Counterparty risk assessment with 2-step Montecarlo and Parallel Computing

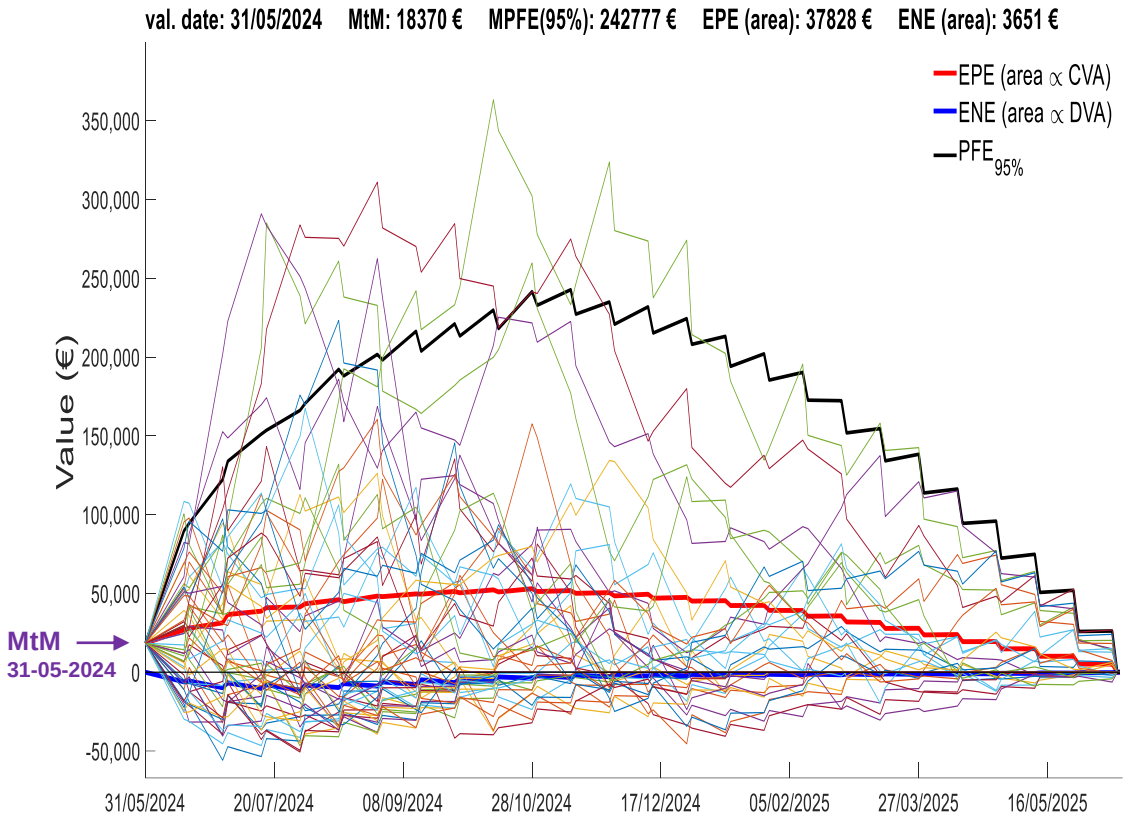
Pablo García Estébanez

Banco Sabadell

Index

1. Exposure and risk metrics

In its broadest sense, exposure is defined as the present (known) or future (uncertain) value of a position that presents a risk of default.



Exposure	Counterparty risk metric
Expected Positive Exposure $\mathrm{EPE(t)} = \mathrm{E}[\ \max(\ \mathrm{V(t)}\ ;0)\ ]$	**Expected loss by counterparty credit risk** $\mathrm{CVA} = -\int \mathbf{EPE(t)} \cdot LGD_1(\mathrm{t}) \cdot DF(t,t_0) \cdot d\mathrm{PD}_2$
Expected Negative Exposure $\mathrm{ENE(t)} = -\mathrm{E}[\ \min(\ \mathrm{V(t)}\ ;0)\ ]$	**Expected profit by your own credit risk** $\mathrm{DVA} = \int \mathbf{ENE(t)} \cdot LGD_2(\mathrm{t}) \cdot DF(t,t_0) \cdot d\mathrm{PD}_1$
Potential Future Exposure $\mathrm{PFE}_{95\%}(\mathrm{t}) = \mathrm{Percentile}_{95}\{\ \max(0,\mathrm{V(t)})\ \}$	**Maximum loss in the event of counterparty default** $\mathrm{MPFE} = \max_{\mathrm{t}} \left(\mathbf{PFE_{95\%}(t)}\right)$

