



Elements of System Dynamics

Antoine B. Rauzy

PART 3. THE WORLDLAB™ WORKSHOP

LECTURE 5. EXPERIMENTS

LECTURE 5. EXPERIMENTS

Key notions:

- Experiments
- Synecdoche
- Monte-Carlo Simulation
- Random variables, Estimators
- Σ^{TM} Player, Σ^{TM} Calculator, Σ^{TM} Analyst

Learning Objectives

In the WorldLab™ framework, any simulation, calculation or analysis is called an **experiment**. This is not only a matter of wording, although the reference to the notion of **digital laboratory** is important. But more importantly, the WorldLab™ Workshop makes it possible to **record** experiments, so to be able to **replay** them.

The two most important types of experiments are **interactive** and **stochastic simulations** of models.

This lecture has two main objectives.

- To present how the notion of experiment is implemented in the WorldLab™ Workshop.
- To discuss interactive and stochastic simulation in depth.

Agenda

Section 1. Case Study

Section 2. Synecdoche

Section 3. Use Cases

Section 4. Back to the Case Study

Section 5. Monte-Carlo Simulation

Section 6. Statistics

Section 7. Application to the Case Study

Section 8. Wrap-Up

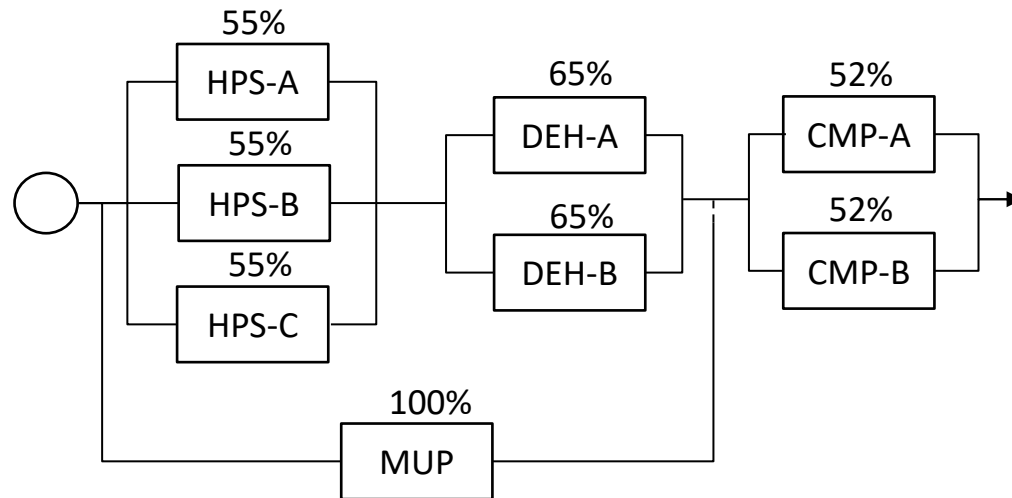
LECTURE 5. EXPERIMENTS

SECTION 1. CASE STUDY

Gas Production System

The system is a production facility consisting of height units. Gas separated from the well fluid at upstream side is fed to the facility, treated through separators (HPS-A,B,C) and dehydrators (DEH-A,B), and led to compressors (CMP-A,B).

The make-up compressor (MUP) is installed to enable the CMP-A and B to discharge gas with full flow rate even if some of gas treatment units (HPS or DEH) are failed. It is assumed that the MUP is equipped with gas treatment units, which are dedicated to the MUP.



The maximum production capacities of each unit is described on the diagram.

Reliability Data

- Failures and repairs of components are assumed to be statistically independent.
- Failures are described by Weibull distributions.
- Repairs by uniform distributions.
- Components are as good as new after a repair.

Component	Failure: Weibull distribution		Repair: Uniform distribution	
	Scale factor (h)	Shape factor	Lower bound (d)	Upper bound (d)
HPS	1.12e4	3	15	20
DEH	3.21e4	3	8	12
CMP	2.86e4	3	6	10
MUP	1.23e4	5	14	21

We want to assess:

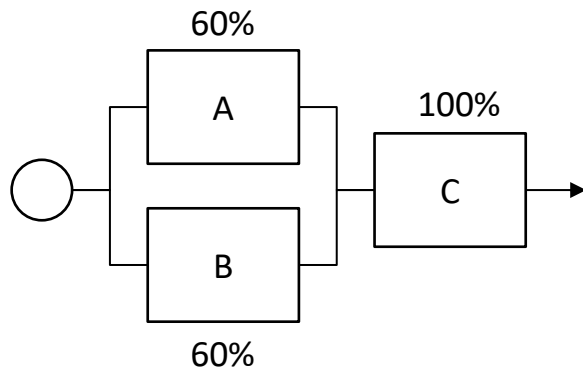
- The probability that the production stops completely over a 10 years (87600 hours) period.
- The average production over the same period.

