Aalto University
School of Electrical Engineering

Lecture 8: Using Mathematica to Simulate Markov Processes

Pasi Lassila
Department of Communications and Networking

Aim of the lecture

- Use Mathematica to implement simple examples

- We consider 2 examples
  - General Markov process
  - Simple M/M/1 queue simulator (birth-death process)

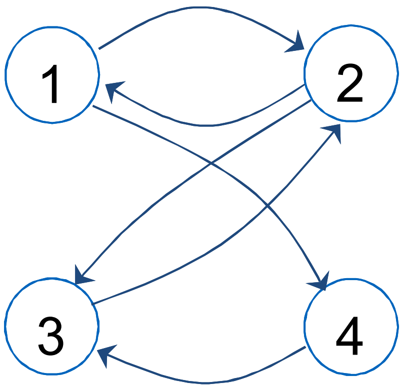
- Idea is to write Mathematica code yourself!
  - So start a Mathematica notebook on your computer
  - We implement simple models that already work
  - They help you in completing the weekly assignments

Contents

- Simulation of a simple general Markov process
- Simulation of M/M/1 queue
- Assignment 1

Concrete example

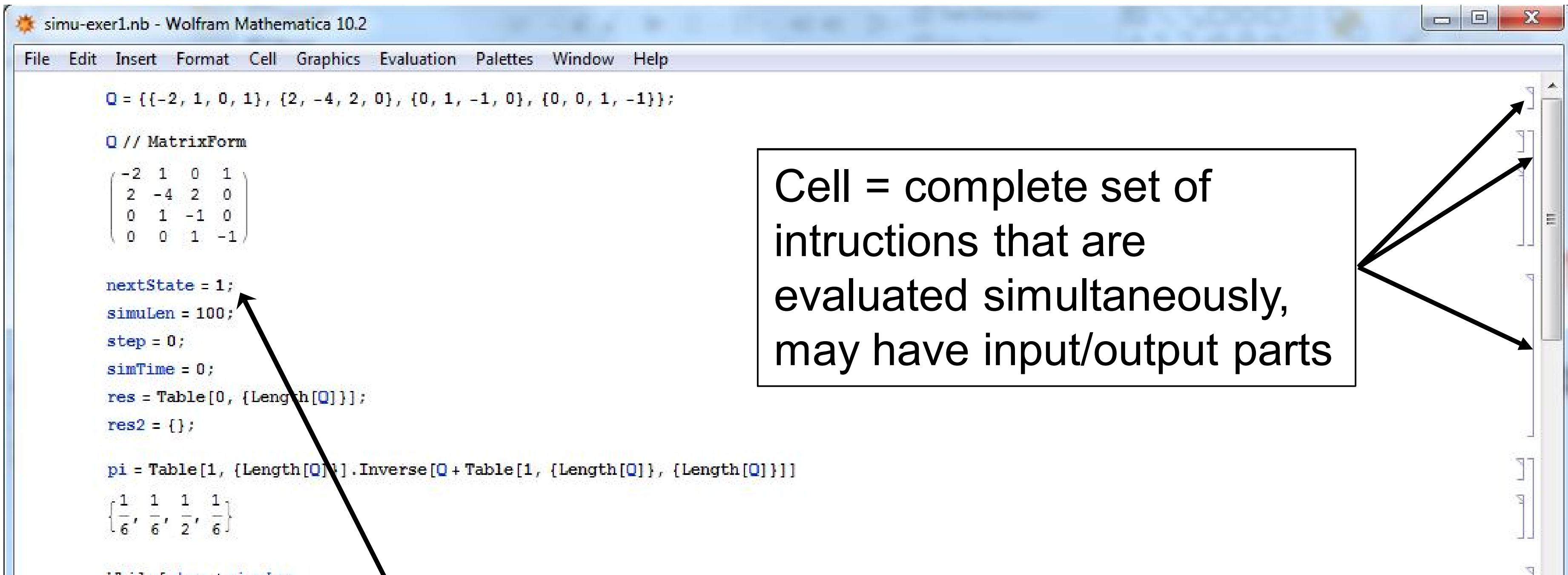
- Consider a Markov process $X(t)$, with state space S = {1,2,3,4} and the following transition rate matrix Q



$$Q = \begin{bmatrix} -2 & 1 & 0 & 1 \\ 2 & -4 & 2 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & -1 \end{bmatrix}$$