PD: Probability of Default
LGD: Loss Given Default
DF: Discount factor

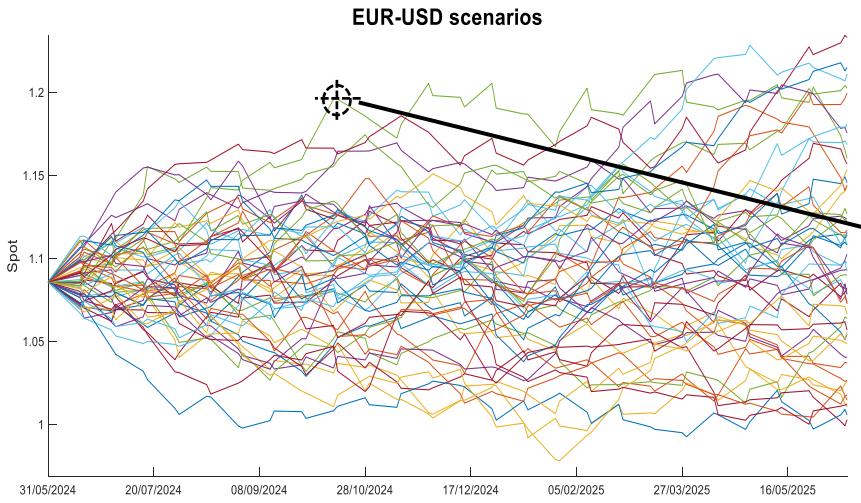
2. Computational framework (I). The main issue

Most exotic derivatives don't have a closed valuation formula, and even when they do, multiple scenarios and netting times need to be considered.

a) Mark-to-Market: based on current market inputs. It requires a limited computational effort that can be handled with standard tools.

b) Valuating in simulated future scenarios: For every deal and every "future time", it is necessary to perform thousands of valuations, and the computational cost becomes unfeasible with conventional architecture/tools.

b1. Future N_1 scenario generation. Stochastic model with market input (CVA/DVA) or historical data (MPFE)



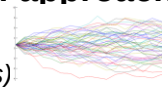
b2. Valuation in each scenario

2.1 Closed formula

(semi-analytical approach)

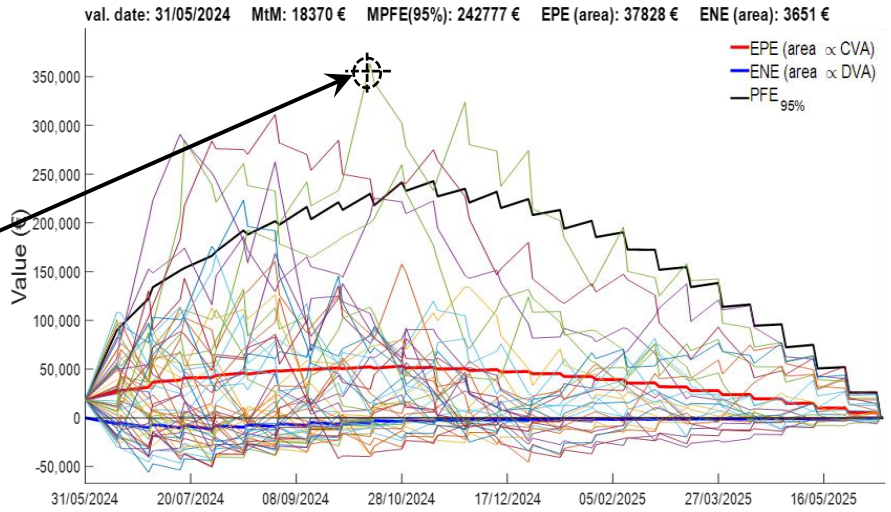
2.2 numerical approach

- *Montecarlo (N_2 samples)*



- Partial differential equation (PDE)
- *Trees*
- ….

