modeFRONTIER 2023
Product update

Summary

INTEGRATION AND AUTOMATION

- Guided Process
- Test run in Easy Driver

OPTIMIZATION DRIVEN DESIGN

- Planner and Autonomous Algorithms Updates
- New Python Scheduler bridge

Python Interface to Design Space

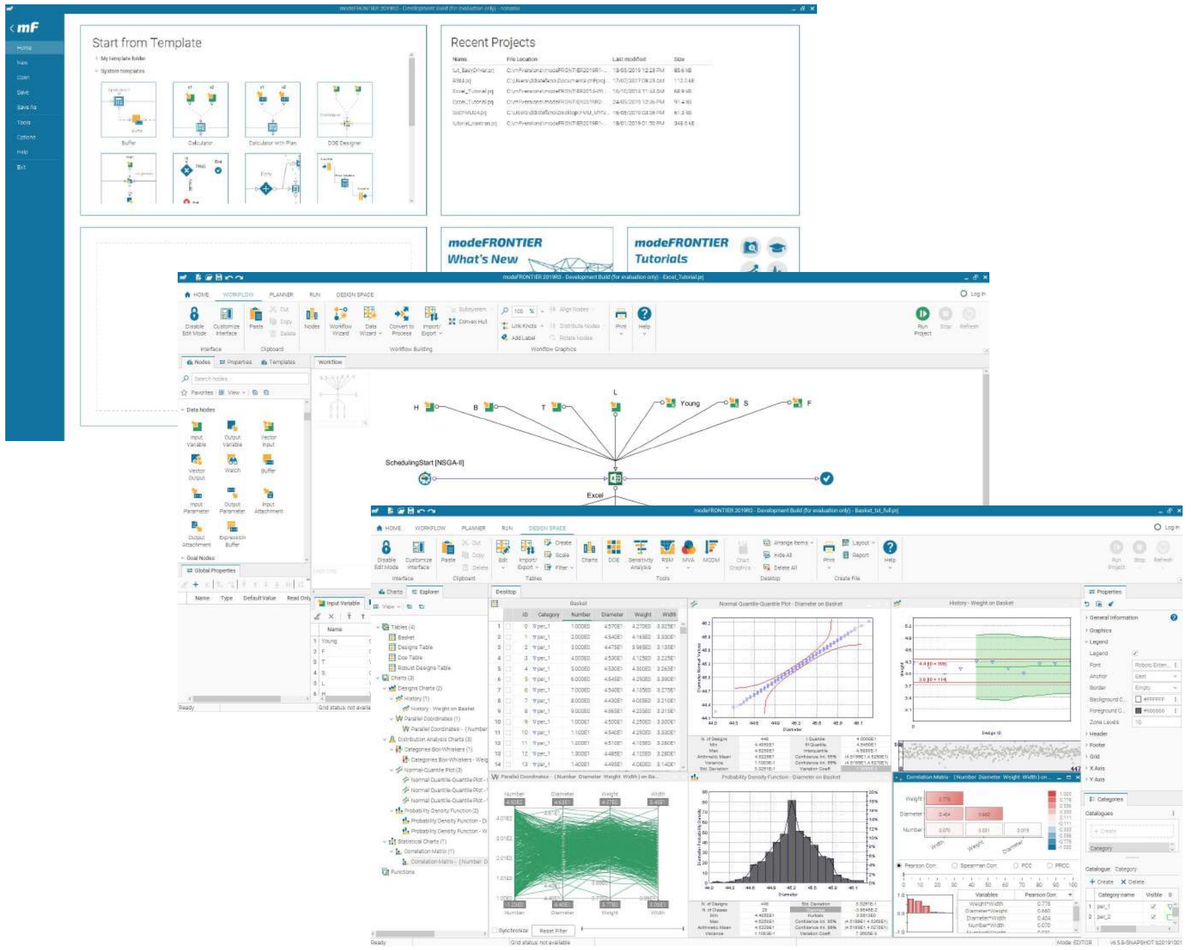
- pyCONSOLE

Snapshots on future implementations

- Connectors SDK
- Plan Task node
- pyCONSOLE as a server

Introducing modeFRONTIER

The leading software solution for **simulation process automation** and **design optimization**



Design better products, faster

Find the optimal design

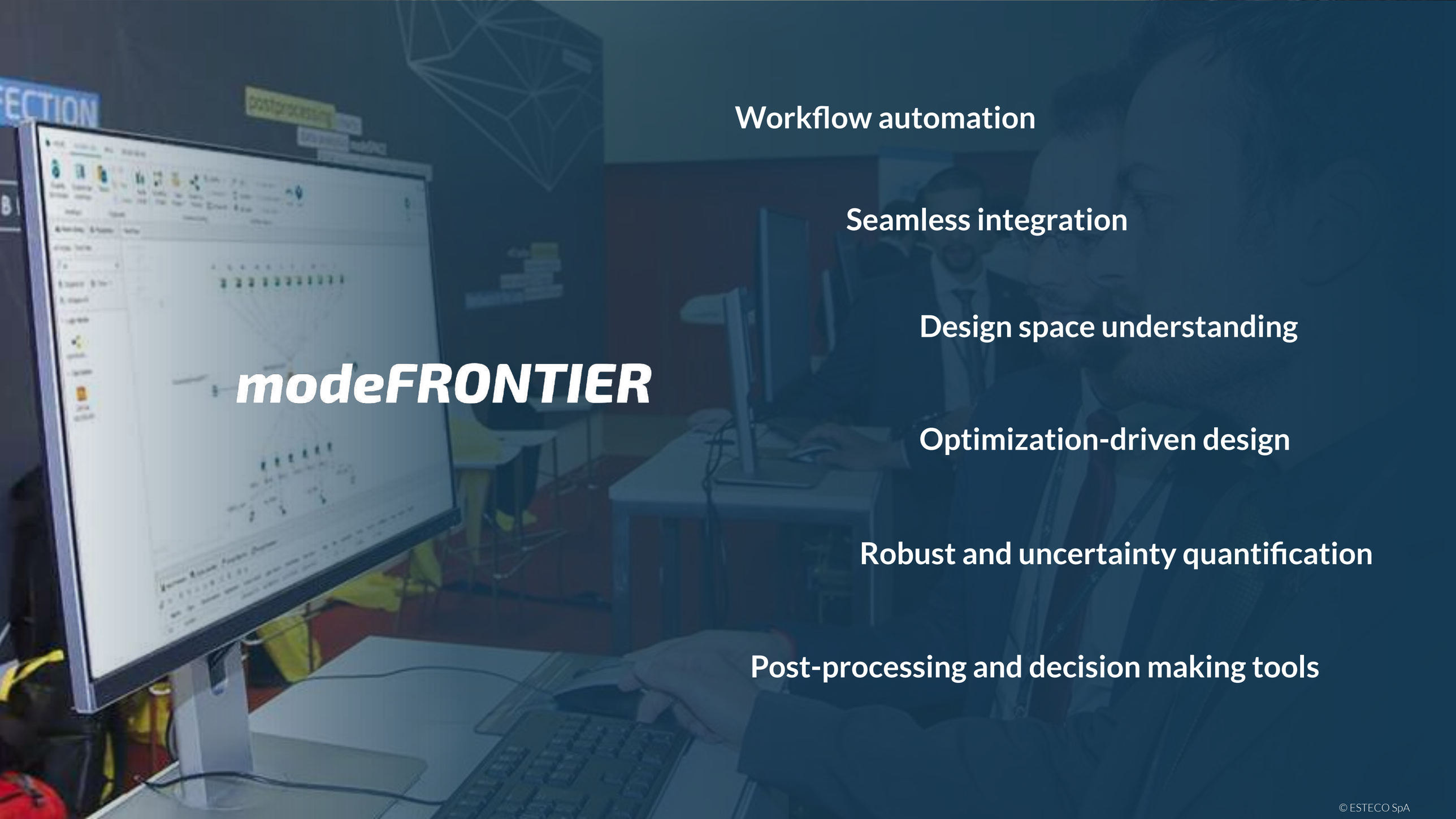
Handle your design parameters and balance conflicting objectives

