

# Speed Up Macroeconomic Modeling with MATLAB and Parallel Computing

Eduard Benet Cerdà

01 - Oct - 2025

*“we have a BVAR model that we would need to speed up”*

*“we have a DSGE model that we would need to speed up”*

*“we have a SSM model that we would need to speed up”*

*“we have a Regime Switching model that we would need to speed up”*

*...*

# Why is speed important?

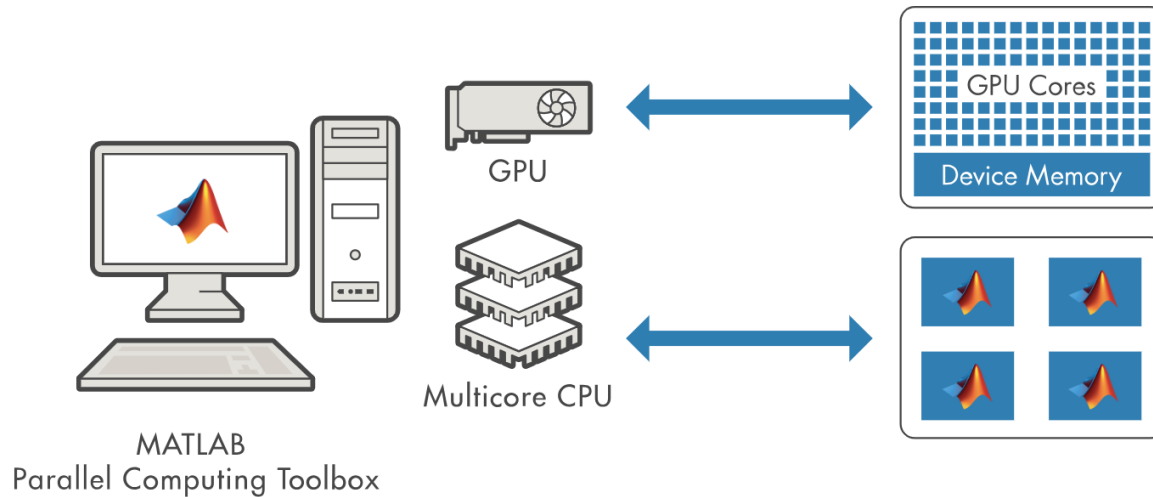
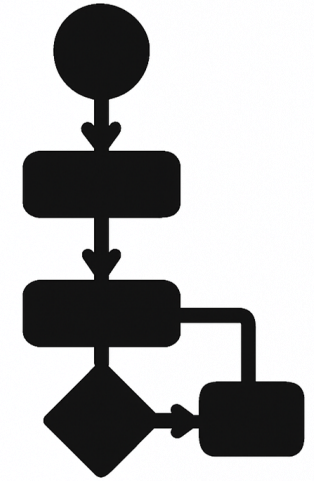
- **Long Computation times**
  - Many macroeconomic models can take 20, 40, or even 100+ hours to run a single estimation.
- **Hardware Utilization**
  - There is often uncertainty about whether current computational resources are being fully leveraged.
- **Scalability needs**
  - Access to more powerful computers is now easier
  - There is a need to distribute calculations across multiple computers
- **Repeated Evaluations**
  - Tasks such as parameter sweeps and Monte Carlo simulations require running many independent model evaluations, which multiplies the total computation time.
  - It also requires a good documentation and coordination of inputs and outputs

# Approaches to Speeding Up Modeling



Refactor and optimize existing code for efficiency.

Adopt more efficient techniques (e.g. auto-diff)  
or develop new algorithms

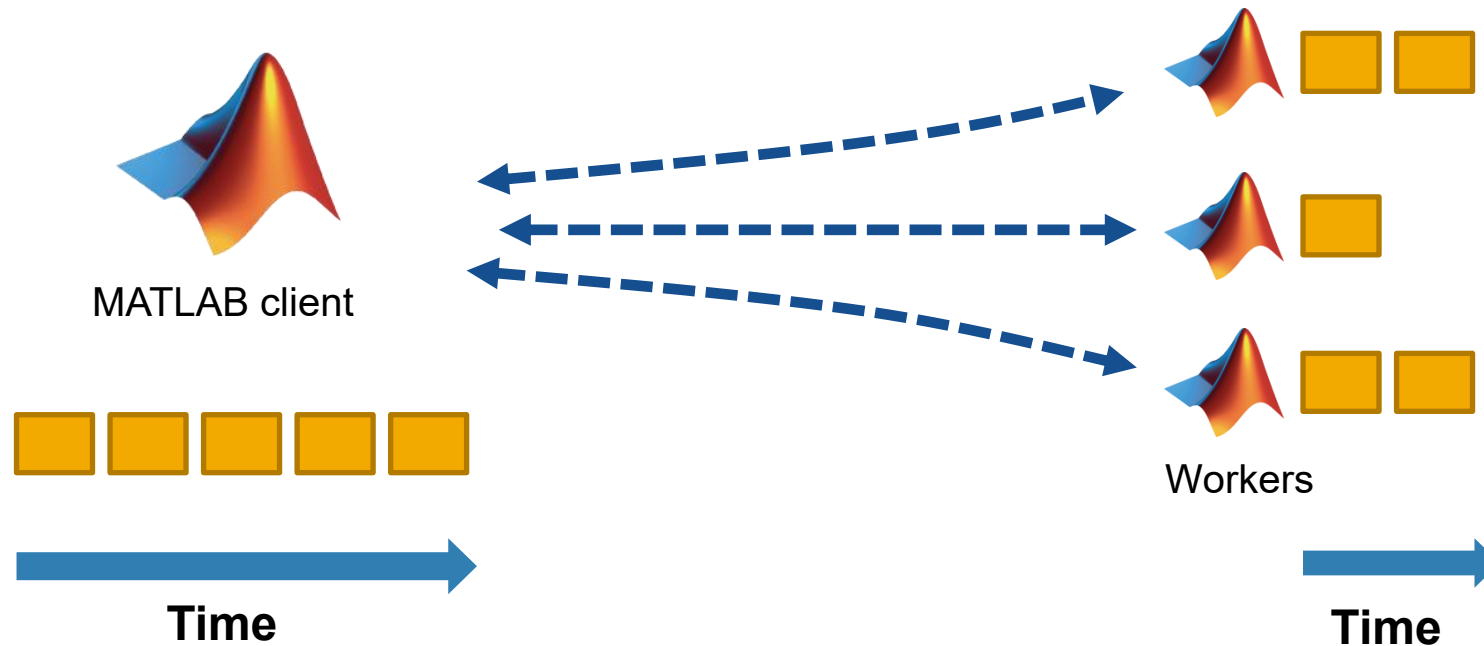


Harness the power of modern multi-core and  
cloud computing to distribute workloads.