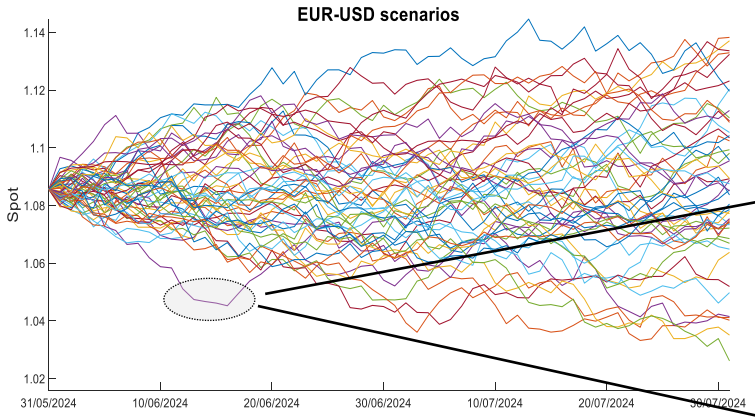
b3. Exposure calculation



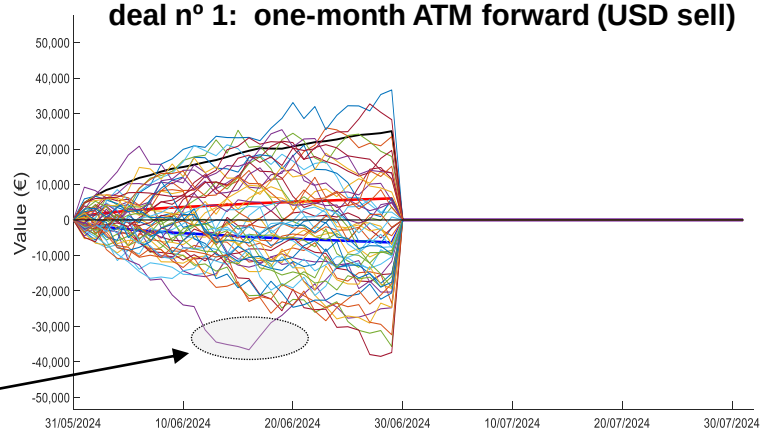
2. Computational framework (II). Netting

1. Generation of risk factor scenarios → **2. Valuation of each deal in each future scenario** → **3. Netting** (scenario by scenario) **& Exposure calculation**