Maximize IT resources

Exploit all computational resources and engineering solvers

Deliver results on time

Accelerate the engineering process and run multiple simulations



Workflow automation

Seamless integration

Design space understanding

Optimization-driven design

Robust and uncertainty quantification

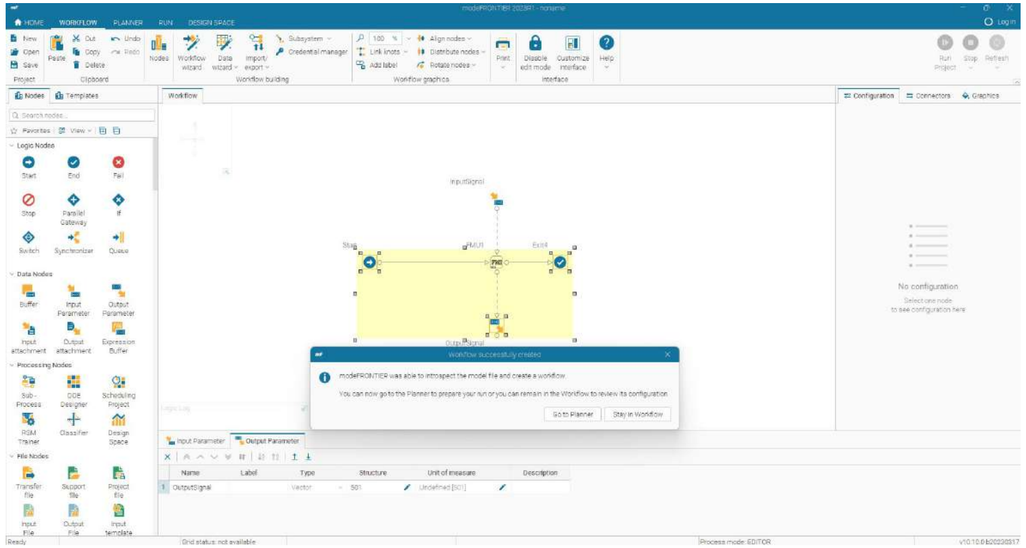
Post-processing and decision making tools

INTEGRATION AND AUTOMATION

New Guided Process
New Test run in Easy Driver
Updates of direct integrations

Guided Process

- Provides a fast gateway for your model to be optimized in modeFRONTIER.
- Drag and drop your model file to the workflow canvas. modeFRONTIER will extract all the parameters and responses from it. Then you can skip the workflow and go directly to the Planner to start the optimization.



modeFRONTIER 2023R1 - ReleaseCandidate3 - RC3 - noname

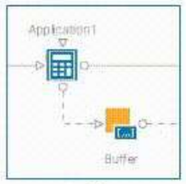
Log in

- Home
- New
- Open
- Save
- Save As
- Tools
- Options
- Info
- Help
- Exit

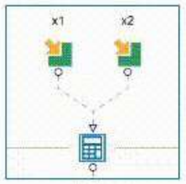
Start from template

My template folder

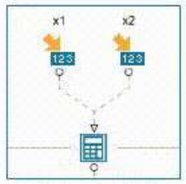
System templates



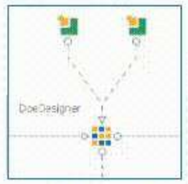
Buffer



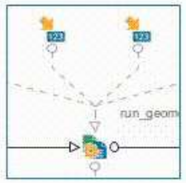
Calculator

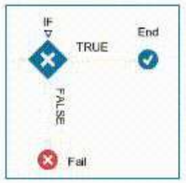


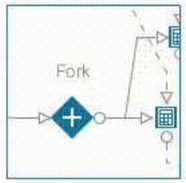
Calculator with Plan

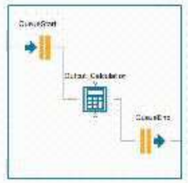


DOE Designer





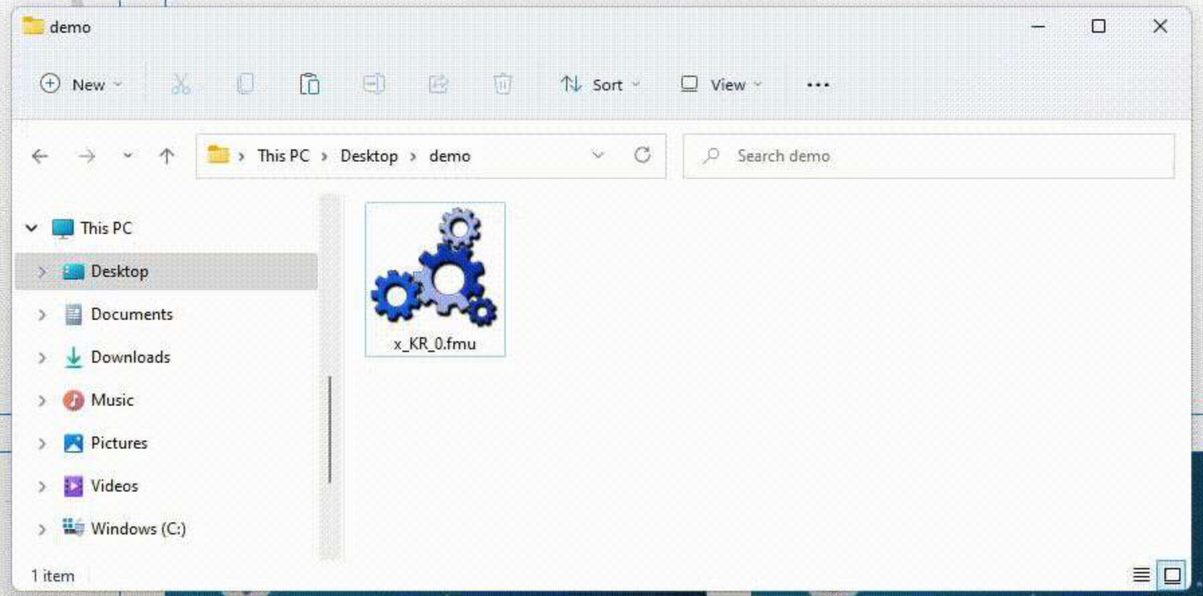




Projects

Pinned projects

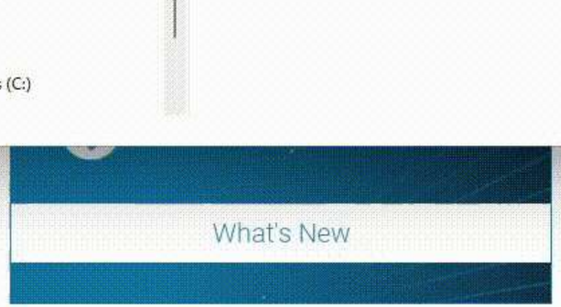
Recent projects

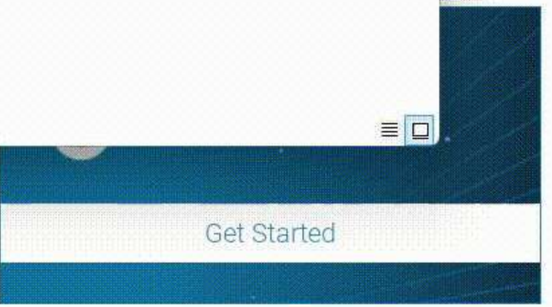


Drag & Drop

a model file or browse to create a complete workflow

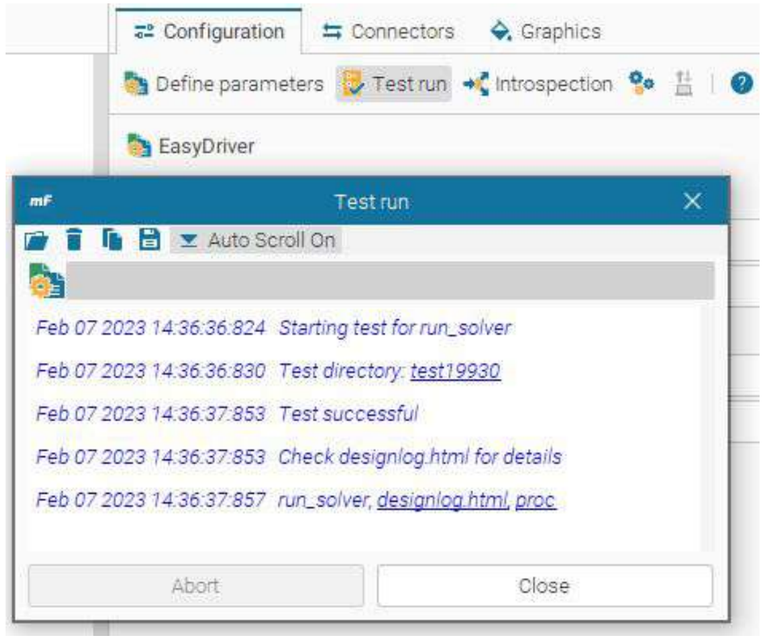
Browse





Easydriver Test Run

- Save time by debugging your integration without running the entire modeFRONTIER workflow.
- You can reproduce the actual runtime environment of your integration and check that everything is right with variables and files in your driver.
- Available in the Easydriver node configuration toolbar.