- Next we are going to implement this on Mathematica

Mathematica notebook window

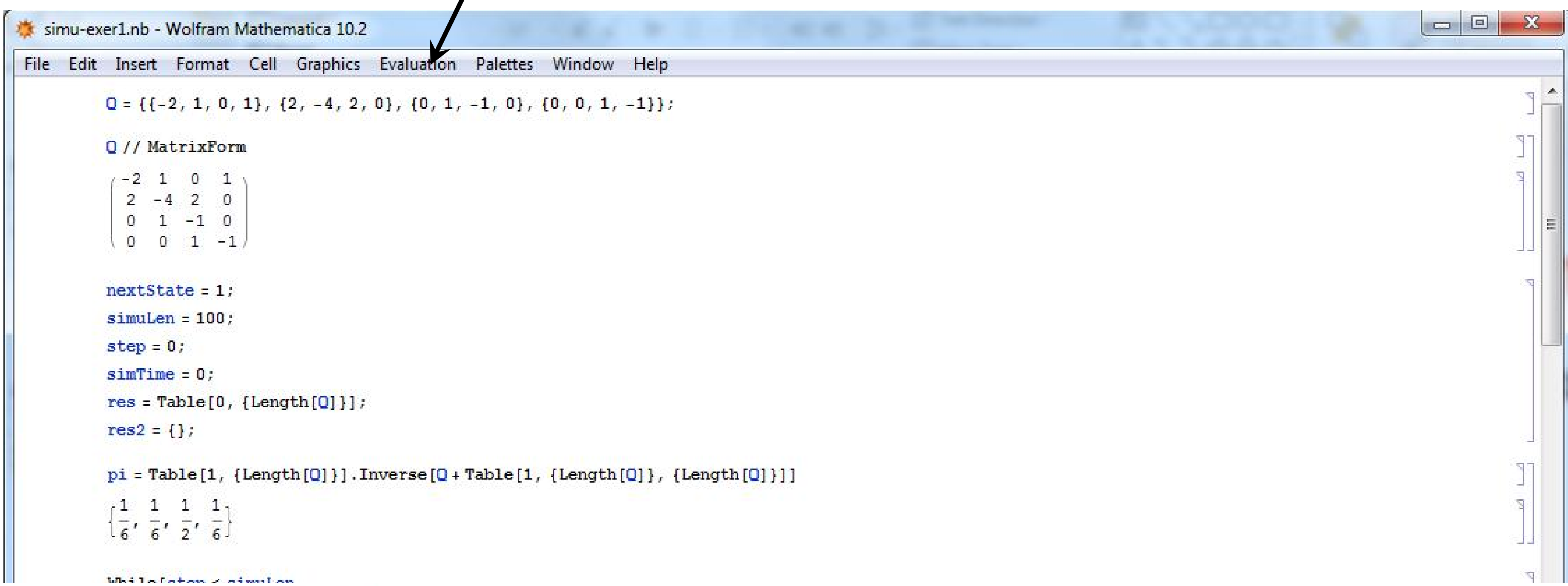


Cell = complete set of intructions that are evaluated simultaneously, may have input/output parts

- A semi-colon (;) at end of statement means that no output is produced for that command
- To evaluate a cell you must press [SHIFT + RETURN] !!
  - Type 2+2 on your notebook window and evaluate

Mathematica kernel

- Mathematica consists of the notebook window and the computational engine, the **kernel**
  - When you evaluated 2+2, the kernel process was started
- In the Evaluation-menu there are commands to control kernel
  - Abort evaluation
  - Quit kernel (if nothing else helps)



Recap of Method 1 for simulating a Markov process

- Aim: Simulate process $X(t)$ with initial state x_0 for K transitions

- Initialize: state $x=x_0$ and transition counter $\mathrm{step}=0$
- Stopping condition: If $\mathrm{step} < K$, then
  - Draw a sample $t_j(x)$ of times to next possible events in state x for all $j=1,\ldots,N$, i.e., each $t_j(x) \sim \mathrm{Exp}(q_{xj})$
  - The holding time (time to next transition) in state x, is given by $\min\,(t_1(x),\ldots,\, t_N(x))$
  - Next state x where the process moves is $x=\arg\min\,(t_1(x),\ldots,\, t_N(x))$
  - Increase step counter: $\mathrm{step}=\mathrm{step}+1$

- So what are the steps we need to generate the next transition?