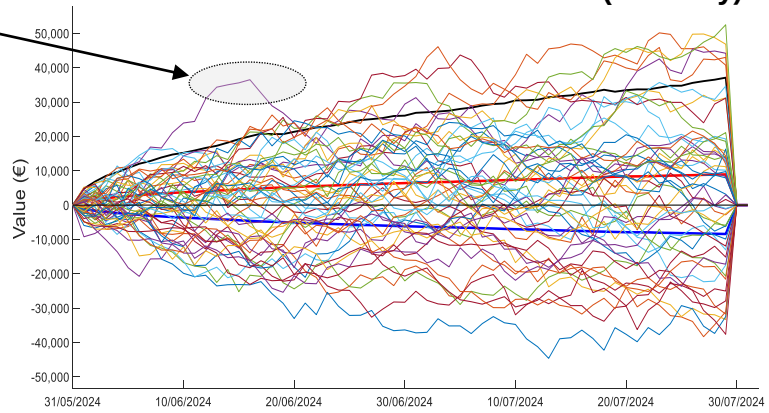
Netting of the two deals



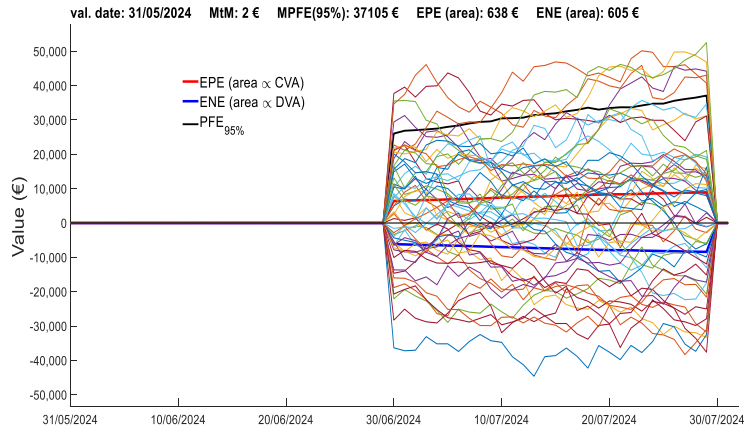
deal nº 1: one-month ATM forward (USD sell)



deal nº 2: two-months ATM forward (USD buy)

