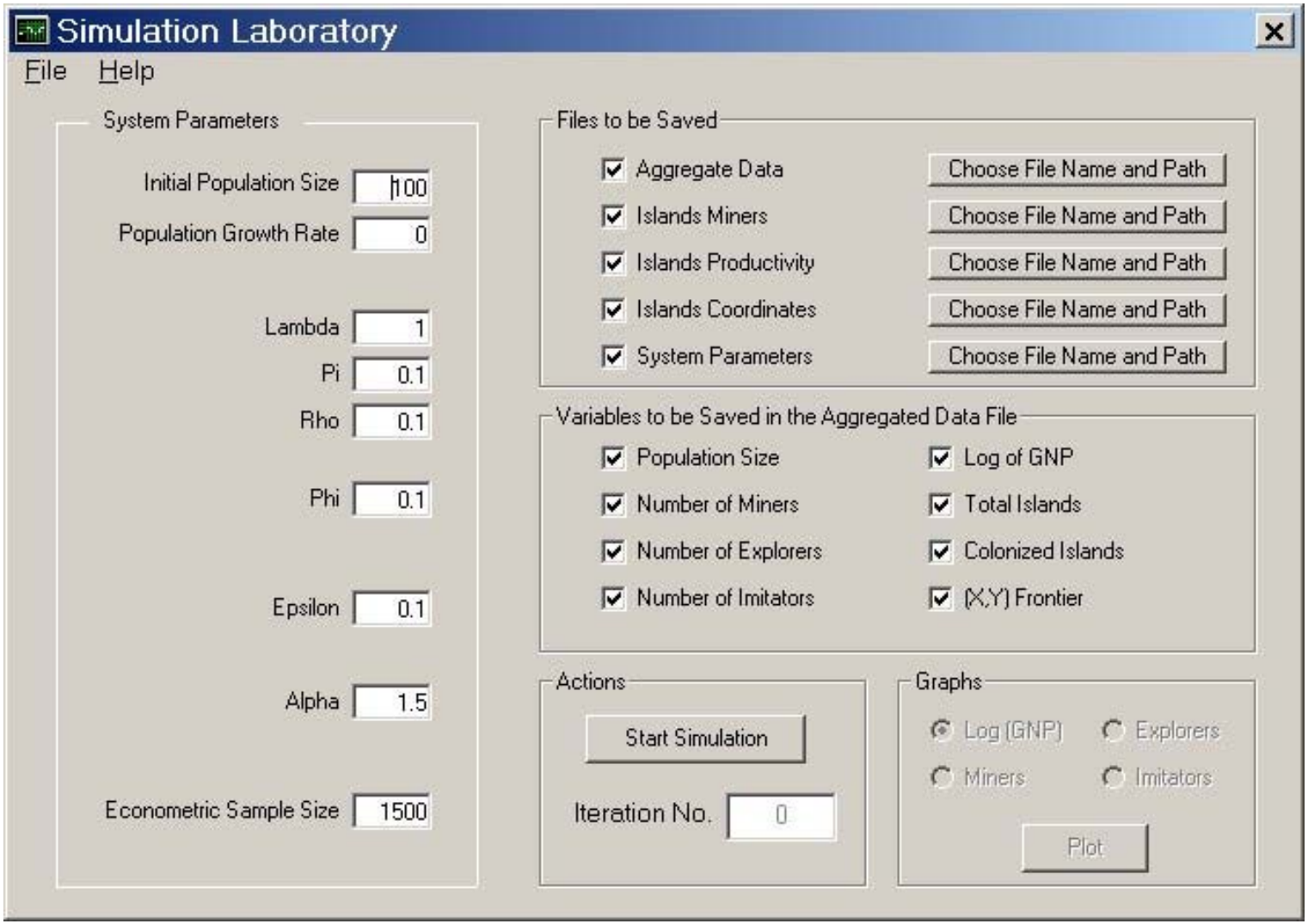
- Simple setup
  - CRTS, no info diffusion, no path-dependency
  - Injecting in the economy a representative rational firm (RRF) who decides whether to exploit or explore by maximizing expected returns
  - RRF knows the structure of the economy and the direction where best islands are (but not where they are)



A laboratory for further research

- Possible extensions
  - Learning
  - Multi-layer economies
  - Demand side and Keynesian cycles
  - Growth and development
  - …

… and have fun …


Simulation Laboratory
File
Help
System Parameters
Initial Population Size
100
Population Growth Rate
0
Lambda
1
Pi
0.1
Rho
0.1
Phi
0.1
Epsilon
0.1
Alpha
1.5
Econometric Sample Size
1500
Files to be Saved
Aggregate Data
Choose File Name and Path
Islands Miners
Choose File Name and Path
Islands Productivity
Choose File Name and Path
Islands Coordinates
Choose File Name and Path
System Parameters
Choose File Name and Path
Variables to be Saved in the Aggregated Data File
Population Size
Log of GNP
Number of Miners
Total Islands
Number of Explorers
Colonized Islands
Number of Imitators
(X,Y) Frontier
Actions
Start Simulation
Iteration No.
0
Graphs
Log (GNP)
Explorers
Miners
Imitators
Plot