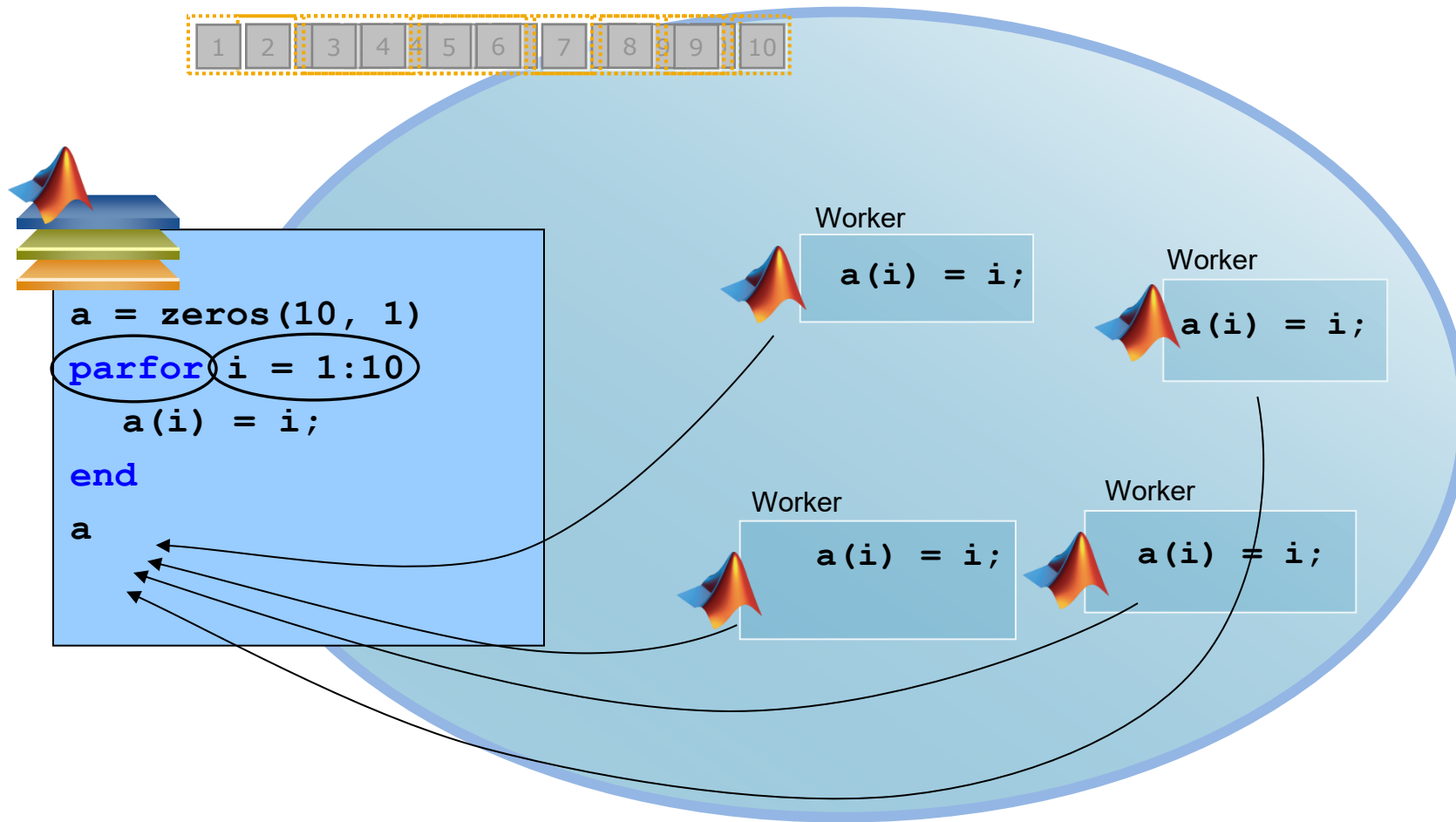
# What is parallel computing?

```
for i = 1:5  
    y(i) = myFunc(myVar(i));  
end
```

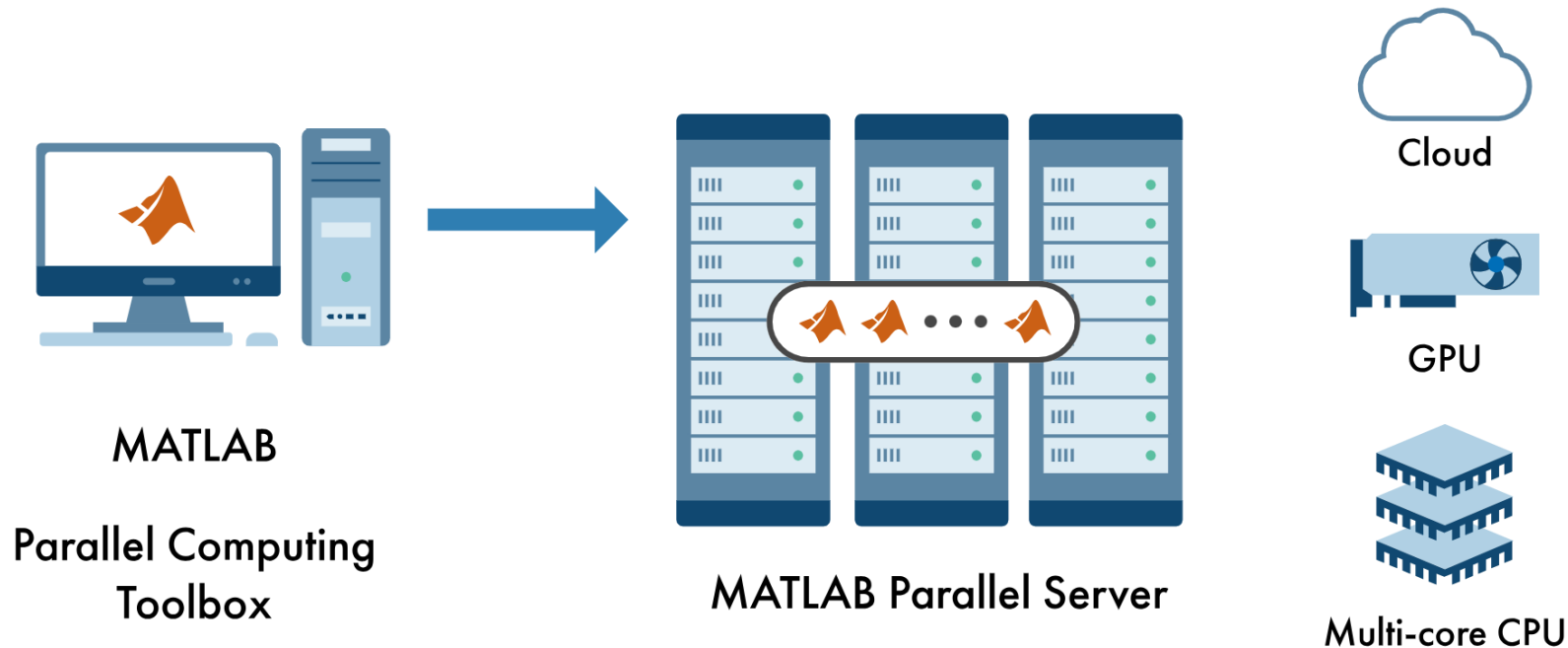
```
parfor i = 1:5  
    y(i) = myFunc(myVar(i));  
end
```