Working with lists (1)

- Output of any command or function in Mathematica is always a list!

- Start by creating the list representing the Q-matrix

```
Q={{-2,1,0,1},{2,-4,2,0},{0,1,-1,0},{0,0,1,-1}};
```

  - To create the list, use curly braces - {} !!
  - To evaluate remember to press [SHIFT+RETURN]

Working with lists (2)

- To see the list formatted as a matrix you can type

```
MatrixForm[Q]
```

- List operations:
  - `Q[[i,j]]` : element (i,j) of Q
  - `Q[[i,All]]` : all elements of i:th row in Q
  - `Q[[All,j]]` : all elements of j:th column in Q
  - Other list commands `Append[]`, `Select[]`,….
  - Commands for data arrays: `Total[] (= sum of elements), Mean[], Variance[],…`

- Example: access the first element in the list

```
Q[[1,1]]
```

Observe the need for two brackets [[ ]] !

Generating exponential random variables

- Creating an object representing the exponential distribution

```
ExponentialDistribution[λ]
```

  - λ = intensity parameter (mean = 1/ λ)
- Mathematica supports many other distributions, see a list by typing

```
?*Distribution
```

? = help
* = wildcard

  - It is possible to give a distribution object as an argument to many functions, like `PDF[],CDF[]`…

- Generating a sample from exponential distribution

```
RandomVariate[ExponentialDistribution[λ]]
```

Exponential holding times

- Next we need to generate a list of exponential variables corresponding to the potential holding times in a given state
- Dynamic list is created conveniently by

```
Table[{…},{iterator}]
```

  – Here {…} means any sequence of commands!
- Now, we just need to generate an exponential sample for all strictly positive rates out from a given state

```
state=1;
eventTimes=Table[
  If[Q[[state,i]]>0,
   RandomVariate[ExponentialDistribution[Q[[state,i]]]],
   Infinity]
,{i,1,Length[Q]}];
```

Selecting the next event

- The holding time in the state is then simply selected by

```
timeInState=Min[eventTimes]
```

- Finally, the next state is given by Position-function

```
state=Position[eventTimes,timeInState]
```

- However, to get rid of extra {}-symbols we write

```
state=Position[eventTimes,timeInState][[1,1]]
```

Take first element of output list from Position-command

  – Note that we assume here that states are labelled 1, 2, 3, …!

Next state generation, complete

- Now we have the complete code to generate the next transition from a given state

```
state=1;
eventTimes=Table[
 If[Q[[state,i]]>0,
  RandomVariate[ExponentialDistribution[Q[[state,i]]]],
  Infinity]
,{i,1,Length[Q]}]
timeInState=Min[eventTimes]
state=Position[eventTimes,timeInState][[1,1]]
```

- Copy the above in a cell in your Mathematica notebook and try!

Adding the stopping condition and some statistics collection

- Stopping condition can be implemented with the while-loop

```
While[stopping condition, {…}]
```

Any sequence of commands

- If we want to print some state info we can use Print-command
  - Let's print state just after transition
  - Requires us to keep track of simulation time!

```
Print[{simTime,state}]
```

Putting everything together…

```
state=1;
simuLen=30;
simTime=0;
step=0;
While[simTime<=simuLen,
  eventTimes=Table[
    If[Q[[state,i]]>0,
      RandomVariate[ExponentialDistribution[Q[[state,i]]]],
      Infinity
    ],
    {i,1,Length[Q]}];
  timeInState=Min[eventTimes];
  simTime+=timeInState;
  state=Position[eventTimes,timeInState][[1,1]];
  Print[{simTime,state}];
  step++;
]
```

Creating a function in Mathematica

- To package a block of code you can use the `Module[]` command and define it as a function

```
myFunction[var1_,var2_,…]:=Module[
{list of local variables},
{
  my code here …
}
]
```

Variable name must end with _

Assigns *rhs* to be the delayed value of *lhs*. *rhs* is maintained in an unevaluated form. When *lhs* appears, it is replaced by *rhs*, evaluated afresh each time.