Updates to Third-Party Integrations

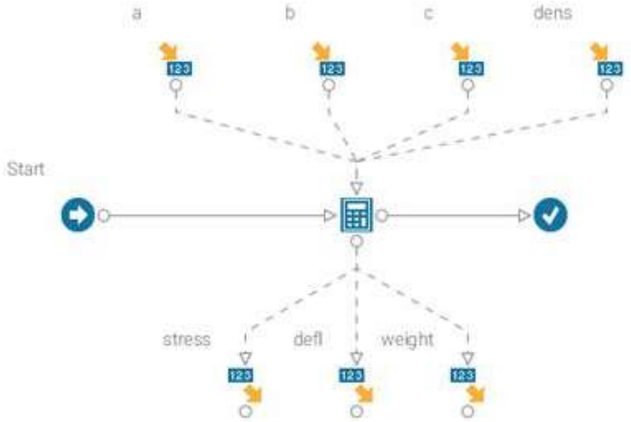
Support of third-party software
extended to the following versions:

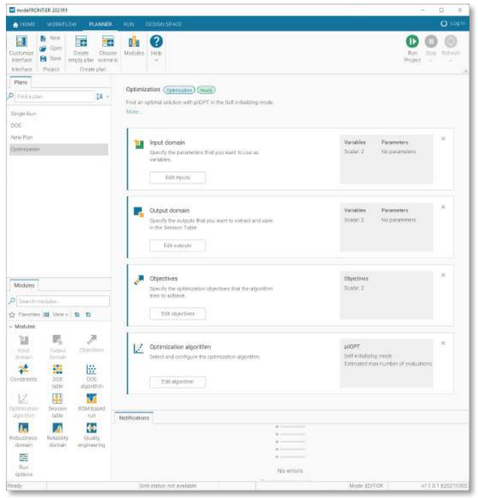
ADAMS/Car 2023.1
ADAMS/View 2023.1
ASMI 2021.2 and 2023
AVL AST 2020.1, 2021.1, 2022.1, 2023.1
CATIA V5-6 R2023
Creo Parametric 10.0
CST Studio Suite 2023
GT-SUITE 2023.1
JMAG 21.1, 22.0 and 22.1
MATLAB 2023a
MSC Nastran 2023.1

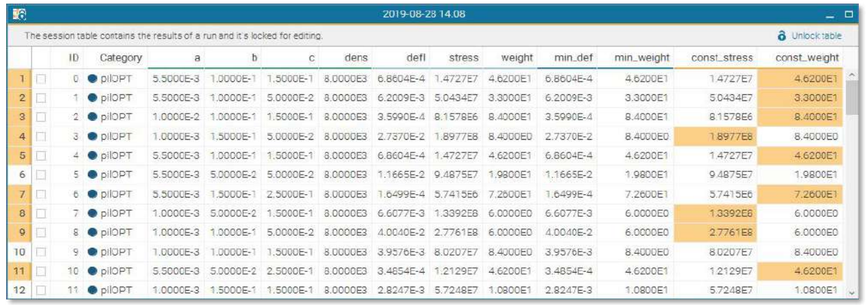
OPTIMIZATION DRIVEN DESIGN

New Planner and Autonomous Algorithms
Python Scheduler Bridge
Python DoE Bridge

A new paradigm – Process, plans, sessions







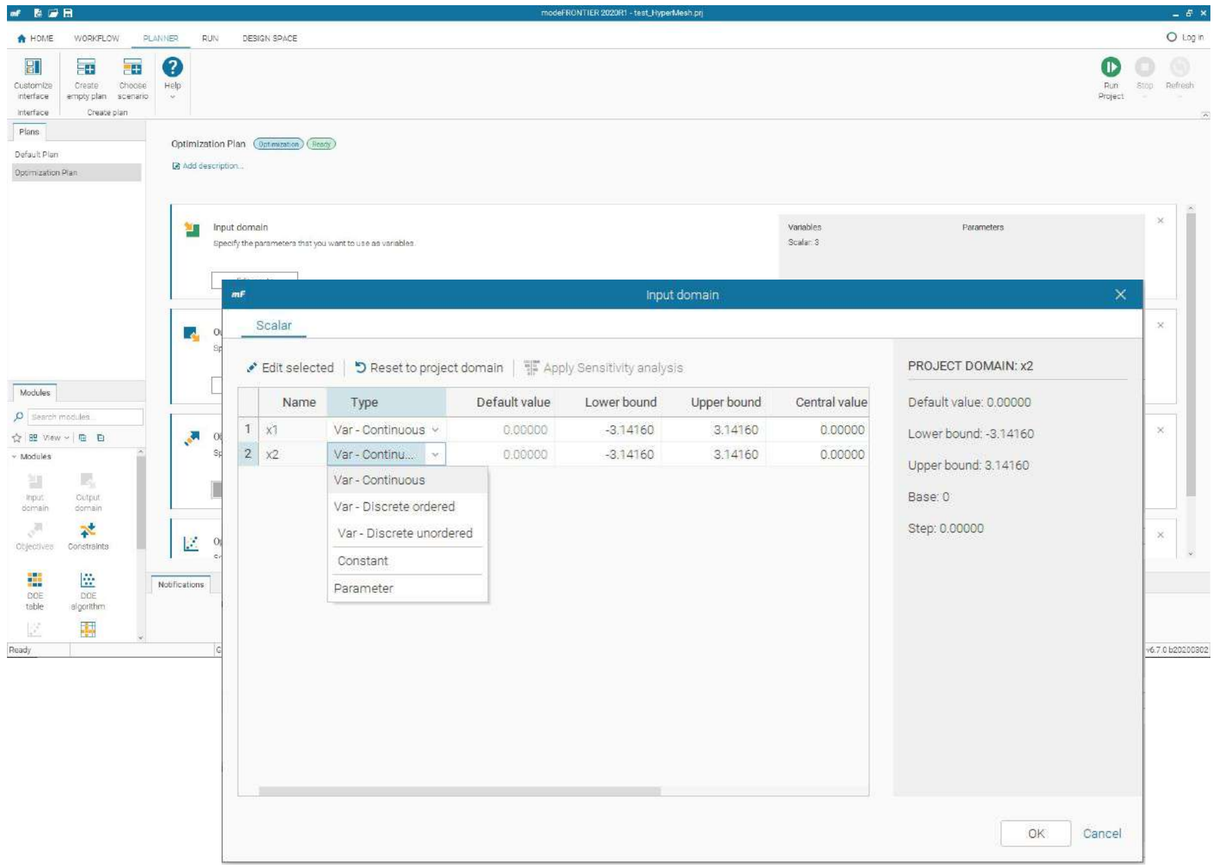
	ID	Category	a	b	c	dens	defl	stress	weight	min_def	min_weight	const_stress	const_weight
1	0	pilOPT	5.5000E-3	1.0000E-1	1.5000E-1	8.0000E3	6.8604E-4	1.4727E7	4.6200E1	6.8604E-4	4.6200E1	1.4727E7	4.6200E1
2	1	pilOPT	5.5000E-3	1.0000E-1	5.0000E-2	8.0000E3	6.2009E-3	5.0434E7	3.3000E1	6.2009E-3	3.3000E1	5.0434E7	3.3000E1
3	2	pilOPT	1.0000E-2	1.0000E-1	1.5000E-1	8.0000E3	3.5990E-4	8.1578E6	8.4000E1	3.5990E-4	8.4000E1	8.1578E6	8.4000E1
4	3	pilOPT	1.0000E-3	1.5000E-1	5.0000E-2	8.0000E3	2.7370E-2	1.8977E8	8.4000E0	2.7370E-2	8.4000E0	1.8977E8	8.4000E0
5	4	pilOPT	5.5000E-3	1.0000E-1	1.5000E-1	8.0000E3	6.8604E-4	1.4727E7	4.6200E1	6.8604E-4	4.6200E1	1.4727E7	4.6200E1
6	5	pilOPT	5.5000E-3	5.0000E-2	5.0000E-2	8.0000E3	1.1665E-2	9.4875E7	1.9800E1	1.1665E-2	1.9800E1	9.4875E7	1.9800E1
7	6	pilOPT	5.5000E-3	1.5000E-1	2.5000E-1	8.0000E3	1.6499E-4	5.7415E6	7.2600E1	1.6499E-4	7.2600E1	5.7415E6	7.2600E1
8	7	pilOPT	1.0000E-3	5.0000E-2	1.5000E-1	8.0000E3	6.6077E-3	1.3392E8	6.0000E0	6.6077E-3	6.0000E0	1.3392E8	6.0000E0
9	8	pilOPT	1.0000E-3	1.0000E-1	5.0000E-2	8.0000E3	4.0040E-2	2.7761E8	6.0000E0	4.0040E-2	6.0000E0	2.7761E8	6.0000E0
10	9	pilOPT	1.0000E-3	1.0000E-1	1.5000E-1	8.0000E3	3.9576E-3	8.0207E7	8.4000E0	3.9576E-3	8.4000E0	8.0207E7	8.4000E0
11	10	pilOPT	5.5000E-3	5.0000E-2	2.5000E-1	8.0000E3	3.4854E-4	1.2129E7	4.6200E1	3.4854E-4	4.6200E1	1.2129E7	4.6200E1
12	11	pilOPT	1.0000E-3	1.5000E-1	1.5000E-1	8.0000E3	2.8247E-3	5.7248E7	1.0800E1	2.8247E-3	1.0800E1	5.7248E7	1.0800E1

The Planner

The Planner is a new environment for creating and defining design space exploration/optimization plans.