LECTURE 5. EXPERIMENTS

SECTION 3. SYNECDOCHE

Synecdoche*

We shall start by designing a deterministic model representing a simplified system. It is often a good approach to clear the ground by **reducing the problem** at stake both in terms of size and of complexity.



	Failure: Weibull distribution		Repair: Uniform distribution	
Unit	Scale factor (h)	Shape factor	Lower bound (d)	Upper bound (d)
A, B, C	1.0e4	3	15	20

(*) A synecdoche is a figure of speech in which a part is made to represent the whole (it is a special form of metonymy). e.g.

- "All hands on deck!" meaning all sailors of ship's deck.
- The character of Madame Bovary (Flaubert) represents the provincial bourgeois life

ProductionPlant1



Units (1)

```
domain Mode {OPERATION, REPAIR}
```

```
class Unit
```

```
  parameter float operationTime(unit = "h") = 10000;
```

```
  parameter float repairTime(unit = "h") = 1000;
```

```
  Mode mode(init = OPERATION);
```

```
  bool working(init = false);
```

```
  parameter float capacity = 100;
```

```
  port float inCapacity = 0;
```

```
  port float outCapacity =
```

```
    if mode==OPERATION and working
```

```
    then min(capacity, inCapacity)
```

```
    else 0;
```

```
end
```

Constant for now

Propagation of the maximum
capacity

Units (2)

```
activity Unit.Operate
```

```
  trigger:
```

```
    return not working and mode==OPERATION;
```

```
  start:
```

```
    working = true;
```

```
  completion: {
```

```
    mode = REPAIR;
```

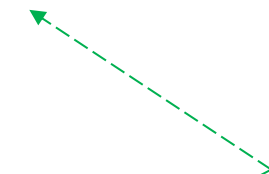
```
    working = false;
```

```
  }
```

```
  duration:
```

```
    return operationTime;
```

```
end
```



Alternation of operation and repair

```
activity Unit.Repair
```

```
  trigger:
```

```
    return not working and mode==REPAIR;
```

```
  start:
```

```
    working = true;
```

```
  completion: {
```

```
    mode = OPERATION;
```

```
    working = false;
```

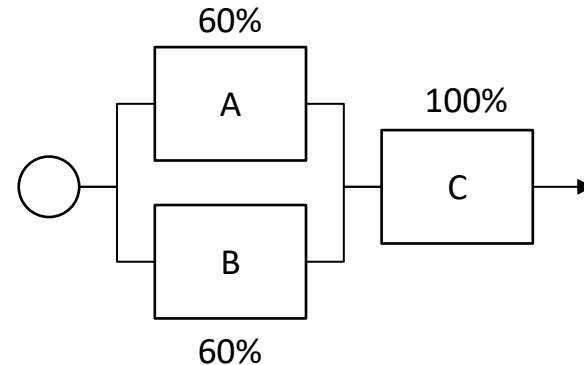
```
  }
```

```
  duration:
```

```
    return repairTime;
```

```
end
```

Units (3)



```
system ProductionPlant
  Unit A;
  Unit B;
  Unit C;
  parameter float A.capacity = 60;
  parameter float B.capacity = 60;
  port float A.inCapacity = 100;
  port float B.inCapacity = 100;
  port float C.inCapacity = min(A.outCapacity + B.outCapacity, 100);
  observer production = C.outCapacity;
  observer shutdown = if C.outCapacity==0 then 1 else 0;
end
```

Key performance indicators to be assessed.

LECTURE 5. EXPERIMENTS

SECTION 4. USE CASES

Rational

Use cases have two main objectives:

- **Validate** the objectives and the construction of the systemic digital twin with stakeholders.
- **Check** (debug) the model.

Problem:

- The model is **non-deterministic** as the duration of activities Operate and Repair are. This is the case for most of the models of complex technical and socio-technical systems.
- Setting parameter values is not fully satisfying.