- Let us also save the state just after transition in a list
  - We already keep track of simulation time
  - We append the element {simTime, state} in a list

```
Append[target_list, list_element]
```

Function for Markov process simulator

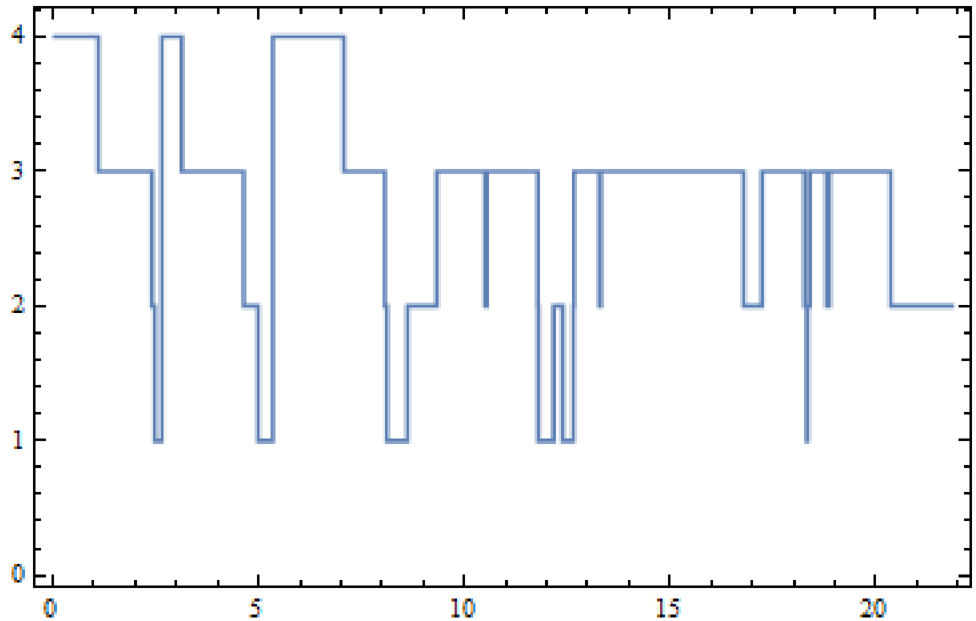
```
simulator[Q_,st_,simuLen_]:=Module[
  {simTime,eventTimes,timeInState,res,state},
  state=st;
  simTime=0;
  res={};
  While[simTime<=simuLen,
    eventTimes=Table[
      If[Q[[state,i]]>0,
        RandomVariate[ExponentialDistribution[Q[[state,i]]]],
        Infinity
      ],
      {i,1,Length[Q]}];
    timeInState=Min[eventTimes];
    simTime+=timeInState;
    state=Position[eventTimes,timeInState][[1,1]];
    res=Append[res,{simTime,state}];
  ];
  res
]
```

Using the simulator

- Run the function and save output in a variable and plot results

```
SeedRandom[101]
res=simulator[Q,1,20]
ListStepPlot[res,Frame->True]
```

- Used to initialize random number generator
- Useful for repetable simulations!



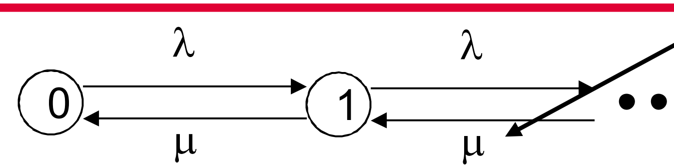
Contents

- Simulation of a simple general Markov process
- Simulation of M/M/1 queue
- Assignment 1

M/M/1queue

- The **M/M/1 queueing system**:
  - packets arrive according to a **Poisson process** (with rate λ)
  - packet lengths are i.i.d. according to the **exponential distribution** with mean L, server processes packets at rate C
  - Thus, service times are exponential with mean $1/\mu = L/C$
  - queuing discipline is **FIFO** , with 1 server and infinite queue size
- This is just a birth death process

$\mu = C/L$



- We observe that in any state there are only two exponential random variables: arrival and departure
  - This is true for any (one-dimensional) BD-process

Generating the transitions (1)

- Possible transitions: departure/arrival
  - Rate parameters are also constant, independent of state
  - Except when system is empty (state 0)

- However, because of the Markovian structure we do not necessarily even need to care about that!
  - For example, even if system is empty we can still generate a potential transition time from the minimum of Exp(λ) and Exp(μ) random variables
  - If the minimum corresponds to departure, Exp(μ), then just ignore that transition and generate a new potential transition until minimum is realized by arrival event
  - This is equivalent with just generating time until next arrival from Exp(λ) distribution

- Possible because of Poisson arrival process (memoryless property of exponential interarrival times)!