# Mechanics of `parfor` Loops



# Where does it run?

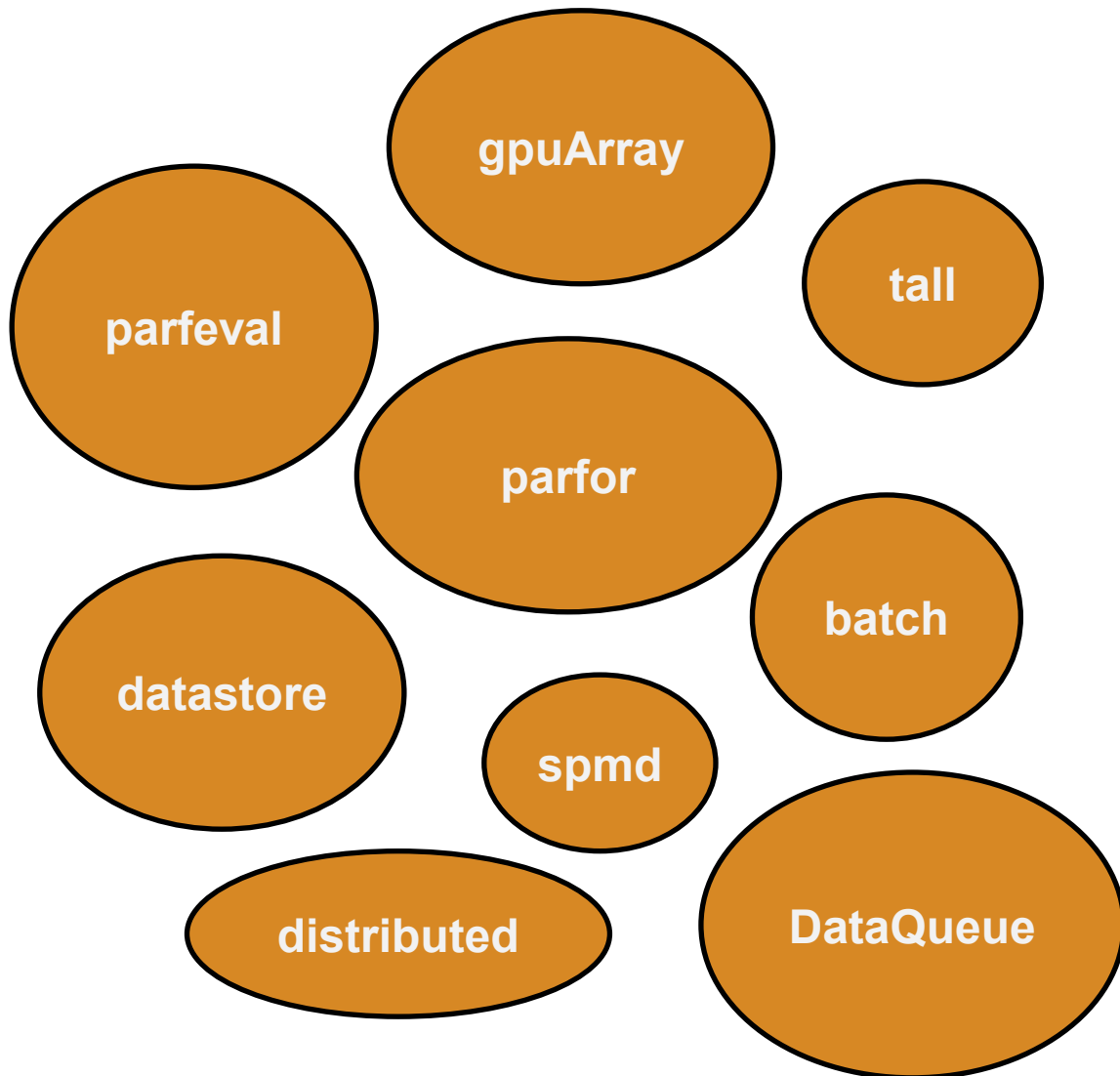


You have full control on where and how the code the code runs, locally, or in the cloud

```
c = parcluster('clusterName');  
c.NumWorkers = x  
c.NumThreads = y
```

```
p = parpool(c)  
parfor i = 1 : N
```

# There is more to parallel computing



## Is not always a trivial problem..

- Some problems can be difficult to adapt into a parfor loop
- parfor is not always the solution, e.g. a Gibbs sampling process.
- Is the potential parallelism obvious to the researcher?
- What is the random seed of my workers?
- How costly is the data transfer?

*Implementing this requires time and expertise, and there is no silver bullet, but there are common patterns.*

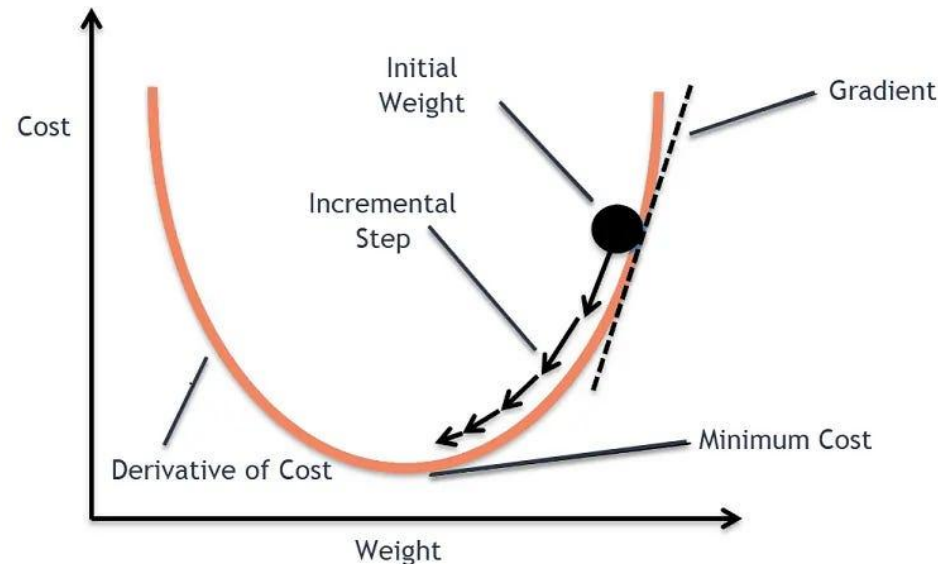


# Parallelism in optimization problems

A primary computational bottleneck in macroeconomic modeling is often an optimization procedure. Luckily, MATLAB streamlines this process by incorporating parallel computing capabilities into most of its optimization algorithms:

```
options = optimoptions('fmincon', UseParallel = true)
```

This will use parallel computing to calculate the derivative in each direction and step of the optimization process.



# Parallelism in optimization problems

A primary computational bottleneck in macroeconomic modeling is often an optimization procedure. Luckily, MATLAB streamlines this process by incorporating parallel computing capabilities into most of its optimization algorithms:

```
options = optimoptions('fmincon', UseParallel = true)
```

This will use parallel computing to calculate the derivative in each direction and step of the optimization process.

An example is ***Prior Selection for Vector Autoregressions (Giannone, Lenza, Primiceri - 2012)*** where the hyperparameters of a BVAR are calculated by maximizing the marginal likelihood

## Pseudocode:

```
opts          = optimoptions('fmincon', UseParallel = true)
hyperparams   = fmincon(@nlogmdd, x0, upperBound, LowerBound, opts);
Mdl           = conjugatebvarm(nseries, nlags, hyperparams)
```

## We might not be calling an optimizer directly...

Often, we use a high-level model or tool that wraps the optimization process. For example, the estimation of a Bayesian State-Space requires tuning a proposal distribution (an optimization problem):

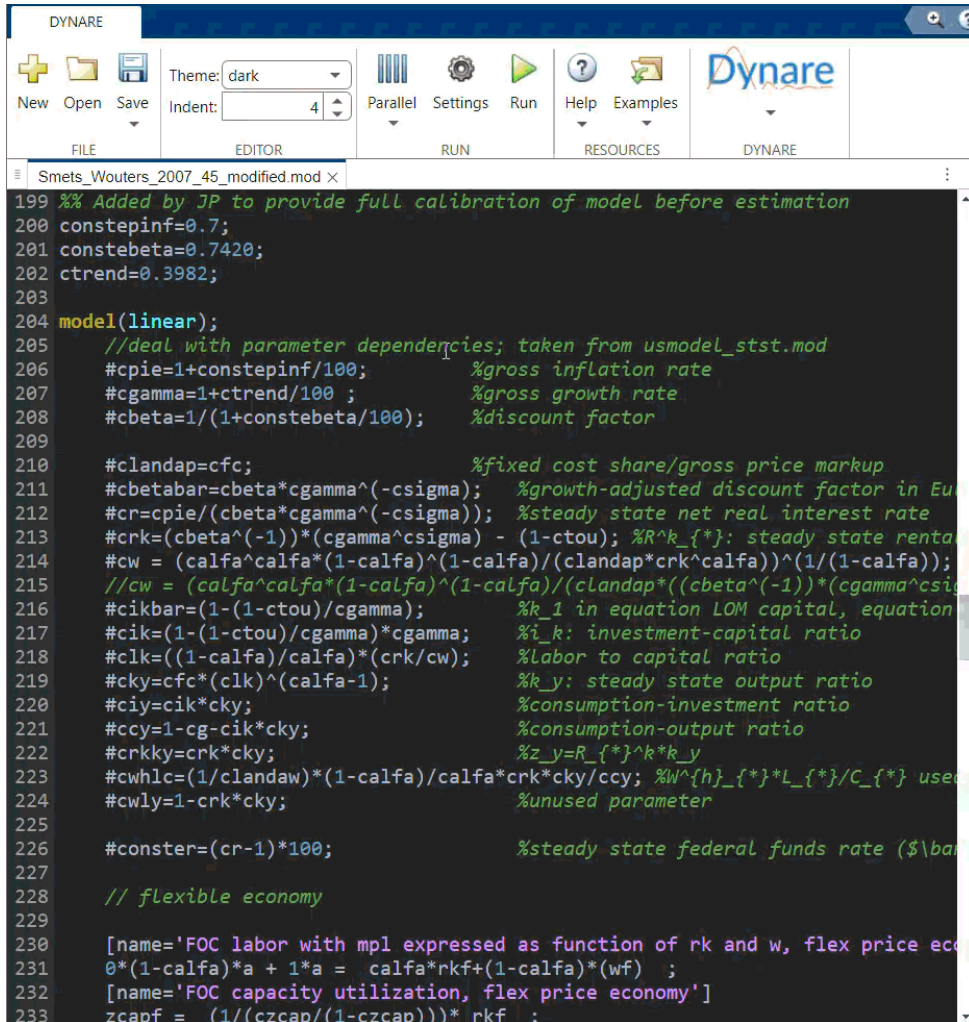
```
MdlB = bssm(@dsgeModel, @dsgePrior);  
PosteriorMdl = estimate(MdlB,Y,params0);
```



```
opts = optimoptions('fminunc', UseParallel = true);  
PosteriorMdl = estimate(MdlB, Y, params0, Options = opts);
```

If the model uses only built-in functions, you can probably take advantage of the above!