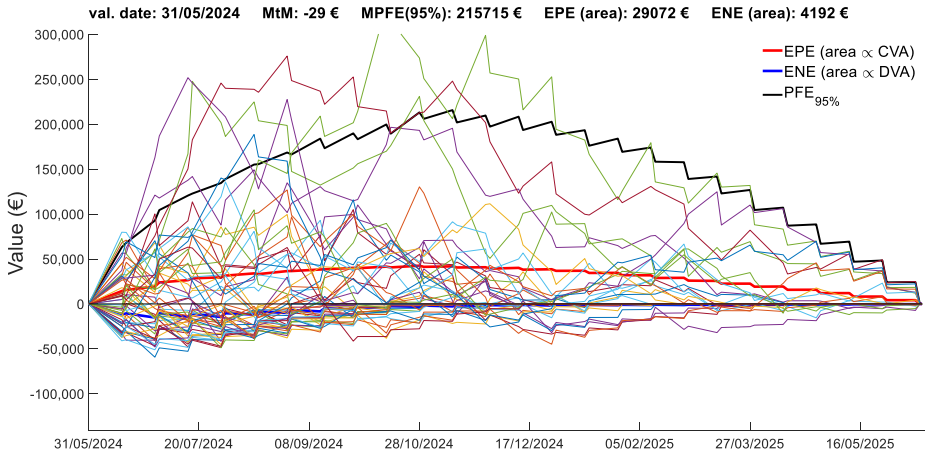


netting of the two deals

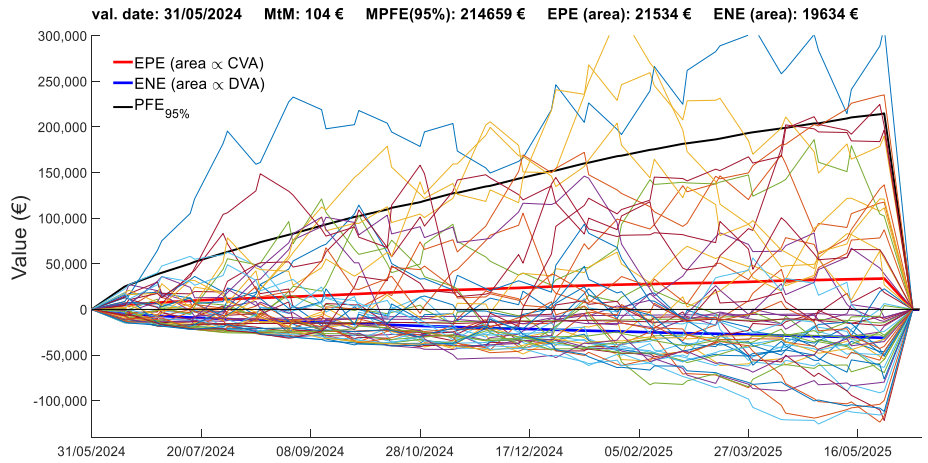


2. Computational framework (II). Real netting example

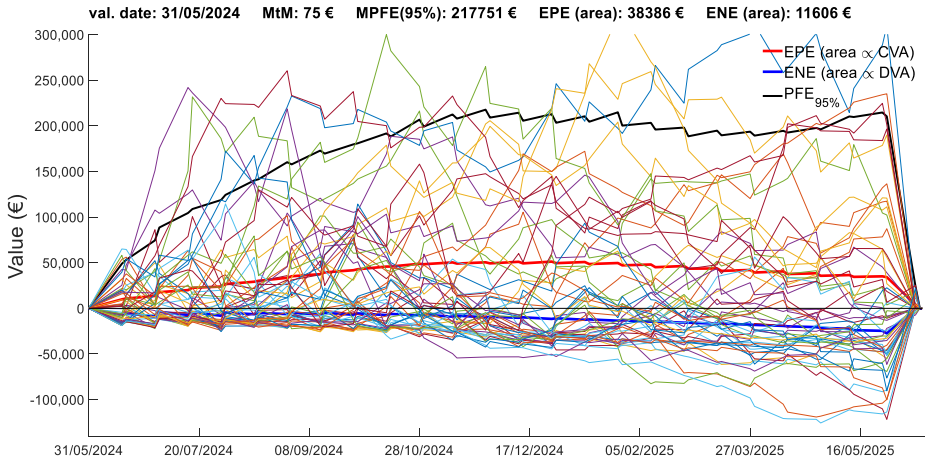
deal nº 1 : USD-sell accrual TARF



deal nº 2 : USD-buy Accumulator



netting of the two deals



Exposure (k€)	deal nº 1	deal nº2	netting
MPFE	216	215	218
EPE area	29	22	38
ENE area	4	20	12

2. Computational framework (III). Complexity with a single deal

- **2-step Montecarlo valuation** — first step: generation of $\mathbf{N_1}$ scenarios.
  second step : in each scenario, Montecarlo valuation with $\mathbf{N_2}$ simulations.
- **Semi-analytical valuation** — first step: generation of $\mathbf{N_1}$ scenarios.
  second step : in each scenario, closed formula valuation.

Computational complexity (c.c.)

2-step Montecarlo: $\boldsymbol{c.c.} \propto N_1 \text{ x } N_2 \text{ x } N_{\text{deals}} \text{ x } N_{\text{dates}}$ → computational expensive

semi-analytical: $\boldsymbol{c.c.} \propto N_1 \text{ x } N_{\text{deals}} \text{ x } N_{\text{dates}}$ → computational cheap

- N_1 is more important in path-dependent deals (due to dependency on the past).
- N_1 is more important when the max. exposure is near to maturity (no more future variance).
- In semi-analytical approach, N_2 is irrelevant.
- In 2-step Montecarlo, N_2 is more relevant when the max. exposure is close to t_0 valuation date.

2. Computational framework (III). Complexity with a netting set

- **Full 2-step Montecarlo valuations:** N_1 scenarios x N_2-Montecarlo
  - Since we need all deal valuations in all relevant dates, computation time will increase quadratically with the number of deals at first. However, it will converge to a linear increase rate as dates become repeated.
  - Computation time increases quadratically with the accuracy of the mean estimators, as usual in Montecarlo methods. The trade-off between computational time vs. percentile accuracy is more complicated to establish, but you can use previous rule as start point for a deeper analysis.

- **Semi-analytical valuation:** N_1 scenarios and closed formula
  - Semi-analytical is significantly faster than 2-step Montecarlo, therefore this is not the primary concern.
  - With N_1, the computational time increases linearly. In early times, the number of scenarios does not affect the accuracy, but close to maturity, the running time will quadratically increase with accuracy as the problem becomes a simple one-step N_1 Montecarlo problem.