You can define and save any number of plans in the same project to cover different scenarios.

This new feature enables and simplifies the set up and analysis of different design space exploration strategies for an engineering design problem.

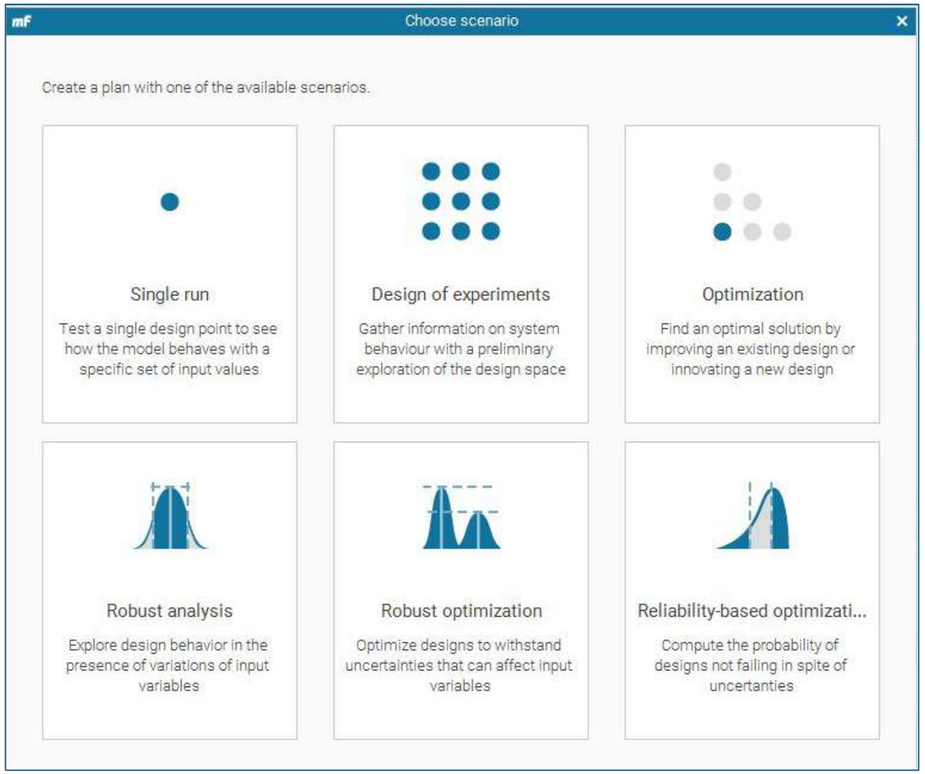


Planner pre-defined scenarios

To perform a study you can choose from 6 types of plans, called scenarios:

- Single Run
- Design of Experiments
- Optimization
- Robust Analysis/Optimization - Reliability

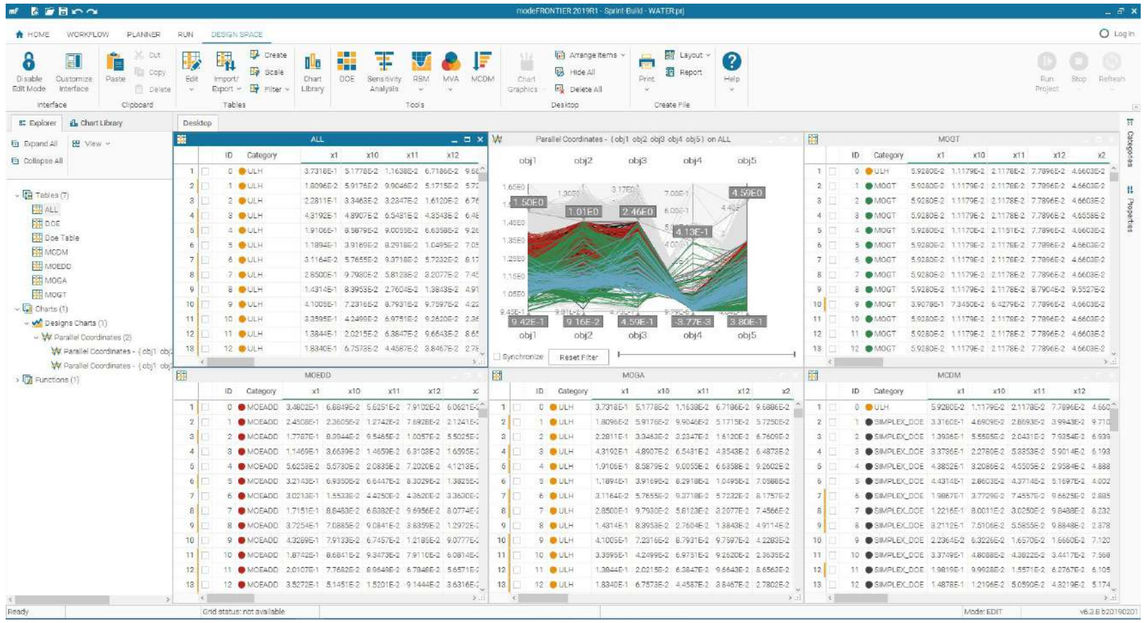
Each scenario includes all steps required for a specific study – you only need to configure them.



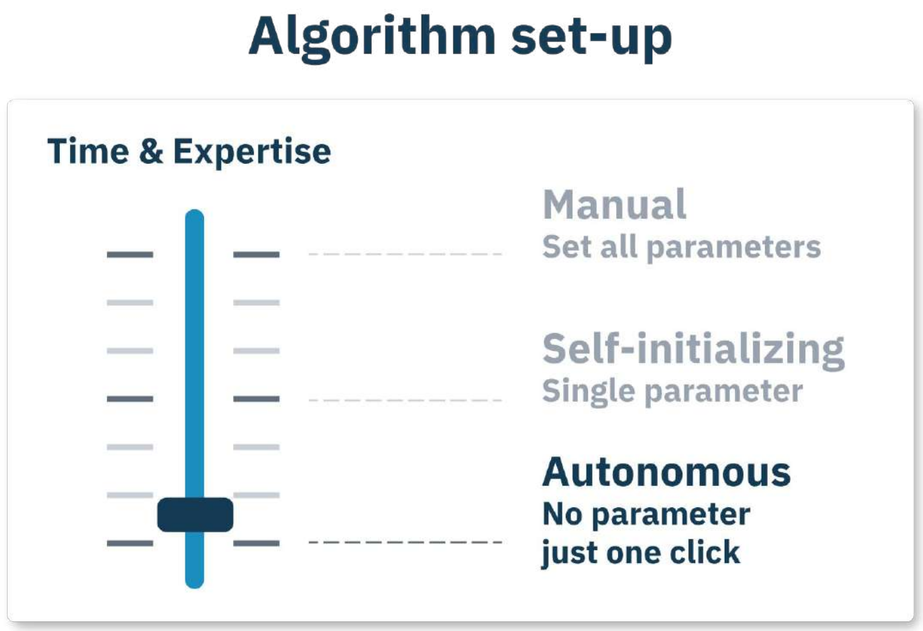
Design Space - Session Tables

You can freely decide where you want to save the results of a session without duplicating the project.

The table with the results of a session is called Session Table and it is not affected by any change in the workflow. In this way you never lose any data.



Pick your mode to run algorithms



- **Manual**
  Set all the algorithm parameters to build your own optimization strategy

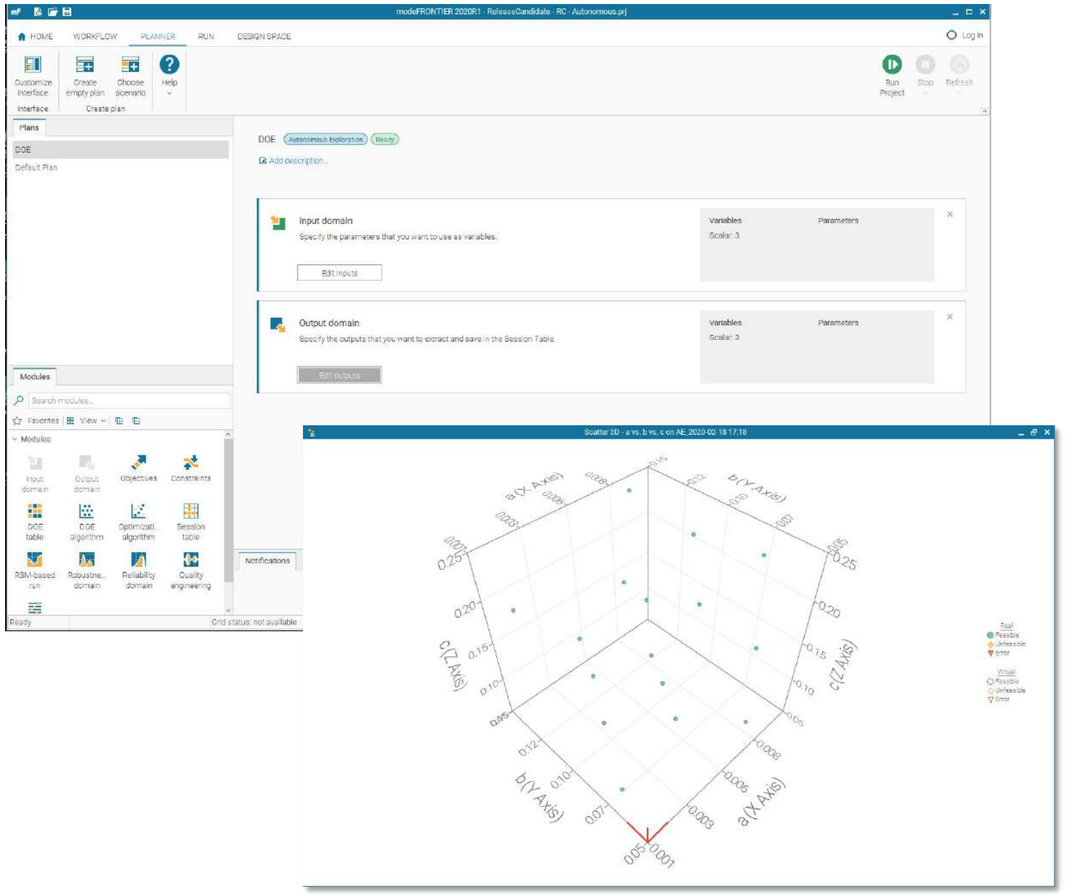
- **Self-Initializing**
  Simply set the number of evaluations to start searching for optimal solutions