Solution:

- Use **empirical series** to encode stochastic quantities.

Empirical Series

Empirical series are functions from positive integers to constants (of any type). However, no argument is provided. Rather, the first time the empirical series S is called, $S(1)$ is returned, the second time $S(2)$ is returned and so on.

Empirical series are stored into CSV files

1	1.01
2	2.01
3	3.01
5	4.01
8	5.01
13	6.01

$S(1) = 1.01$

$S(2) = 2.01$

$S(3) = 3.01$

$S(4) = 3.01$

$S(5) = 4.01$

$S(6) = 4.01$

...

$S(13) = 6.01$

$S(14) = \text{nil}$

```
empiricalSeries("Data/Fibonacci.csv")
```

Units (4)

```
domain Mode {OPERATION, REPAIR}

class Unit
  parameter str failureTimesFilePath = "failureTimes.csv";
  parameter str repairTimesFilePath = "repairTimes.csv";

  Mode mode(init = OPERATION);
  bool working(init = false);

  parameter float capacity = 100;
  port float inCapacity = 0;
  port float outCapacity =
    if mode==OPERATION and working
    then min(capacity, inCapacity)
    else 0;
end
```

Will be used to set
empirical studies of
components

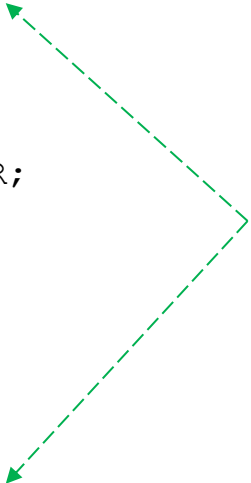
ProductionPlant2



Units (5)

```
activity Unit.Operate
  trigger:
    return not working and mode==OPERATION;
  start:
    working = true;
  completion: {
    mode = REPAIR;
    working = false;
  }
  duration:
    return empiricalSeries(failureTimesFilePath);
end
```

```
activity Unit.Repair
  trigger:
    return not working and mode==REPAIR;
  start:
    working = true;
  completion: {
    mode = OPERATION;
    working = false;
  }
  duration:
    return empiricalSeries(repairTimesFilePath);
end
```



Replacing stochastic quantities by empirical series

Failure then Repair of Unit A

Preconditions: The three units A, B and C are working properly.

Postconditions: The three units A, B and C are working properly.

Trigger: Failure of the Unit A

Story:

1. The three units A, B and C are started at time 0.
2. Unit A fails at time 5000 i.e. its activity Operate is completed at this time.
3. The production that was at 100% drops to 60%.
4. The repair of Unit A is started at this date.
5. The repair of Unit A is completed at time $5000 + 600 = 5600$
6. The operation of Unit A is resumed at this date.
7. Meanwhile Units B and C continue to work correctly.
8. Units A, B and C go on working correctly until the mission time, i.e. time 87600.

Application to the Case Study

Mission time: 87600

Failure Times

A

1	5000
*	100000

B

*	100000
---	--------

C

*	100000
---	--------

Repair Times

A

1	600
*	100000

B

*	100000
---	--------

C

*	100000
---	--------

Infinity

Can be stored in a single CSV file

Experiment

Create a parameter file

The screenshot shows the 'Sigma Player' configuration window for a simulation named 'new* - ProductionPlant'. The window has a menu bar with 'File', 'Simulation', 'Window', and 'Help'. The 'Configuration' tab is active, showing several sections:

- Interface file:** A dropdown menu set to '[Default Interface]' with a 'Browse...' button. Below it are five checked checkboxes: 'Parameters', 'Main System', 'Observers', 'Schedule', and 'History'.
- Sigma parameter file:** A text field containing 'Optional', with 'New', 'Edit', and 'Browse...' buttons to its right. A green dashed arrow points from the text 'Create a parameter file' to the 'New' button.
- Simulation settings:** Contains two fields: 'Mission time:' set to '87600' and 'Random generator seed:' set to '12345'. A green dashed arrow points from the text 'Setting the mission time' to the 'Mission time' field.
- Observers time series file:** A text field containing 'Optional', with 'Visualize' and 'Browse...' buttons to its right.
- Video settings:** Contains two fields: 'Time step:' set to '1.00' and 'Steps per second:' set to '24'.

A 'Launch' button is located at the bottom right of the configuration panel.