- **Hybrid approach**
  - N_1 is fixed by the number of deals with a 2-step Montecarlo valuation, as the semi-analytical process is highly efficient.
  - N_2 only relevant in 2-step Montecarlo deals.

3. Tools. Matlab Parallel Computing Toolbox (I)

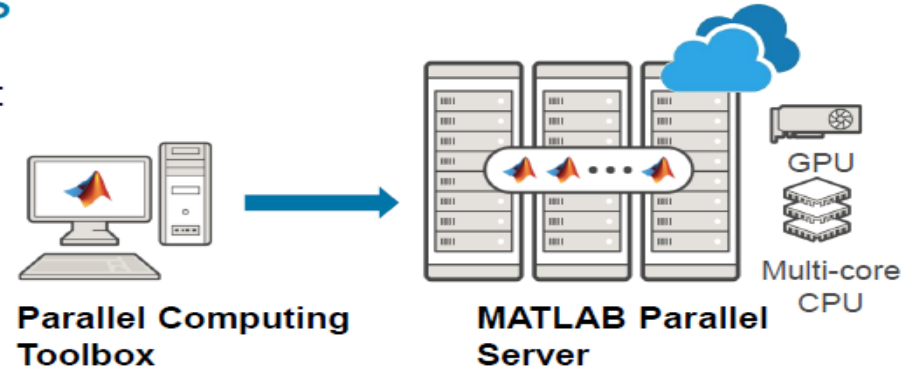
MathWorks

Scaling up: parallel code and simulations

Scaling up our calculations can be needed for various reasons:

- Parallelize computations that take hours or days to run
- Use GPUs for training or computing
- Process Big Data sets stored in cloud machines
- Overcome memory barriers

CUDA MPI

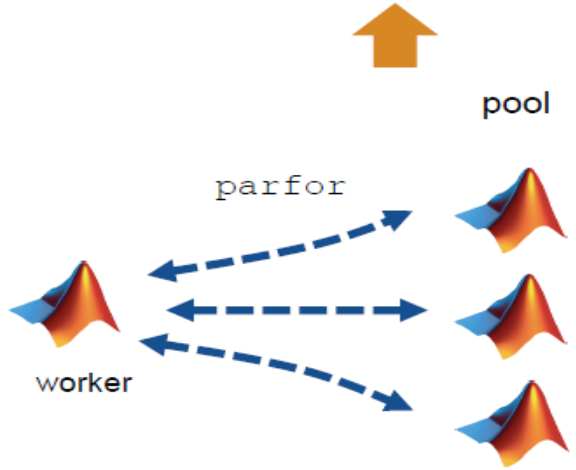



```
>> parfor i = 1:100
     a(i) = max(abs(eig(rand(500))));
   end

>> j = batch(@rand, 1, {10})

>> A = gpuArray(rand(500));
>> e = max(abs(eig(A)));

>> ds = datastore("hdfs://myBigData")
>> mean(tall(ds))
```



17

source: When MATLAB Algorithms Leave their Development Environment, MathWorks 09/02/2023

3. Tools. Matlab Parallel Computing Toolbox (II)

MathWorks

Example: Monte Carlo simulation

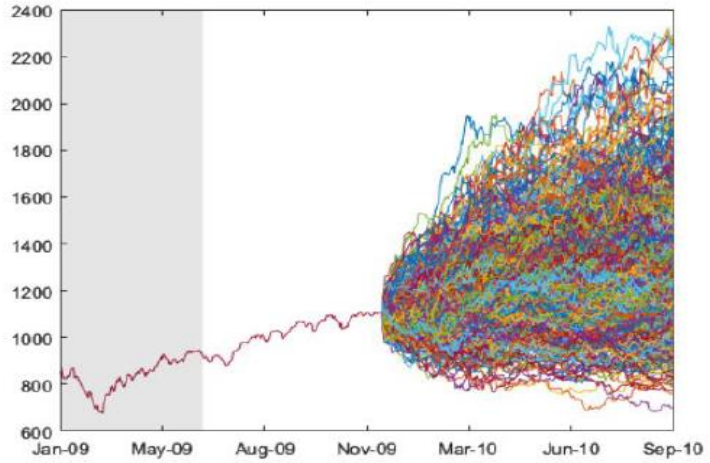
Serial Computation

No change is needed except switching 'for' to 'parfor'.