Generating the transitions (2)

- So we just need to generate the time until next potential transition

```
eventTimes={RandomVariate[ExponentialDistribution[la]],
  RandomVariate[ExponentialDistribution[mu]]};
timeInState=Min[eventTimes];
```

- To change the state, we take into account the type of the event and whether we are in state 0 or not

```
nextState=If[eventTimes[[1]]<eventTimes[[2]],
  nextState+1,
  Max[nextState-1,0]
];
```

Estimating the mean queue length

- Need to save time since the previous event (`prevEvTime`)

- Given time of new event (`simTime`) and state since previous event, queue length integral is incrementd by

```
meanqlen+=state*(Min[simTime,simuLen]-prevEvTime)
```

Module for M/M/1 queue length

```
simulator3[la_,mu_,st_,simuLen_]:=Module[{simTime,eventTimes,
  timeInState,res,state,meanqlen,prevEvTime},
  state=st;
  simTime=0;
  meanqlen=0;
  While[simTime<=simuLen,
    prevEvTime=simTime;
    eventTimes={RandomVariate[ExponentialDistribution[la]],
                RandomVariate[ExponentialDistribution[mu]]};
    timeInState=Min[eventTimes];
    simTime+=timeInState;
    meanqlen+=state*(Min[simTime,simuLen]-prevEvTime);
    state=If[eventTimes[[1]]<eventTimes[[2]],
             state+1,Max[state-1,0]]
    ];
  ];
  meanqlen/(la*simuLen)
]
```

Mean delay as a function of load (1)

- Use Table-command to iterate over several load values

```
la={0.1,0.3,0.5,0.6,0.7,0.8,0.9};
simures=Table[
   {la[[i]],simulator3[la[[i]],1,1,10000]},
   {i,1,Length[la]}
];
```

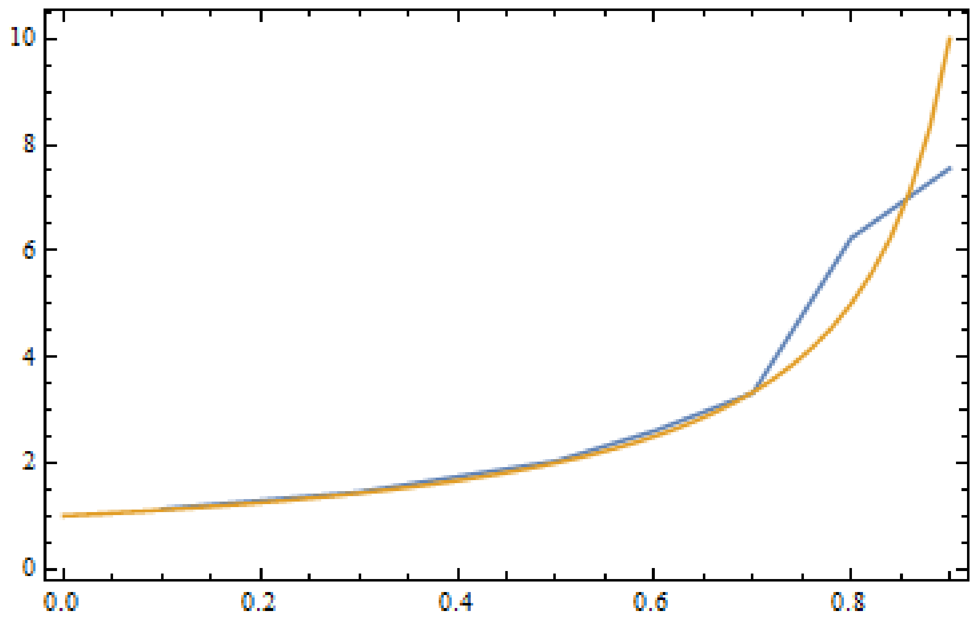
- Calculate the exact value in a vector at discrete points

```
exactres=Table[{la,1/(1-la)},{la,0,0.9,0.02}];
```

Mean delay as a function of load (2)

- Plot the two results in one figure to compare!

```
ListLinePlot[{simures,exactres},Frame->True]
```



Contents

- Simulation of a simple general Markov process
- Simulation of M/M/1 queue
- Assignment 1