Setting the mission time

Parameter File

	1	2
1	#projectName ProductionPlant	
2	#timestamp 2024-12-29T09:59:59Z	
3	<i>ProductionPlant.A.capacity</i>	60
4	<i>ProductionPlant.A.failureTimesFilePath</i>	<i>Data/failureTimesA1.csv</i>
5	<i>ProductionPlant.A.repairTimesFilePath</i>	<i>Data/repairTimesA1.csv</i>
6	<i>ProductionPlant.B.capacity</i>	60
7	<i>ProductionPlant.B.failureTimesFilePath</i>	<i>Data/beyondMissionTime.csv</i>
8	<i>ProductionPlant.B.repairTimesFilePath</i>	<i>Data/beyondMissionTime.csv</i>
9	<i>ProductionPlant.C.capacity</i>	100
10	<i>ProductionPlant.C.failureTimesFilePath</i>	<i>Data/beyondMissionTime.csv</i>
11	<i>ProductionPlant.C.repairTimesFilePath</i>	<i>Data/beyondMissionTime.csv</i>

Failures of Unit A and C

Preconditions: The three units A, B and C are working properly.

Postconditions: The three units A, B and C are working properly.

Trigger: Failure of the Unit A, then of Unit C

Story:

1. The three units A, B and C are started at time 0.
2. Unit A fails at time 5000 i.e. its activity Operate is completed at this time.
3. The production that was at 100% drops to 60%.
4. The repair of Unit A is started at this date.
5. Unit C fails at time 5500.
6. The production that was at drops from 60% to 0% (shutdown).
7. The repair of Unit C starts immediately.
8. The repair of Unit C is completed at time $5500 + 700 = 6200$.
9. The production is thus resumed at 60%.
10. The repair of the Unit A is completed at time $5000 + 1300 = 6300$.
11. The operation of Unit A is resumed at this date.
12. The production is thus resumed at 100%
13. Meanwhile Unit B continues to work correctly.
14. Units A, B and C go on working correctly until the mission time, i.e. time 87600.

Application to the Case Study

Mission time: 87600

Failure Times

A

1	5000
*	100000

B

*	100000
---	--------

C

1	5500
*	100000

Repair Times

A

1	1300
*	100000

B

*	100000
---	--------

C

1	700
*	100000

LECTURE 5. EXPERIMENTS

SECTION 4. BACK TO THE CASE STUDY

Units

```
class HighPressureSeparator
  extends Unit;
  parameter float capacity = 55;
end
```

← Same as before

```
class Dehydrator
  extends Unit;
  parameter float capacity = 65;
end
```

```
class Compressor
  extends Unit;
  parameter float capacity = 52;
end
```

```
class MakeUpCompressor
  extends Unit;
  parameter float capacity = 100;
end
```

GasProductionSystem1



System Hierarchy

```
system GasProductionSystem
  system HPS
    HighPressureSeparator A;
    HighPressureSeparator B;
    HighPressureSeparator C;
    port float inCapacity = 0;
    port float A.inCapacity = inCapacity;
    port float B.inCapacity = inCapacity;
    port float C.inCapacity = inCapacity;
    port float outCapacity =
      min(A.outCapacity + B.outCapacity + C.outCapacity, 100);
  end
  system DEH
    Dehydrator A;
    Dehydrator B;
    port float inCapacity = 0;
    port float A.inCapacity = inCapacity;
    port float B.inCapacity = inCapacity;
    port float outCapacity =
      min(A.outCapacity + B.outCapacity, 100);
  end
  ...
```

System Connections

```
system CMP
  Compressor A;
  Compressor B;
  port float inCapacity = 0;
  port float A.inCapacity = inCapacity;
  port float B.inCapacity = inCapacity;
  port float outCapacity =
    min(A.outCapacity + B.outCapacity, 100);
end
MakeUpCompressor MUP;

port float HPS.inCapacity = 100;
port float DEH.inCapacity = HPS.outCapacity;
port float MUP.inCapacity = 100;
port float CMP.inCapacity =
  max(DEH.outCapacity, MUP.outCapacity);

observer production = CMP.outCapacity;
observer shutdown = if CMP.outCapacity==0 then 1 else 0;
end
```

Failure then Repair of Unit CMP.B

Preconditions: All units are working properly.

Postconditions: All units are working properly.