# What if it was not directly MATLAB, e.g. Dynare?



```

DYNARE
+ New Open Save Theme: dark Indent: 4 Parallel Settings Run Help Examples Dynare
FILE EDITOR RUN RESOURCES DYNARE
Smets_Wouters_2007_45_modified.mod x
199 %% Added by JP to provide full calibration of model before estimation
200 constepinf=0.7;
201 constebeta=0.7420;
202 ctrend=0.3982;
203
204 model(linear);
205 //deal with parameter dependencies; taken from usmodel_stst.mod
206 #cpi=1+constepinf/100; %gross inflation rate
207 #cgamma=1+ctrend/100; %gross growth rate
208 #cbeta=1/(1+constebeta/100); %discount factor
209
210 #clandap=cfc; %fixed cost share/gross price markup
211 #cbetabar=cbeta*cgamma^(-csigma); %growth-adjusted discount factor in Eu
212 #cr=cpi/(cbeta*cgamma^(-csigma)); %steady state net real interest rate
213 #crk=(cbeta^(-1))*(cgamma^csigma) - (1-ctou); %R^k_*: steady state rental
214 #cw = (calfa^calfa*(1-calfa)^(1-calfa))/(clandap*crk^calfa)^(1/(1-calfa));
215 //cw = (calfa^calfa*(1-calfa)^(1-calfa))/(clandap*((cbeta^(-1))*(cgamma^csi
216 #cikbar=(1-(1-ctou)/cgamma); %k_1 in equation LOM capital, equation
217 #cik=(1-(1-ctou)/cgamma)*cgamma; %i_k: investment-capital ratio
218 #clk=((1-calfa)/calfa)*(crk/cw); %labor to capital ratio
219 #cky=cfc*(clk)^(calfa-1); %k_y: steady state output ratio
220 #ciy=cik*cky; %consumption-investment ratio
221 #ccy=1-cg-cik*cky; %consumption-output ratio
222 #crkky=crk*cky; %z_y=R_*^k*k_y
223 #cwhlc=(1/clandap)*(1-calfa)/calfa*crk*cky/ccy; %W^h_*L_*^k_*C_*^k_* used
224 #cwly=1-cr*cky; %unused parameter
225
226 #conster=(cr-1)*100; %steady state federal funds rate ($\bar{r})
227
228 // flexible economy
229
230 [name='FOC labor with mpl expressed as function of rk and w, flex price ec
231 0*(1-calfa)*a + 1*a = calfa*rkf+(1-calfa)*(wf) ;
232 [name='FOC capacity utilization, flex price economy']
233 zcapf = (1/(czcap/(1-czcap)))*rkf ;
  
```

- Dynare is a popular tool to enter DSGE models.
- A common option is to estimate a model by maximum likelihood. The code below wraps an “fmincon” problem

```

estimation(datafile=usmodel_data, mode_compute=1,
           first_obs=1, presample=4, lik_init=2,
           prefilter=0, mh_replic=0, mh_nblocks=2,
           mh_jscale=0.20, mh_drop=0.2, nograph, nodiagnostic, tex);
  
```



We could be tempted to try this

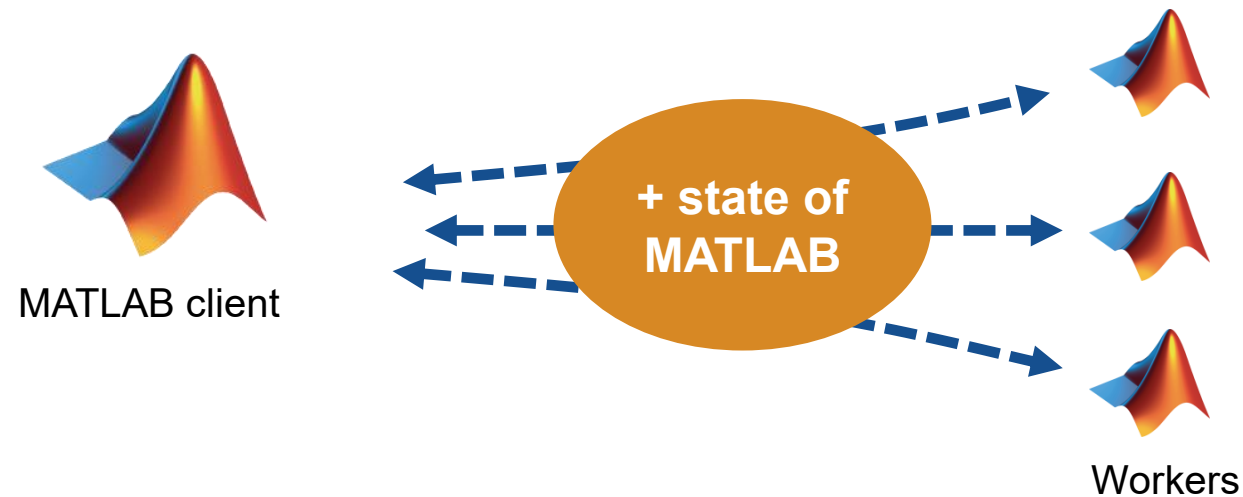
```

estimation(optim=('UseParallel', true),
           datafile=usmodel_data, mode_compute=1,
           first_obs=1, presample=4, lik_init=2,
           prefilter=0, mh_replic=0, mh_nblocks=2,
           mh_jscale=0.20, mh_drop=0.2, nograph, nodiagnostic, tex);
  
```

**Shocks and Frictions in US Business Cycles: A Bayesian DSGE Approach** Smets, Frank and Wouters, Rafael (2007)

## This will not be as trivial... until Dynare 7

- Dynare is a rather complex tool, it uses patterns like variable persistence, and it controls the random seeds. A bit more care needs to be put in distributing parallel work.



- A careful worker initialization is needed

# How is this done?

Since this is a very popular question, we put together the following function:

```
estimation(optim=('UseParallel', true),  
            datafile=usmodel_data, mode_compute=1,  
            first_obs=1, presample=4, lik_init=2,  
            prefilter=0, mh_replic=0, mh_nblocks=2,  
            mh_jscale=0.20, mh_drop=0.2, nograph, nodiagnostic, tex);
```

## Dynare 5, 6

```
>> parpool Processes  
>> out = dynareParallel( 'SmetsWouters.mod', Flags = ["nolog"]);
```

## Dynare 7 (tentative)

```
>> dynare SmetsWouters.mod nolog
```

*On the original model, with “mode\_compute=1”, a 4 core intel I7-1270P provides a speed-up of ~30% of the time.*