- **Autonomous**
  Embrace the one-click revolution, press play to start the optimization process

Autonomous Design of Experiment

Autonomous mode is now available also for Design of Experiments. It uses the number of input variables to automatically define appropriate DOE size and distribution.

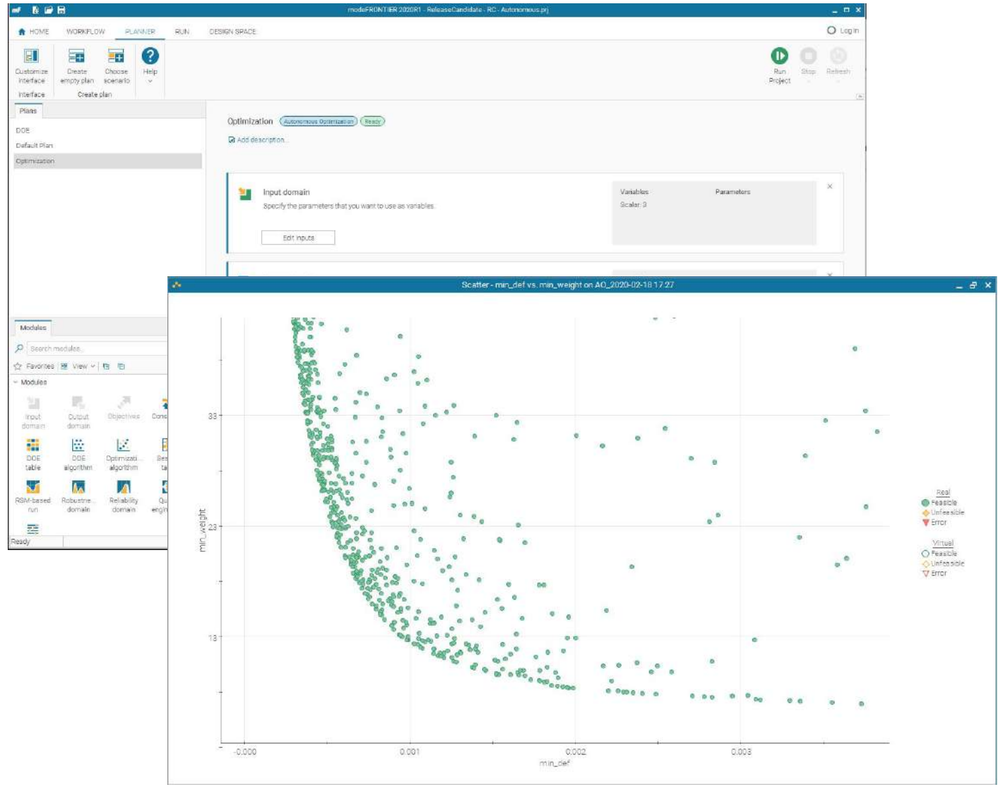
As alternative, DOE can be specified (algorithm and option) manually



Autonomous or Self-initializing optimization

By simply adding objectives to an Autonomous exploration plan turns it into an Autonomous optimization plan (pilOPT), ready to be used with no need to specify and configure any optimization algorithm.

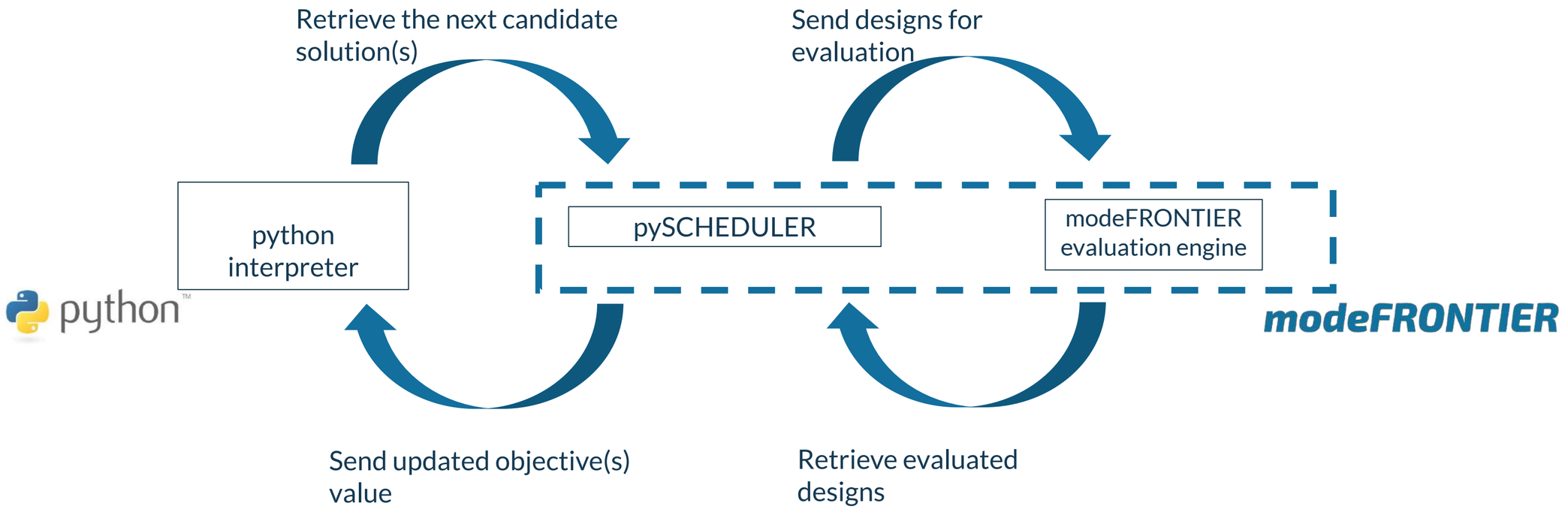
As alternative, in Optimization Algorithm module, any algo can be used in Autonomous mode, Self-initializing mode (just specify total number of design evaluations) or Manual mode (advanced settings)



pySCHEDULER

- You can drive your design space exploration sessions in modeFRONTIER with python scripts.
- A set of internal python APIs allows you to link your optimization algorithm or built-in python algorithms to the modeFRONTIER evaluation engine.
- Available both in the Scheduling mode and the Process mode.

pySCHEDULER – operating principle



pySCHEDULER bridges the modeFRONTIER evaluation engine and the external python interpreter. The optimization algorithm in the python interpreter does the math; pySCHEDULER sends evaluation requests to the modeFRONTIER evaluation engine, retrieves the evaluated design, and then sends updated objective values to the python interpreter, in which the algorithm determines the next candidate solution, which is then retrieved by pySCHEDULER.

pySCHEDULER – basic python code structure

```
...
from estecopy import scheduler
...
```

1. Import the estecopy modules

```
ctx = scheduler.get_optimization_context()
variable_names = ctx.get_variables()
doe = ctx.get_doe()
objectives = …
...
```

2. Get the optimization context from modeFRONTIER(inputs, outputs, objectives, doe designs, …)

```
def evaluate(point):
    design = ctx.receive_design(ctx.send_design(point))
    ...
    return objective_value
 ...
```