Trigger: Failure of the Unit CMP.B

Story:

1. All units are started at time 0.
2. Unit CMP.B fails at time 5000 i.e. its activity Operate is completed at this time.
3. The production that was at 100% drops to 52%.
4. The repair of Unit CMP.B is started at this date.
5. The repair of Unit CMP.B is completed at time $5000 + 600 = 5600$
6. The operation of Unit CMP.B is resumed at this date.
7. Meanwhile all other units continue to work correctly.
8. All units go on working correctly until the mission time, i.e. time 87600.

Collective Exercise

Design a few use cases that you think important to check the model.

Set the parameters to play these use cases.

Save the use cases as experiments.

Final Model: Base Units

```
class Unit
  parameter float failureScaleFactor(unit = "h") = 1.0e4;
  parameter float failureShapeFactor = 3;
  parameter int repairLowDuration(unit = "d") = 10;
  parameter int repairHighDuration(unit = "d") = 20;
  Mode mode(init = OPERATION);
  bool working(init = false);
  parameter float capacity = 100;
  port float inCapacity = 0;
  port float outCapacity = if mode==OPERATION and working then
    min(capacity, inCapacity) else 0;
end

activity Unit.Operate
  ...
  duration:
    return WeibullDeviate(failureScaleFactor, failureShapeFactor);
end

activity Unit.Repair
  ...
  duration:
    return rangeDeviate(repairLowDuration, repairHighDuration)*24;
end
```

GasProductionSystem2



Final Model: Actual Units

```
class HighPressureSeparator
  extends Unit;
  parameter float capacity = 55;
  parameter float failureScaleFactor(unit = "h") = 1.12e4;
  parameter float failureShapeFactor = 3;
  parameter int repairLowDuration(unit = "d") = 15;
  parameter int repairHighDuration(unit = "d") = 20;
end
...
```

The other parts of the model are left unchanged...

We can now perform a few interactive simulations.

LECTURE 5. EXPERIMENTS

SECTION 5. MONTE-CARLO SIMULATION

History

The modern version of the **Monte-Carlo method** was invented in the late 1940s by Stanislaw Ulam, while he was working on nuclear weapons projects at the Los Alamos National Laboratory. Immediately after Ulam's breakthrough, John von Neumann understood its importance and programmed the ENIAC computer to carry out Monte-Carlo calculations.

Being secret, the work of von Neumann and Ulam required a code name. A colleague of von Neumann and Ulam, Nicholas Metropolis, suggested using the name Monte Carlo, which refers to the Monte-Carlo Casino in Monaco where Ulam's uncle would borrow money from relatives to gamble.

Algorithm

The generic **Monte-Carlo simulation algorithm** is as follows.

```
StochasticSimulation ( numberOfExecutions ) :  
    InitializeModel()  
    for t in 1.. numberOfExecutions :  
        AssessModel()  
        UpdateIndicators()  
    MakeStatistics()
```

It is thus quite simple to implement. However, you must take care at:

- Selecting the right **probability distributions** and the right **performance indicators**;
- Performing **sufficiently many tries** (a **sufficiently large sample size**) regarding the probability of the **observed phenomenon**;
- Selecting a good **pseudo-random number generator**;
- To verify that the **model** meets the **reality**.

Examples of uses:

In physics:

- Analysis of neutron & photon transport
- Design of new materials and structures, such as organic LEDs, organic solar cells and Lithium-Ion batteries

In astrophysics:

- Models of evolution of galaxies

In engineering:

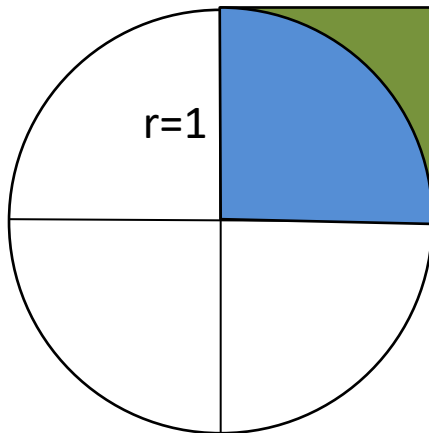
- Inventory processes, job scheduling, vehicle routing, queueing networks, system reliability...

In finance:

- Pricing of financial instruments, risk analysis.

Estimation of $\pi = 3,141\ 592\ 653\ 589\ 793$

In theory, Monte-Carlo simulation can be used to estimate π .