# Other approaches: Bayesian estimation with MCMC chains

A similar problem

This algorithm really  
highly parallelizable

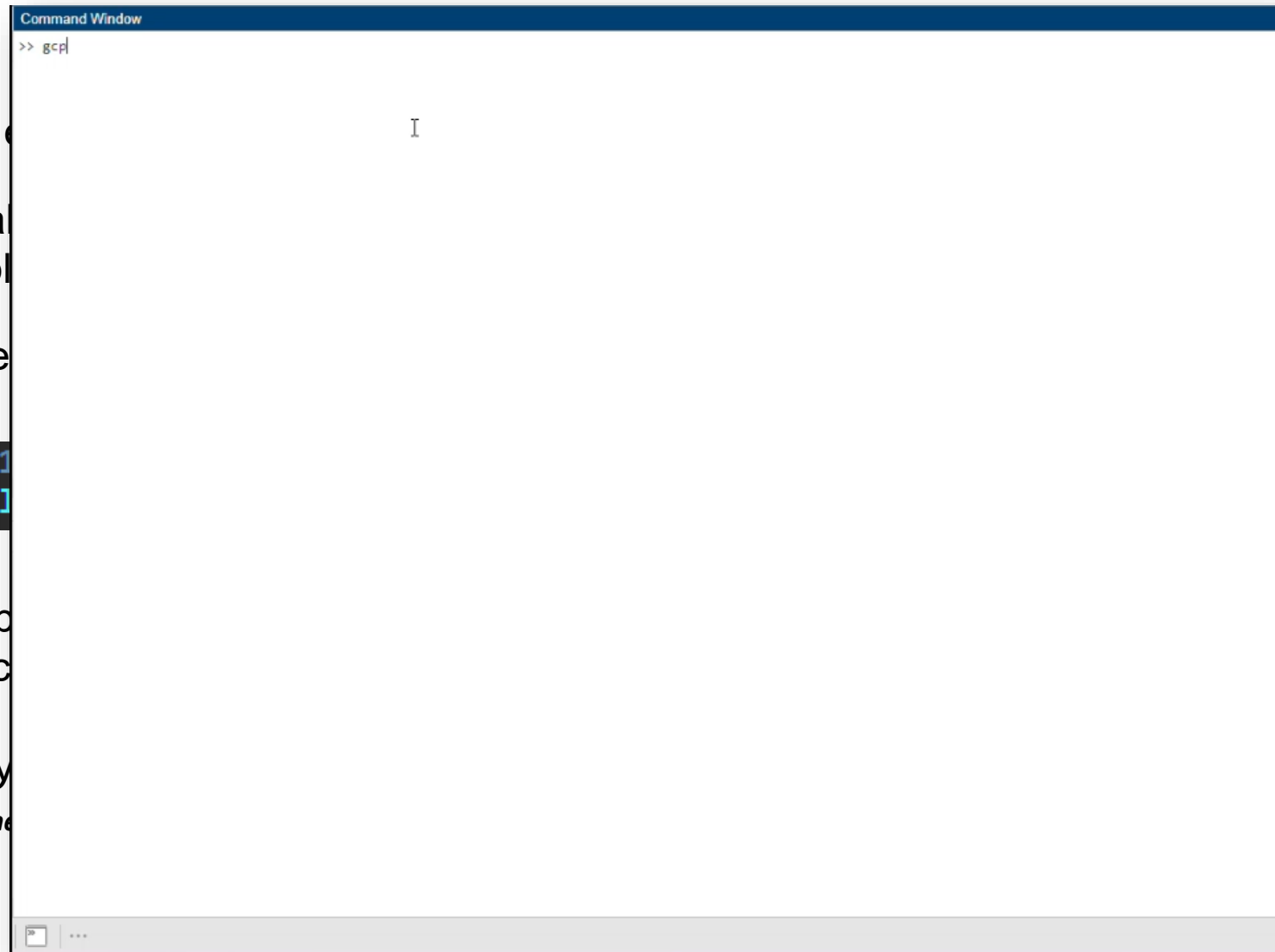
It is a “parfor” type

```
estimation(order=1  
           mh_rep1
```

The benefits can be  
we have enough c

```
out = dy
```

Frank Schorfhe



intense, but

“mh\_nblocks” if

```
ta.m')
```

# Why is speed important?

- **Long Computation times**
  - Many macroeconomic models can take 20, 40, or even 100+ hours to run a single estimation.
- **Hardware Utilization**
  - There is often uncertainty about whether current computational resources are being fully leveraged.
- **Scalability needs**
  - Access to more powerful computers is now easier
  - There is a need to distribute calculations across multiple computers
- **Repeated Evaluations**
  - Tasks such as parameter sweeps and Monte Carlo simulations require running many independent model evaluations, which multiplies the total computation time.
  - It also requires a good documentation and coordination of inputs and outputs



# Repeated evaluations

```
parfor i = 1 : n
    in = myInputs;
    out = runModel(in);
    results{i} = out;
end
```

A second common workflow is the need for running sensitivity analysis, parameter sweeps, or other processes that require multiple model evaluations

- **Error Handling:** Is your code robust in individual runs?
- **Reproducibility:** Are you systematically recording inputs and outputs to allow for reproducibility and audit trails?
- **Interactivity:** Do you need to visualize intermediate results, and is this process streamlined?
- **Scalability:** Can this code be distributed to multiple machines?

## Robust parallel runs can get compiled...

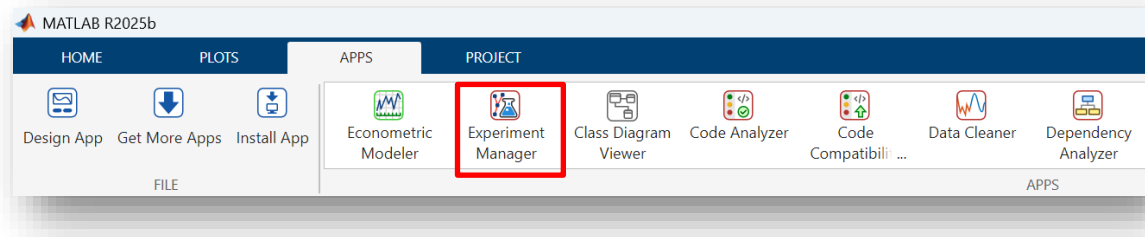
```
f(1:n) = parallel.FevalFuture;  
for i = 1 : n  
    f(i) = parfeval(@runModel, inputs{i});  
end  
  
results = cell(n,1);  
for i = 1:n  
    [idx,out] = fetchNext(f);  
    results{idx} = out;  
end
```

→ Submit asynchronous independent job

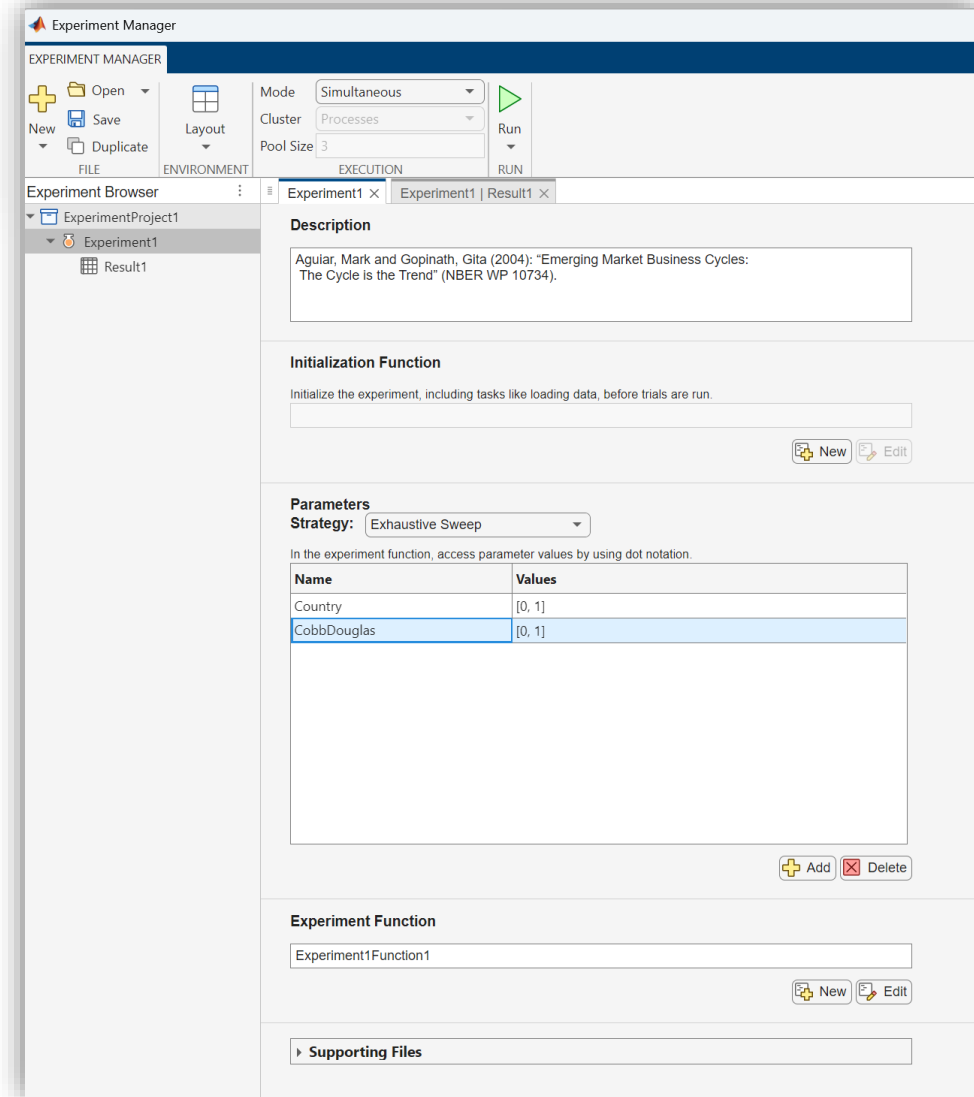
→ Collect and store the results as they finish (not as we submit them)