Simulation

```
tic
for i = 1:numPeriods
    fit = estimate(mdl, ret(:,i), 'display', 'off');
    [E0, V0] = infer(fit, ret(:,i));
    Ysimulations(i,:) = simulate(fit, numTimeSteps, 'NumPaths', paths,...
        'Y0', ret(:,i), 'E0', E0, 'V0', V0);
end
toc
```

```
Elapsed time is 60.179946 seconds.
```



Parfor Computation

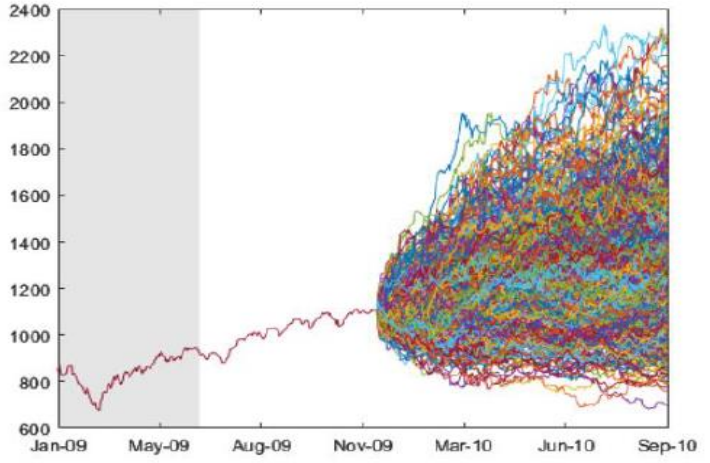
No change is needed except switching 'for' to 'parfor'.

Simulation

```
tic
parfor i = 1:numPeriods
    fit = estimate(mdl, ret(:,i), 'display', 'off');
    [E0, V0] = infer(fit, ret(:,i));
    Ysimulations(i,:) = simulate(fit, numTimeSteps, 'NumPaths', paths,...
        'Y0', ret(:,i), 'E0', E0, 'V0', V0);
end
toc
```

```
Elapsed time is 3.787350 seconds.
```

30 workers: parpool('Cluster', 30)



24

source: High performance computing with MATLAB on the cloud, MathWorks 23/02/2023

4. Conclusion (I). Parallel computing opens a new paradigm

- **The traditional approach** is based on approximate valuation models, which are hard to develop and maintain. There are also some nice models for the simplest cases (e.g. Brigo's formula for CVA of IRS), but they don't fit well in a general framework of netting.

- **The increasing computational power** (cores of a processor, different workstation, cloud computing, etc.) and the generalization of distributed computing, allow us to address the problem in a universal way without such approximations.

A straightforward method to reduce execution time in two-step Monte Carlo problems is to parallelize the valuation across different scenarios among the available computational units (*pool of workers).*

>> parfor 1:N1 *% just one line of Matlab code changes!*

Nevertheless, a large effort may be made on architecture and compute optimization:

- Keeping basic math operations and memory transfers to a minimum.
- Smart selection of simulated dates.
- Dynamic N_1 and N_2 mapping in regard to metric, admitted error and deals of the netting set.
- Finally, the variance of the metric should be evaluated with a representative example.

4. Conclusion (II). Try and compare fast

- **Develop and optimize your architecture with classical *for* loops before changing to *parfor*.**

- **Try different parallelization methods. It's really easy to change with Matlab.**
  - Read toolbox doc first.
  - Choose parallelization method: *Threads* vs. *Process.*
  - Consider *gpuArray* objects.
  - Choose between CPU and GPU…. GPU is not always the best option.

Thank you for your attention.

Pablo García Estébanez
garciapab@bancsabadell.com