Idea: draw a large number N of pairs (x, y) uniformly in $[0, 1] \times [0, 1]$ and count the number K of pairs such that $x^2 + y^2 \leq 1$.
Then, $\pi \approx 4.K/N$

In practice however, only the first decimal of π can be reached.

N	K	π
1 000 000	784 802	3.139208
10 000 000	7 852 310	3.140924
50 000 000	39 264 640	3.141171
100 000 000	78 532 441	3.141298

... but it may used to test your random number generator...

Integral calculation

Assume we have to calculate a complex integral in the form:

$$\int_a^b f(x)dx$$

The value of this integral can be calculated as $(b-a)$ times the mean value of the function f in the interval $[a, b]$.

We can estimate this mean value by means of a **Monte-Carlo integration**: we draw a random n points in the range $[a, b]$ and calculate the value of f in each of this point and then calculate their empirical mean.

This method works in any dimension. The convergence speed is the same, whatever the number of dimensions, namely $\frac{1}{\sqrt{n}}$.

This method has been applied for the first time during the Manhattan project. It is notably used for the assessment of option prices in finance (« call » and « put »).

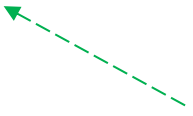
Error Function

$$\text{erf}(z) \stackrel{\text{def}}{=} \frac{2}{\sqrt{\pi}} \int_0^z e^{-x^2} dx$$

- Useful, for example, in determining the bit error rate in digital communication systems.
- Does not have **closed form solution**.

```
system ErrorFunction
  parameter float z = 3;
  float x, v(init = 0);
  bool calculating(init = false);
  activity Calculate
    trigger:
      return not calculating;
    start:
      calculating = true;
    completion: {
      x = uniformDeviate(0.0, z);
      v = (2/sqrt(pi)) * exp(-x*x) * z;
    }
    duration:
      return 1;
  end
end
```

Need to calculate only once



Observers

$$\operatorname{erf}(z) \stackrel{\text{def}}{=} \frac{2}{\sqrt{\pi}} \int_0^z e^{-x^2} dx$$

- Useful, for example, in determining the bit error rate in digital communication systems.
- Does not have **closed form solution**.

```
system ErrorFunction
  observer erf = v;
  indicator ERF = value(erf);
end
```

ErrorFunction



Observers monitor a value throughout the mission time.

Temporal formulas make it possible to calculate over the mission time:

- Sum of values (integral);
- Mean value;
- Minimum and maximum values;
- First change time;
- Number of changes

Monte-Carlo simulations make statistics over executions on the quantities calculated by observers.

- Mean, standard deviation
- Confidence ranges
- Quantiles, distributions

Launching Monte-Carlo Simulation

The screenshot shows a software window titled "new* - ErrorFunction - Sigma Calculator" with a menu bar (File, Window, Help) and a "Configuration" tab. The interface is divided into several sections:

- Sigma parameter files:** A large empty text area labeled "Parameter file(s)" in green. To its right are buttons for "New", "Browse...", "Edit", and "Remove".
- Simulation settings:**
 - Mission time:** A spinner box set to 1000.00, with a green annotation "Times at which statistics are made" pointing to it.
 - Observation dates:** Includes "First date:" (0.00), "Last date:" (0.00), and "Step:" (0.00), each with a spinner box.
 - Additional observation dates:** A text box with the instruction "(separated with semicolons)".
 - Number of executions:** A spinner box set to 10000, with a green annotation "Number of executions" pointing to it.
 - Random generator seed:** A spinner box set to 12345.
- Results file:** A text box containing "StochasticSimulation/statistics.csv" with a close button (X) and "Visualize" and "Browse..." buttons.
- Observations file:** A text box containing "Optional" with "Visualize" and "Browse..." buttons.

A green annotation "Result file" is placed between the "Results file" and "Observations file" sections. A "Launch" button is located at the bottom right of the window.