Getting robust parallel constructs is not always trivial

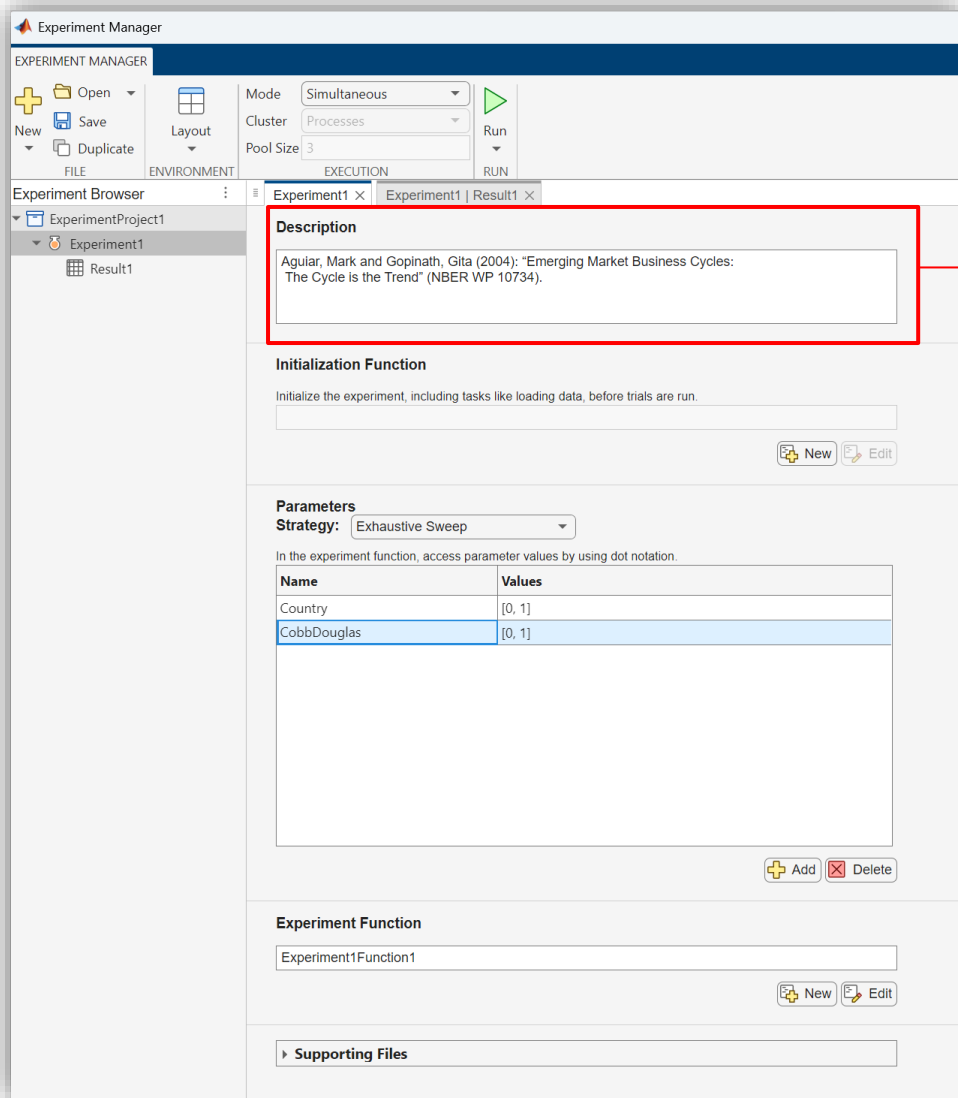
# Experiment Manager



- Organizing multiple experiments and the experiment artifacts and results
- Providing visualizations, filters, and annotations for comparing results
- Storing the experiment definition and parameter combinations for each experiment result



# Experiment Manager



Start with a model parameterized by certain variables. For example, in Dynare

```
// Set the following variable to 0 to get Cobb-Douglas utility
@#define ghh = 1
// Set the following variable to 0 to get the calibration for Canada
@#define country = 1
```

```
dynare myModel.mod -Dghh=1 -Dcountry=0
```

# Experiment Manager

Experiment Manager

EXPERIMENT MANAGER

Mode: Simultaneous  
Cluster: Processes  
Pool Size: 3

Run

Experiment Browser

ExperimentProject1

Experiment1

Result1

Description

Aguilar, Mark and Gopinath, Gita (2004): "Emerging Market Business Cycles: The Cycle is the Trend" (NBER WP 10734).

Initialization Function

Initialize the experiment, including tasks like loading data, before trials are run.

New Edit

Parameters

Strategy: Exhaustive Sweep

In the experiment function, access parameter values by using dot notation.

Name	Values
Country	[0, 1]
CobbDouglas	[0, 1]

Add Delete

Experiment Function

Experiment1Function1

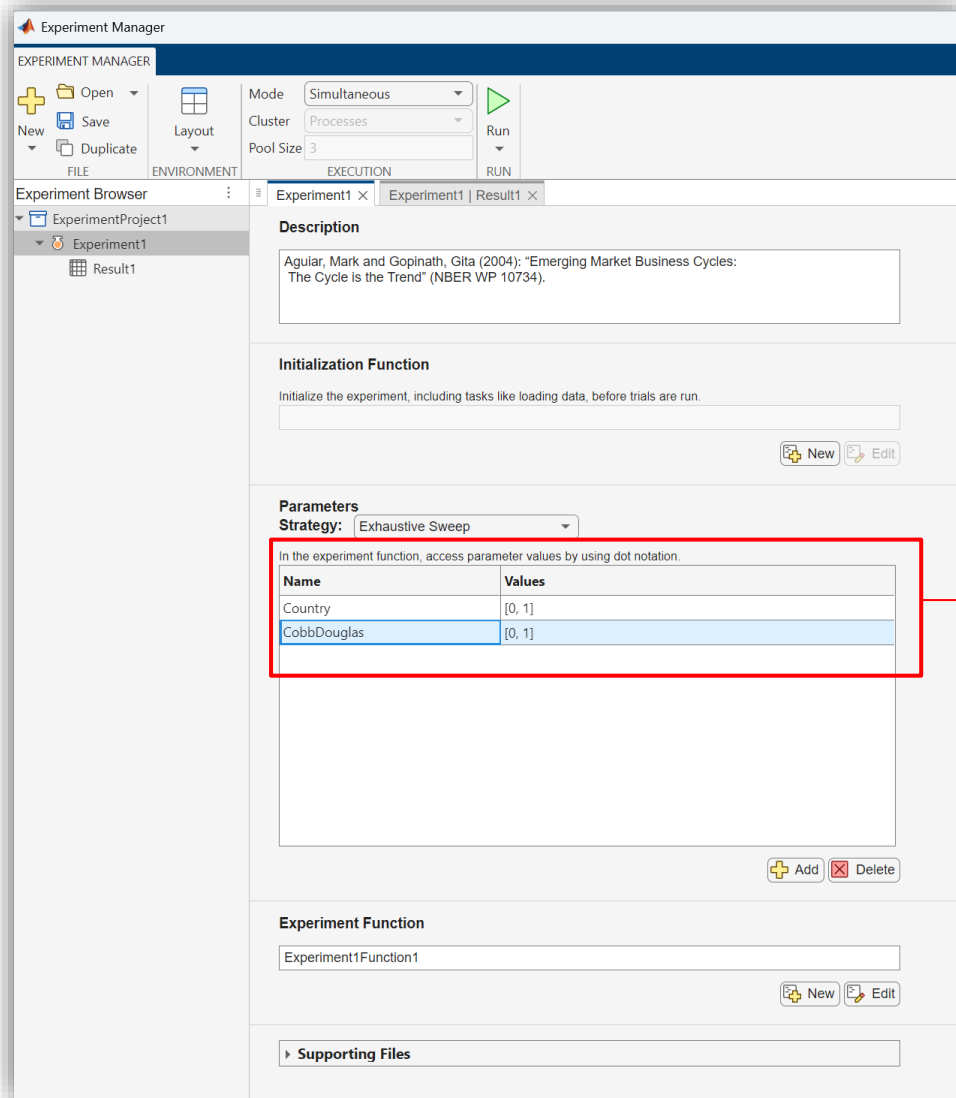
New Edit

Supporting Files

Define a strategy to run your model.

- All parameter combinations?
- A random sample?