3. Define the function that asks modeFRONTIER for design evaluation and then returns the corresponding objective value

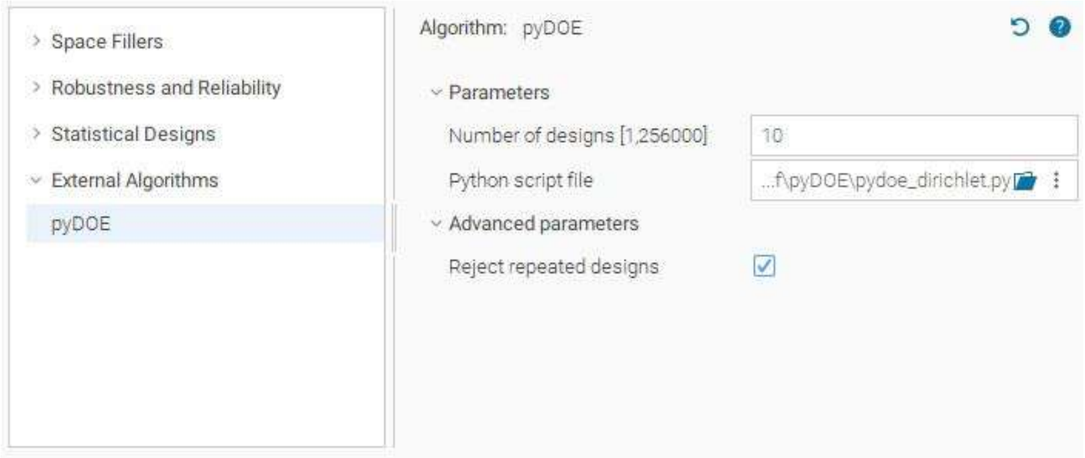
```
f_min_or_max = f_optimize(evaluate,[args],options)
```

4. Start the python optimizer

API documentation is available in the modeFRONTIER users guide. Full code examples are available in the modeFRONTIER installation folder ..*projects**external_schedulers**python*

pyDOE - New Python bridge for Design of Experiments

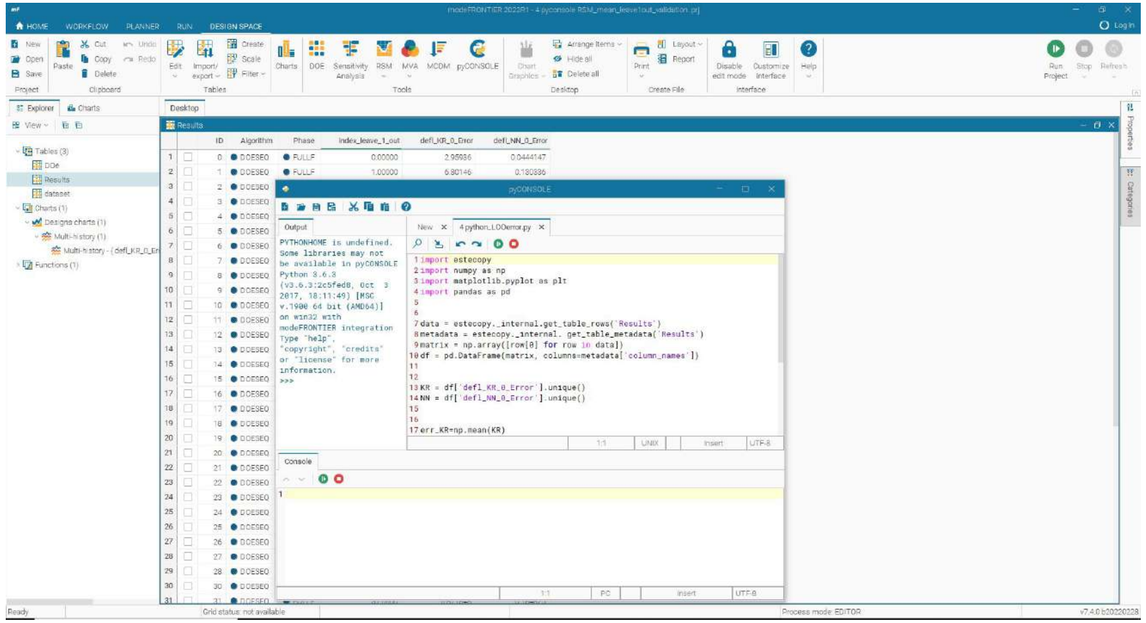
- You can drive your design space exploration sessions in modeFRONTIER with python scripts.
- A set of internal python APIs allows you to link your exploration algorithm or built-in python algorithms to the modeFRONTIER evaluation engine.
- Available both in the Scheduling mode and the Process mode.

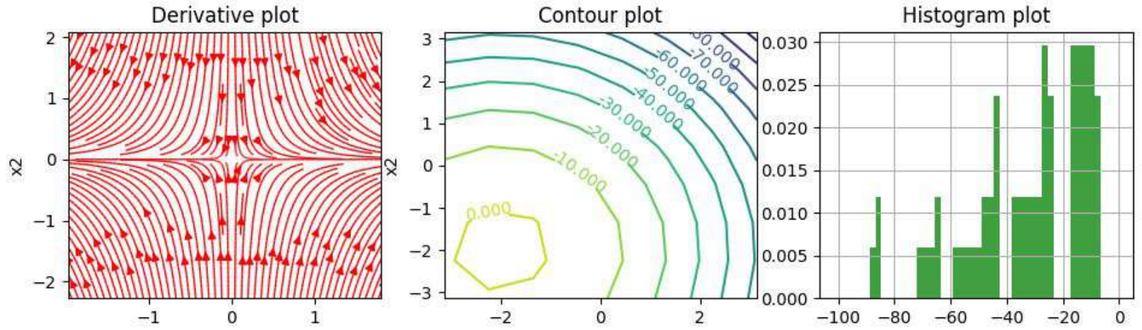


Design Space

- pyCONSOLE

pyCONSOLE

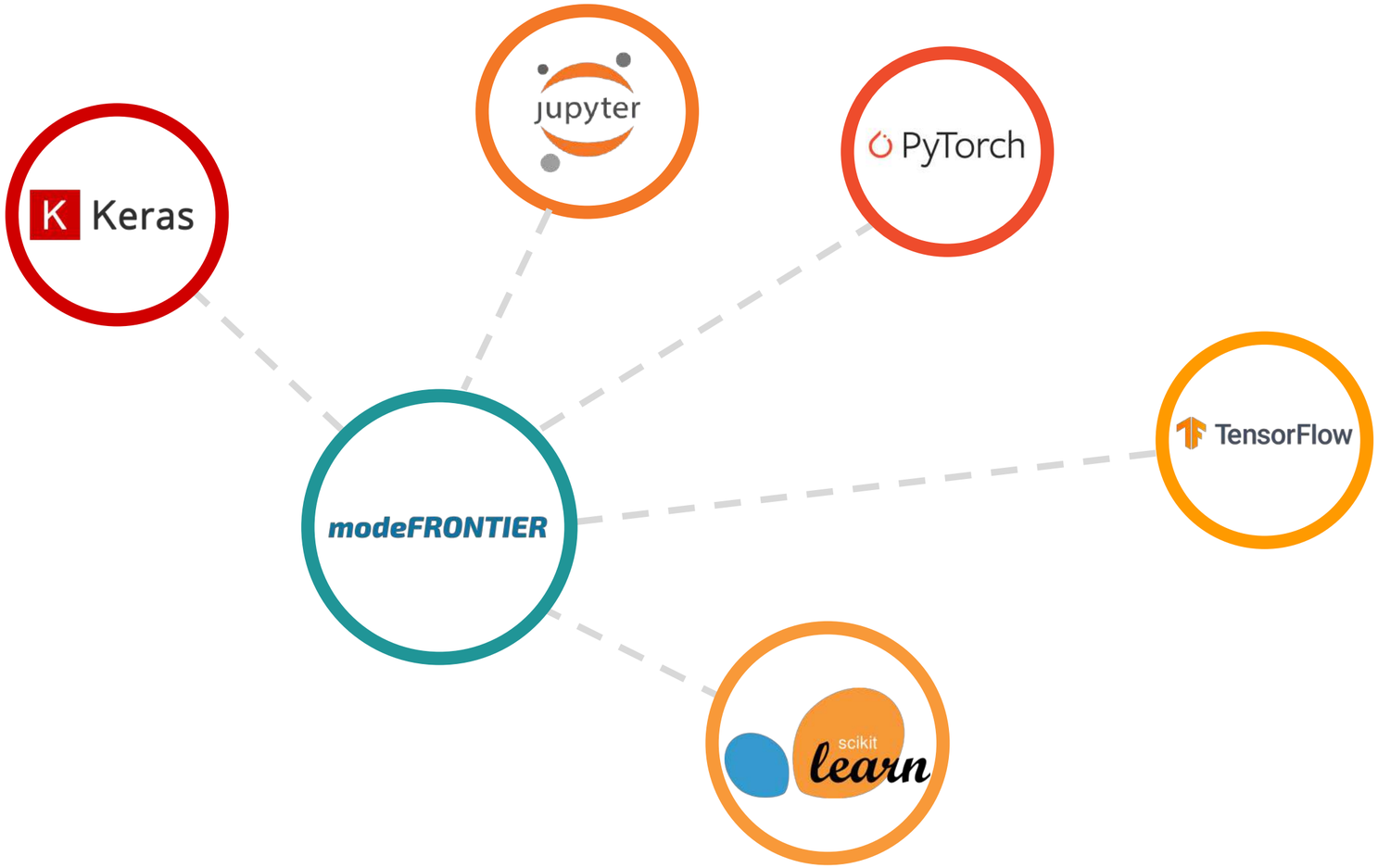




This Python-based console allows to apply customized Python script to automate the analysis and perform advanced postprocessing