Result File

new* - ErrorFunction - Sigma Calculator

File

Window

Help

Results

statistics.csv

Parameters	[Default Parameters]						
Sigma stochastic simulation							
System	ErrorFunction						
Date	2025-01-03 09:10:36						
Executions	100000						
Indicator	ErrorFunction.ERF						
Observer	ErrorFunction.erf						
Type	value						
Time	SampleSize	Mean	StandardDeviation	Minimum	Maximum	ConfidenceRange...	ConfidenceRange...
1000	100000	1.00023	1.17962	0.000417803	3.38514	0.992766	1.00769

Estimated value of the Error Function

Back

Exit

6. EXPERIMENTS

SECTION 6. STATISTICS

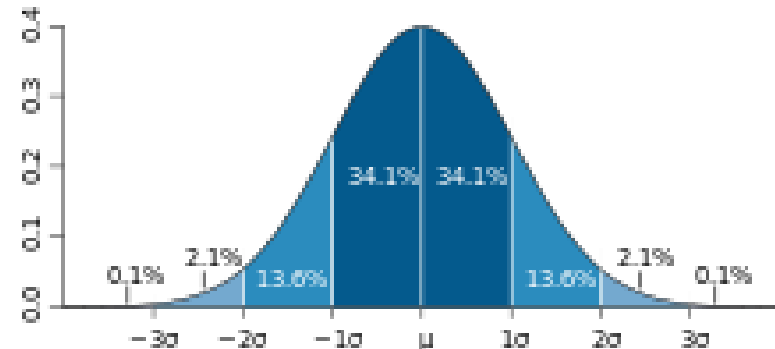
Estimators

Three **estimators** are in general calculated for each indicator: its mean, its standard-deviation and a confidence range (in general 95% confidence range).

- The **mean**: $\mu_X = \frac{\sum_n x_i}{n}$
- The **standard deviation** is a measure of dispersion of data (around the mean).

Formal definition: $\sigma_X = \sqrt{E[X^2] - E[X]^2}$

Usual calculation method: Welford algorithm.



- The p% **confidence range** is the interval centered on the mean such that the actual value of the indicator has a probability $p(/100)$ to be in the interval. For $p=95$, its definition is as follows:

$$I_{95\%} = \left[\mu_X - 1.96 \frac{\sigma_X}{\sqrt{n}}, \mu_X + 1.96 \frac{\sigma_X}{\sqrt{n}} \right]$$

Tables give the values of the parameter (here 1.96) for different values of p .

The WorldLab Workshop makes it possible to assess σ , 2σ , and 3σ confidence ranges

Quantiles

The standard-deviation and the confidence range(s) are really of interest only if the indicator is distributed according to a normal law. In practice, this assumption is seldom verified. To get more insights about indicators, we can estimate their **distributions** and their **quantiles**.

The principle is to sort the observed values and to organize them into **bins** of same size. For each bin, it is then possible to calculate a minimum, a mean and a maximum value. The maximum value is the **quantile**. The value M such that 50% of observed values are smaller than M and 50% bigger is called the **median**.

	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
minimum	2	2	4	10	32	75	127	191	278	388
mean	2.00	2.89	6.71	19.34	52.01	99.53	155.09	230.57	327.85	525.90
maximum	2	4	10	32	74	127	191	278	386	880

30% of the executions terminated in less than 10 steps

40% of the executions terminated in less than 32 steps

median

(*) In 2011, the mean of monthly salaries in France was 2172€, while the median was 1712€

Size of samples

It is well known that is not necessary to consider too large **samples** (number of executions). This is justified by the definition of confidence ranges:

$$I_\alpha = \left] \mu_X - t_\alpha \frac{\sigma_X}{\sqrt{n}}, \mu_X + t_\alpha \frac{\sigma_X}{\sqrt{n}} \right[$$

This definition says that to reduce the uncertainty by a factor k , one must increase the sample size by a factor k^2 . However,

- 1) This applies only in case where the indicator is distributed according to a normal law;
- 2) This applies only if the sample size is big enough.

In practice, if we want to observe a phenomenon that has 1 chance out of 1 million to occur, we need to draw at least 100 millions runs to get some valuable information.

This high number of runs is a limiting factor of the Monte-Carlo simulation.

Here is a general rule of experimental sciences: to observe a phenomenon, we need to have a *a priori* idea of its **nature** and its **amplitude**.

Random number generators

Stochastic simulation relies on the generation of sequences of (pseudo-)random numbers. From a mathematical standpoint, a random sequence is an infinite sequence of symbols of an alphabet (typically $\{0, 1\}$) that has no structure, regularity or identifiable prediction rule. It is actually quite difficult to give a formal definition of random sequences, and several alternative definitions have been proposed.

In practice, the problem is to find **pseudo-random number generator** that “mimics” as much as possible the “real” randomness. There exists two main categories of generators:

- Generators relying on physical devices. They have the double drawback of being hardly testable and non reproducible.
- Algorithmic generators are used most of the time, possibly by initializing them with physical devices (like the computer clock).