# Experiment Manager



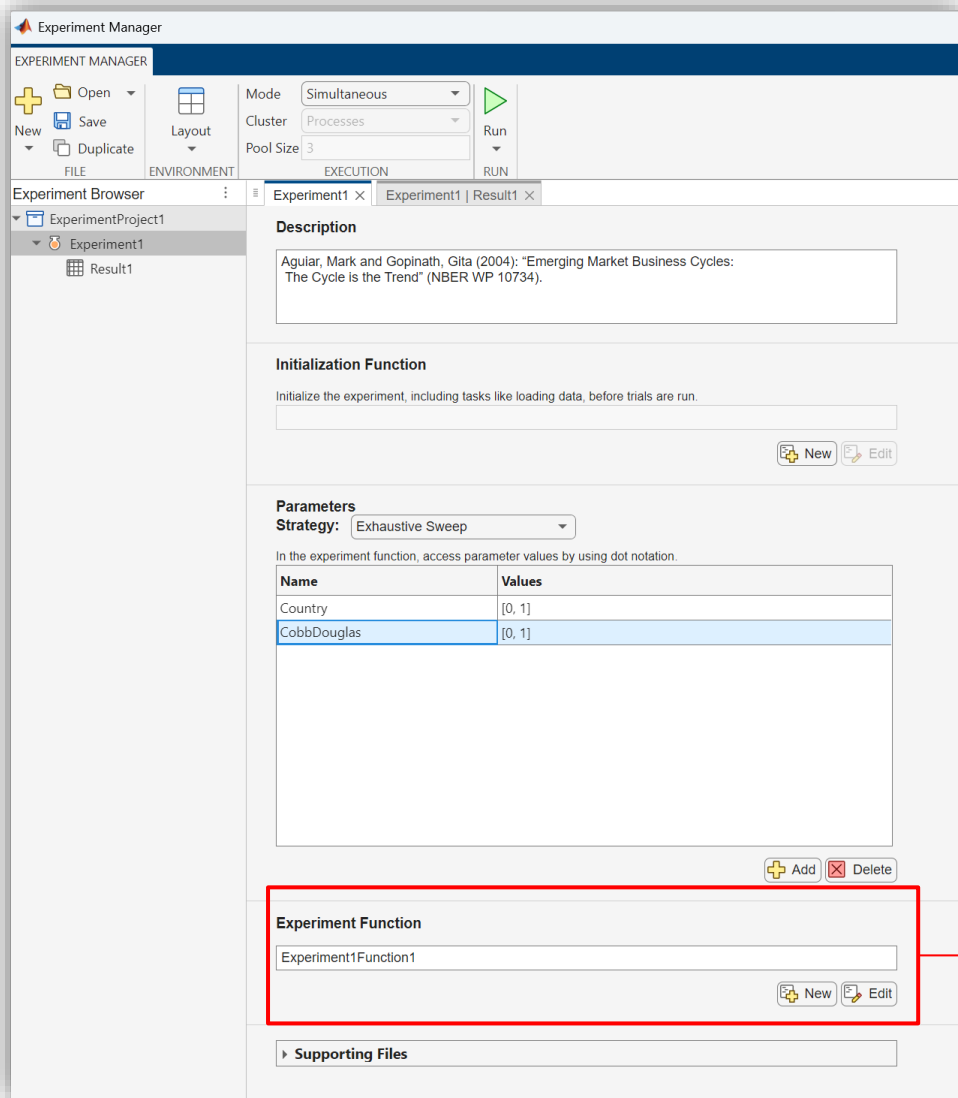
The screenshot shows the Experiment Manager interface. The left sidebar contains the Experiment Browser with a tree view showing 'ExperimentProject1' and 'Experiment1'. The main panel has tabs for 'Experiment1' and 'Result1'. The 'Experiment1' tab is active, showing the 'Parameters' section. The 'Strategy' is set to 'Exhaustive Sweep'. A table lists parameters: 'Country' and 'CobbDouglas', both with values '[0, 1]'. A red box highlights this table, with a red arrow pointing to the code block on the right.

Name	Values
Country	[0, 1]
CobbDouglas	[0, 1]

State the values for your parameter sweep

```
// Set the following variable to 0 to get Cobb-Douglas utility
@#define ghh = 1
// Set the following variable to 0 to get the calibration for Canada
@#define country = 1
```

# Experiment Manager



## Experiment Function

```
function [tb_y, c_y, i_y, oo_] = Experiment1Function1(params)

out = dynareParallel(which('agtrend.mod'), ...
    Flags = [sprintf("-Dghh=%d", params.CobbDouglas), sprintf("-Dcountry=%d", params.Country), "nolog", "nowarn"], ...
    UseParallel = true);

oo_ = out.oo_;

tb_y = oo_.mean(1);
c_y = oo_.mean(2);
i_y = oo_.mean(3);

end
```

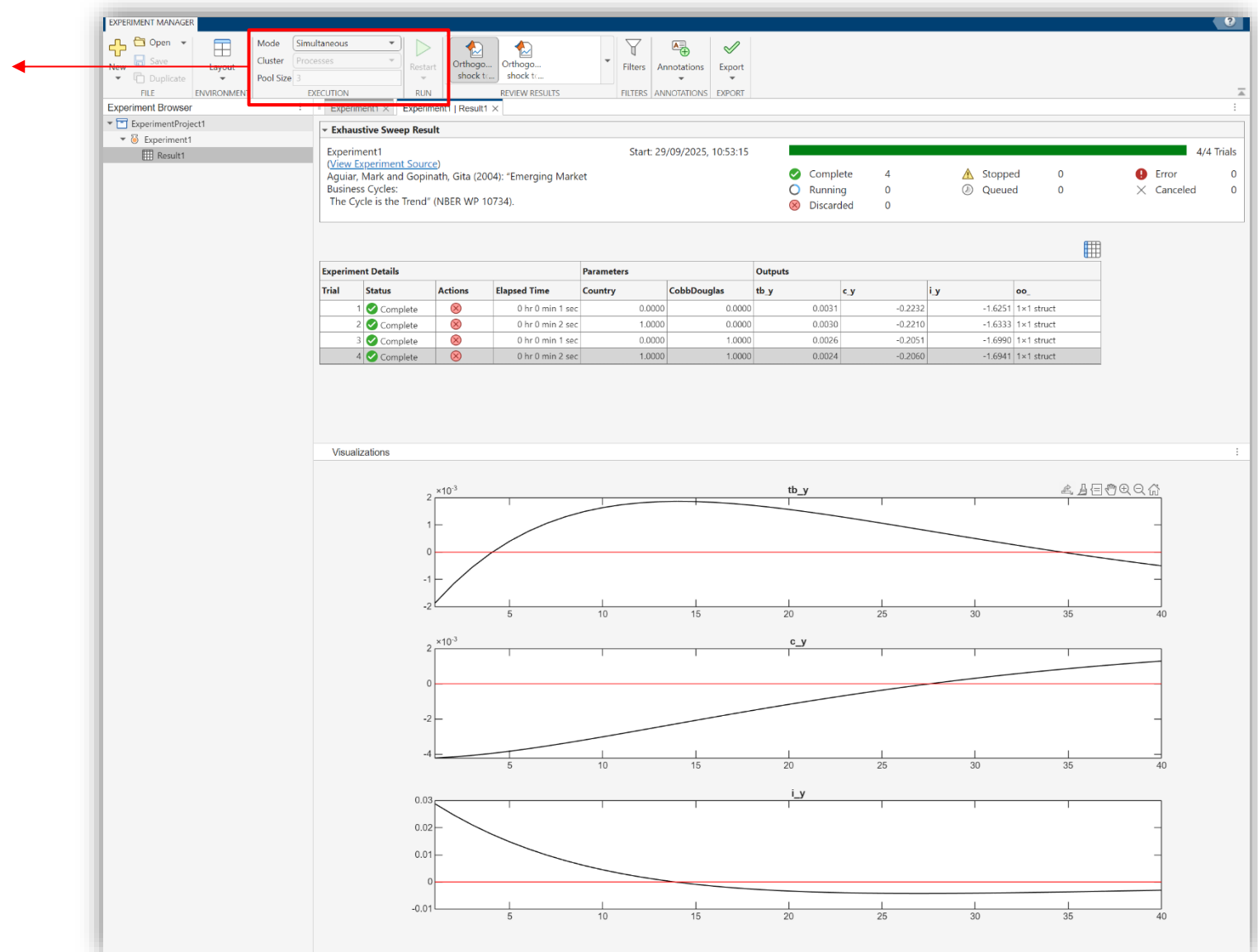
Define the function that calls your model

- Inputs are our parameters
- Recorded outputs are our choice
- All plots are recorded