Python eco-system in modeFRONTIER

- pyCONSOLE
- CPython node
  - pyRSM

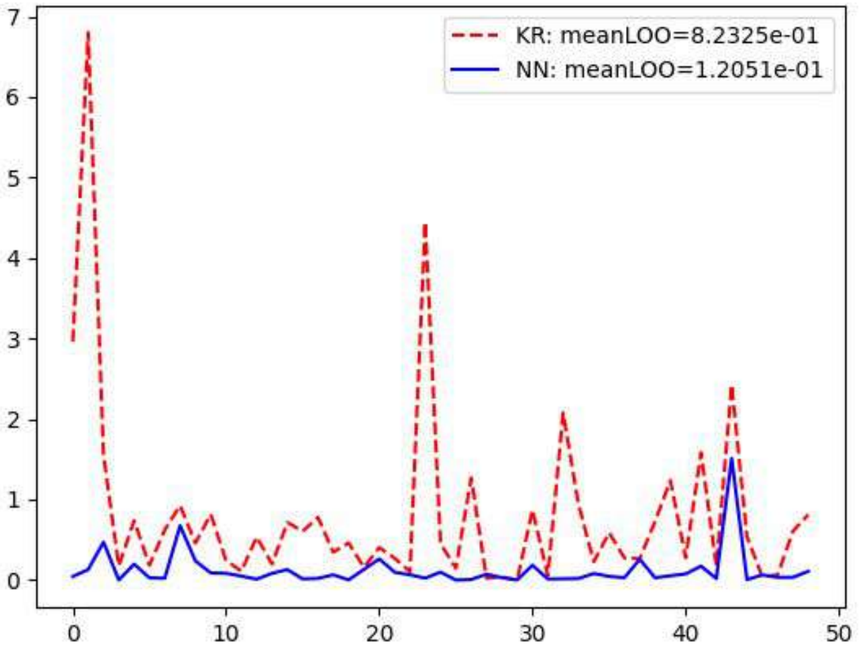


Post-processing by pyCONSOLE (example)

```
import estecopy
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd


data = estecopy.db.get_table('Results').get_rows()
header = data[0]
table = data[1:]
df = pd.DataFrame(np.array(table), columns=header)


KR = df['defl_KR_0_Error'].unique().astype(np.float)
NN = df['defl_NN_0_Error'].unique().astype(np.float)


err_KR=np.mean(KR)
err_NN=np.mean(NN)
print("mean_LOO for KR = ",err_KR)
print("mean_LOO for NN = ",err_NN)


#plot charts
lab_KR="KR: meanLOO={:.4e}".format(err_KR)
lab_NN="NN: meanLOO={:.4e}".format(err_NN)
plt.plot(KR,'--r',label=lab_KR)
plt.plot(NN,'-b',label=lab_NN)
plt.legend()
plt.show()
```

Results

		ID	Algorithm	Phase	index_leave_1_out	defl_KR_0_Error	defl_NN_0_Error
1		0	DOESEQ	FULLF	0.00000	2.95936	0.0444147
2		1	DOESEQ	FULLF	1.00000	6.80146	0.130336
3		2	DOESEQ	FULLF	2.00000	1.55243	0.471060
4		3	DOESEQ	FULLF	3.00000	0.174945	0.000909303
5		4	DOESEQ	FULLF	4.00000	0.737819	0.197907
6		5	DOESEQ	FULLF	5.00000	0.184788	0.0279074
7		6	DOESEQ	FULLF	6.00000	0.623490	0.0221190
8		7	DOESEQ	FULLF	7.00000	0.925284	0.675266
9		8	DOESEQ	FULLF	8.00000	0.460215	0.237680
10		9	DOESEQ	FULLF	9.00000	0.816082	0.0884612
11		10	DOESEQ	FULLF	10.0000	0.244392	0.0841068
12		11	DOESEQ	FULLF	11.0000	0.109338	0.0491662
13		12	DOESEQ	FULLF	12.0000	0.534639	0.0111210



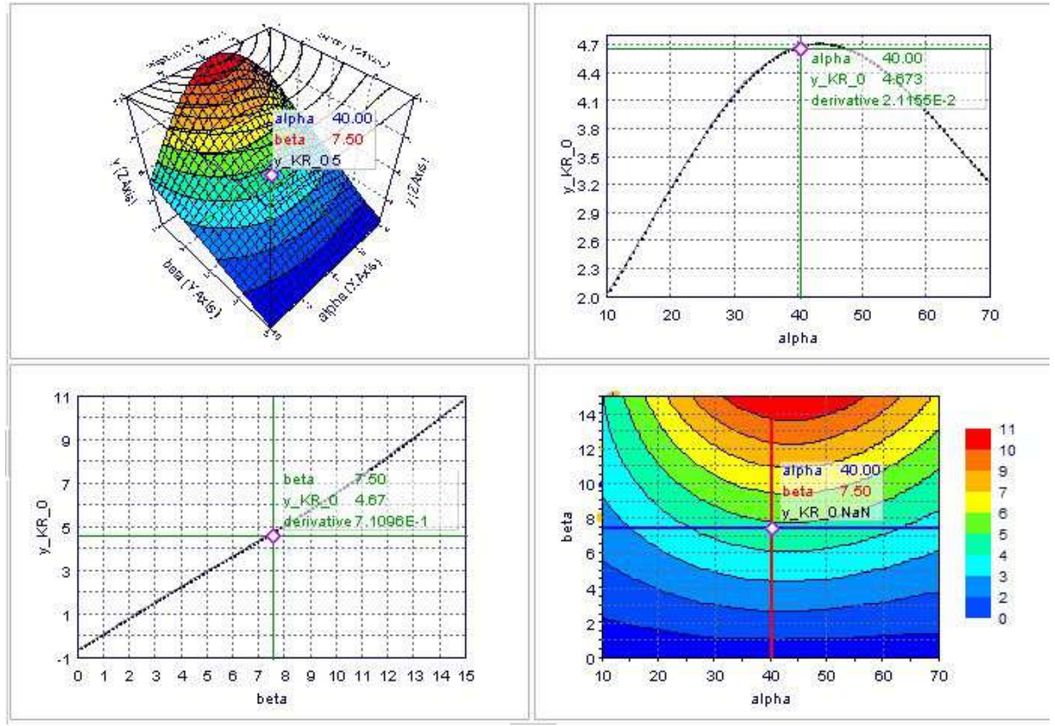
pyRSM – Train and evaluate

mF File Edit Options View Expression Editor

```
1 import numpy as np
2 from sklearn.gaussian_process import GaussianProcessRegressor
3 from sklearn.gaussian_process.kernels import DotProduct, WhiteKernel
4
5 def train(inputs_values, output_values, input_names, output_name):
6     X = np.array(inputs_values)
7     y = np.array(output_values)
8     kernel = DotProduct() + WhiteKernel()
9     return GaussianProcessRegressor(kernel=kernel,random_state=0).fit(X, y)
10
11 def evaluate(gpr, input_data):
12     predTest = np.array(input_data)
13     gprPred,stdPred=gpr.predict(predTest.reshape(3,-1), return_std=True)
14     return gprPred
```