Pseudo-random number generators give sequences of integers uniformly distributed in the range $[0, 2^{31}-1]$ (on 32 bits machines) and $[0, 2^{63}-1]$ (on 64 bits machines). To obtain a floating-point number uniformly distributed between 0.0 and 1.0, it suffices to divide the generated number by $2^{31}-1$ (or $2^{63}-1$).

Mersenne-Twister generator

The **Mersenne-Twister** generator is known for its quality.

- It has a $2^{19937}-1$ period;
- It is uniformly distributed even in a large number of dimensions (623 for 32 number bits) ;
- It is faster than most of the “good” generators;
- It passes the most advanced statistical tests on each bit.

It is however too complex to be described here (those who are interested can have a look to: <http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html>).

That's the generator used in Python, Matlab, Ruby, R, gnu...

It is used for many application, including cryptographic applications.

LECTURE 5. EXPERIMENTS

SECTION 7. APPLICATION TO THE CASE STUDY

Observers and Indicators

```
system GasProductionSystem
  port float production = CMP.outCapacity;
  port float shutdown = if CMP.outCapacity==0 then 1 else 0;

  observer meanProduction(bins = 10) = timedMean(production);
  observer totalProduction = timedSum(production);
  observer shutdownProbability = shutdown;
  observer shutdownTime = timedSum(shutdown);
  observer meanTimeToShutdown = firstChangeTime(shutdown);
end
```

Experiment 1:

- Mission time: 87600
- Number of executions: 100,000

GasProductionSystem2



Distributions

It is sometimes of interest to know not only default statistical estimators, but also the **distribution** of an indicator.

```
system GasProductionSystem
  port float production = CMP.outCapacity;
  port float shutdown = if CMP.outCapacity==0 then 1 else 0;

  observer meanProduction(bins = 10) = timedMean(production);
  // observer totalProduction = timedSum(production);
  // observer shutdownProbability = shutdown;
  // observer shutdownTime = timedSum(shutdown);
  // observer meanTimeToShutdown = firstChangeTime(shutdown);
end
```

Experiment 2:

- Mission time: 87600
- Number of executions: 100,000

GasProductionSystem2



Time Series

It is sometimes of interest to follow the evolution through the time of an estimator, hence getting a **time series** for this indicator.

```
system GasProductionSystem
  port float production = CMP.outCapacity;
  port float shutdown = if CMP.outCapacity==0 then 1 else 0;

  // observer meanProduction(bins = 10) = timedMean(production);
  // observer totalProduction = timedSum(production);
  observer shutdownProbability = shutdown;
  // observer shutdownTime = timedSum(shutdown);
  // observer meanTimeToShutdown = firstChangeTime(shutdown);
end
```

Experiment 3:

- Mission times: 0, 1000, 2000,... 87600
- Number of executions: 10,000,000

GasProductionSystem2



The expected probability is around $4.0e-5$!

LECTURE 5. EXPERIMENTS

SECTION 8. WRAP-UP

Wrap-Up

- **Synecdoche** (starting by modeling a part of the problem at stake) is a good mean to clear the ground when designing a systemic digital twin.
- In the Σ^{TM} modeling framework, the term generic **experiment** covers all automated operations performed to create and to simulate the model. The WordLabTM Workshop makes it possible to **save** and to **reload** experiments.
- **Interactive simulations** can be used to implement **use cases**, by setting the values of **parameters** and, more importantly, by using **empirical series**. They can be saved as experiments.
- **Stochastic simulation** is the Swiss knife of systems engineering. It can be used to estimate a wide variety of **key performance indicators**. Stochastic simulations are implemented using **observers** and **indicators**.

When to use Monte-Carlo simulation?

Monte-Carlo simulation has several advantages over other methods:

- It is much **cheaper** to realize than any physical experiment. Moreover, it is **easy to reproduce**;
- It makes it possible to study phenomena for which **no analytical solution** exists or for which such a solution is much too hard to establish;
- It is an “**anytime**” algorithm: the more you draw runs, the better the result, but you can stop the simulation any time.

However, Monte-Carlo simulation is not a universal panacea:

- It may be very (too) **calculation resource consuming**;
- It gives only **approximated solutions**;
- But more fundamentally, it is often extremely difficult to **check models**;