# Experiment Manager

The Experiment Manager will run our experiments sequentially or in parallel

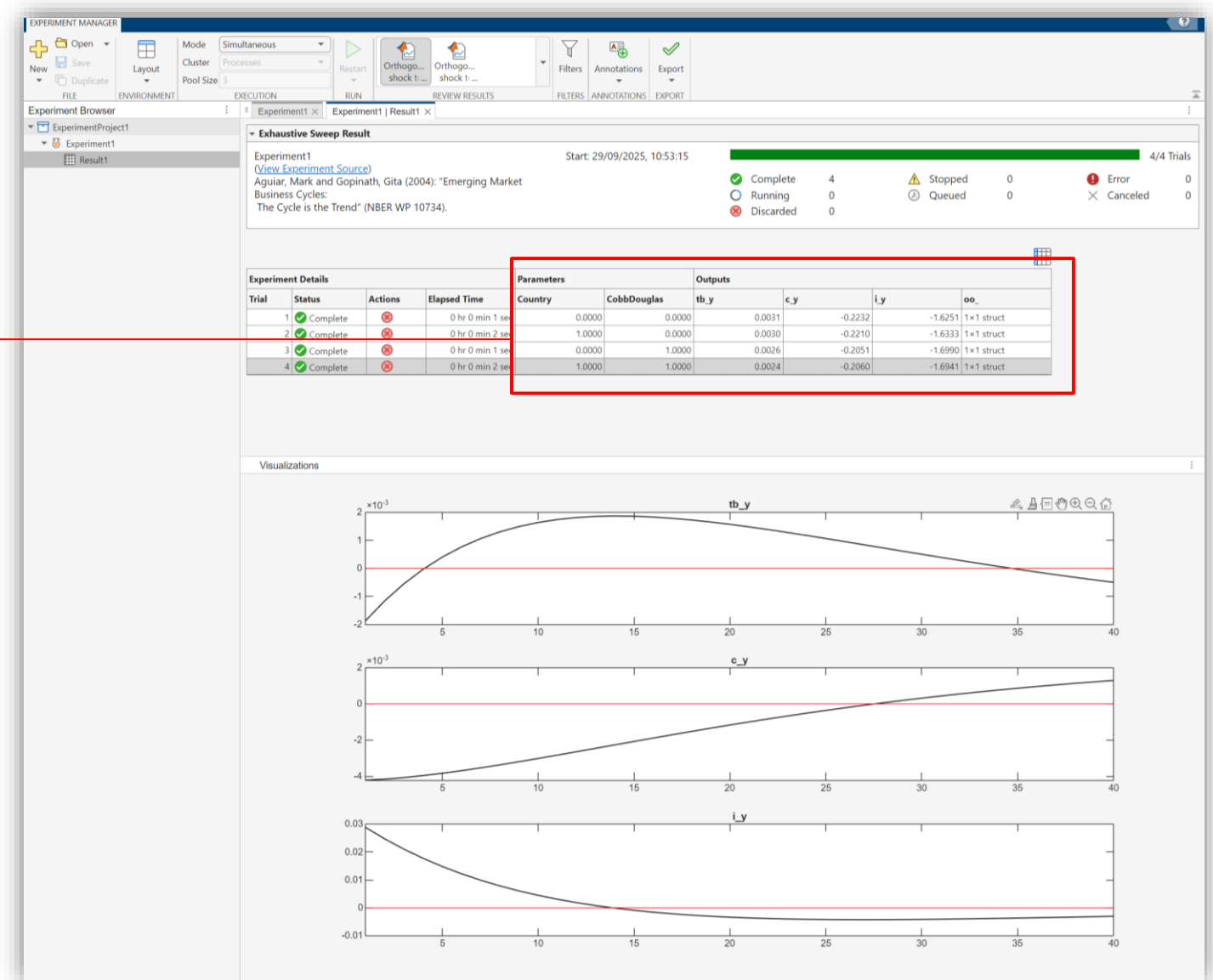
It will also run it in your environment of choice: the current computer or a selected cluster



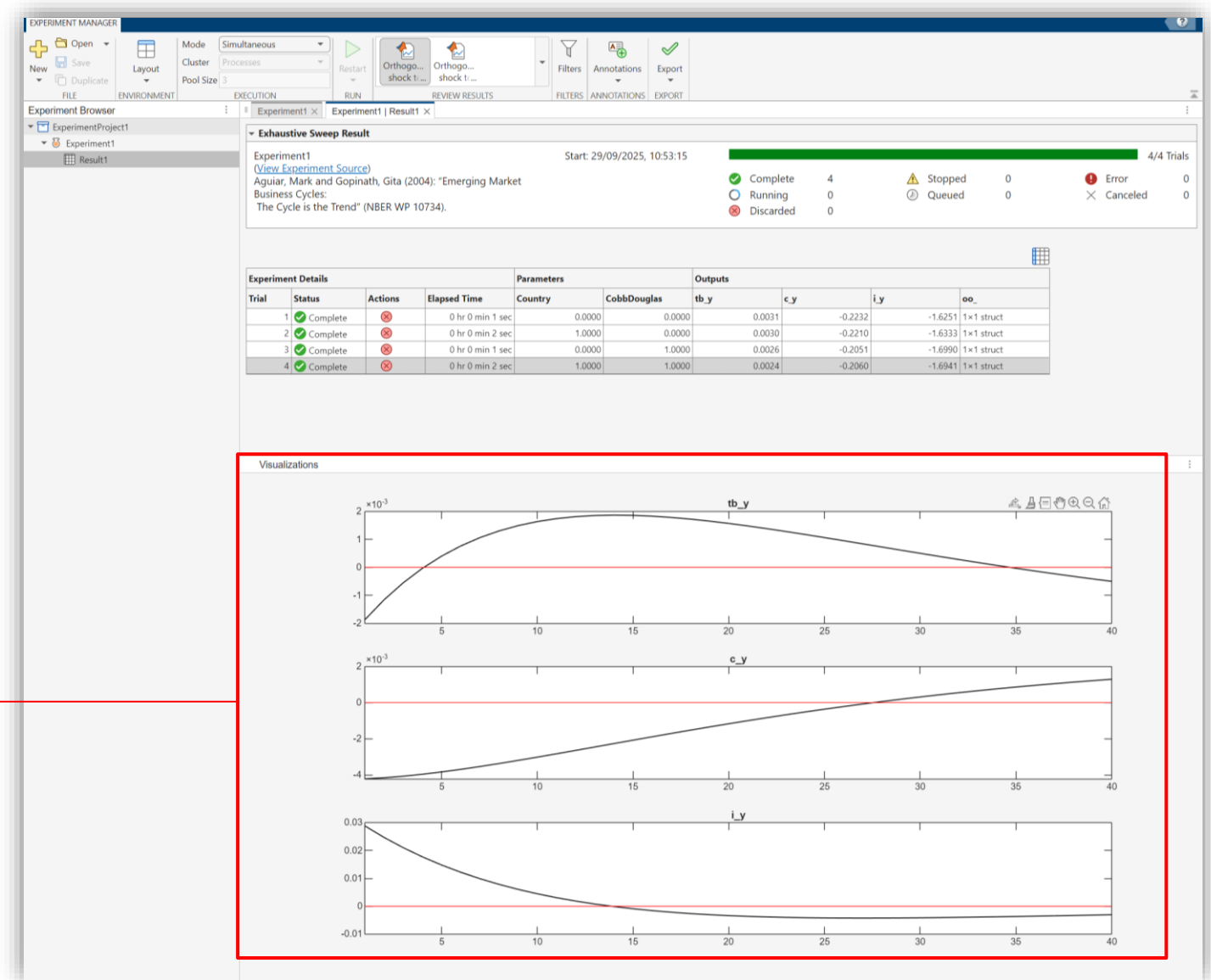


# Experiment Manager

It will save all combinations of inputs and outputs stated in the experiment strategy, and it will let us put any notes into each experiment for the future



# Experiment Manager



Any plot produced by our function, will also be recorded with its corresponding set of inputs and outputs

# Key Takeaways

- **Significant Time Savings**
  - Parallel computing can potentially reduce model estimation times from days to hours
- **Improved Workflow Efficiency**
  - Automate and coordinate large-scale simulation tasks, making research more reproducible and scalable
- **Practical Implementation**
  - MATLAB provides user-friendly constructs (parfor, parfeval, and integration with Dynare) to make parallelization accessible, even for complex macroeconomic models
- **Reach out**
  - If you need help speeding up a model, scaling it, or improving your performance in general, our team is always available to help!

Thank You for joining!

Questions?