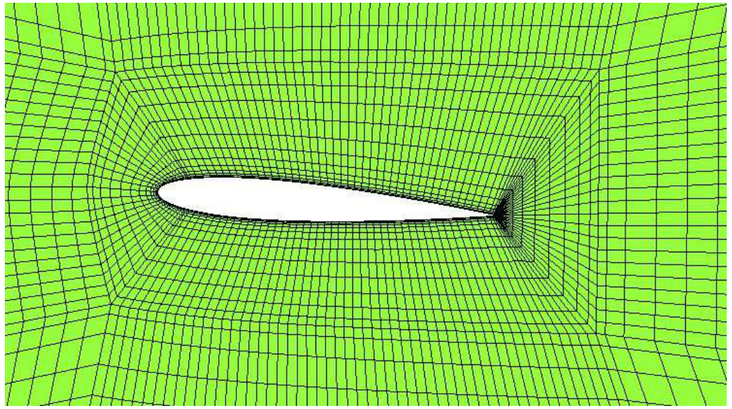
1:19 UNIX Insert windows-1252

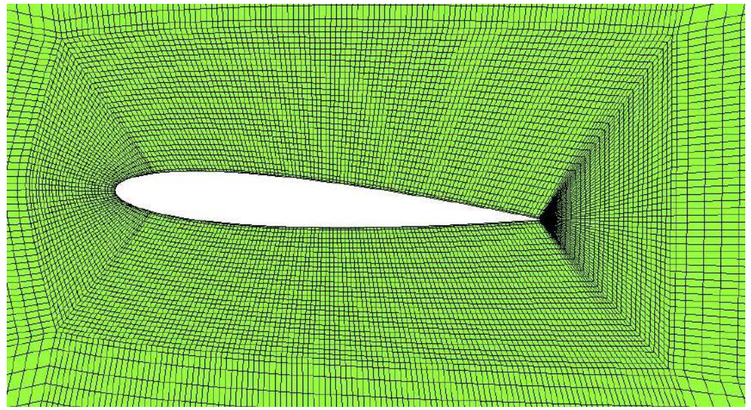
OK Cancel



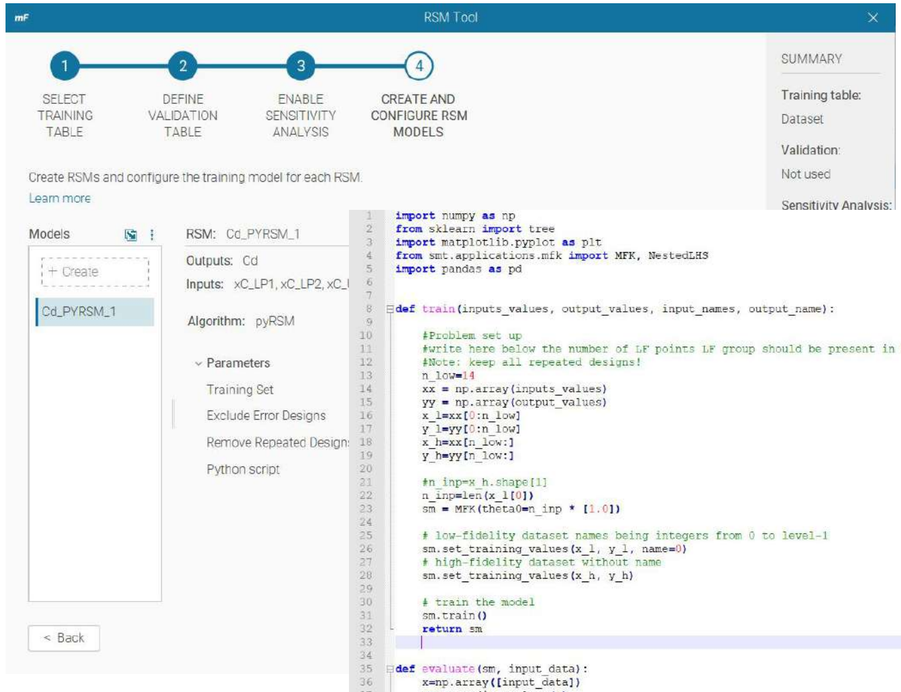
Training and evaluation features
as modeFRONTIER native RSMs

pyRSM: Multi-fidelity RSM (Cokriging)





- Low fidelity: 30,000 elements (50 samples)
- High fidelity: 150,000 elements (10 samples)




```python
import numpy as np
from sklearn import tree
import matplotlib.pyplot as plt
from smt.applications.mfk import MFK, NestedLHS
import pandas as pd

def train(inputs_values, output_values, input_names, output_name):

    #Problem set up
    #write here below the number of LF points LF group should be present in the Training table before the HF group.
    #Note: keep all repeated designs!
    n_low=14
    xx = np.array(inputs_values)
    yy = np.array(output_values)
    x_l=xx[0:n_low]
    y_l=yy[0:n_low]
    x_h=xx[n_low:]
    y_h=yy[n_low:]

    #n_inp=x_h.shape[1]
    n_inp=len(x_l[0])
    sm = MFK(theta0=n_inp * [1.0])

    # low-fidelity dataset names being integers from 0 to level-1
    sm.set_training_values(x_l, y_l, name=0)
    # high-fidelity dataset without name
    sm.set_training_values(x_h, y_h)

    # train the model
    sm.train()
    return sm

def evaluate(sm, input_data):
    x=np.array([input_data])
    y=sm.predict_values(x)
    #mse = sm.predict_variances(x)
    #derivs = sm.predict_derivatives(x, kx=0)
    return y
```

RSM comparison

Output	Name	Mean absolute error	Mean relative error	Mean normalized error	R-squared	AIC
Cd	Kriging_Random	3.86E-5	3.38E-3	7.19E-2	8.81E-1	-1.74E2
Cd	Kriging_DatasetReducer	2.57E-5	2.25E-3	4.79E-2	9.48E-1	-1.80E2
Cd	Cokriging_DatasetReducer	2.62E-5	2.28E-3	4.88E-2	9.61E-1	-2.09E2

High-level Roadmap

TODO

Test run in CAD-CAE nodes
Test nodes before running the workflow

RSM training in the Planner
Schedule RSM training in your plan

Connectors SDK
Create your own direct interfaces

DOING

Plan task node
Run multiple plans in the workflow

pyCONSOLE as a server
Drive pyCONSOLE from python application

New Algorithm
New self-adaptive optimizer

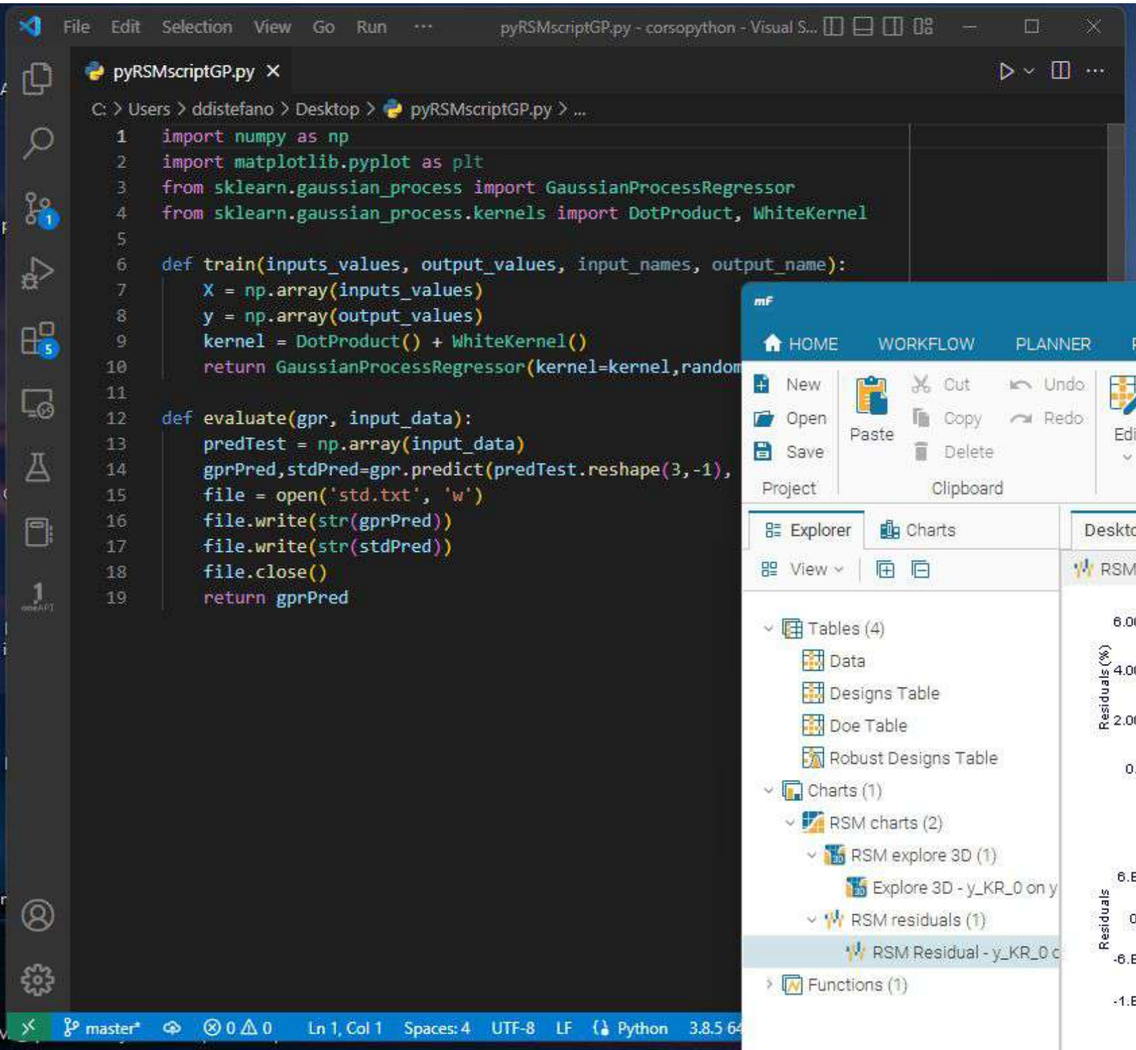
DONE

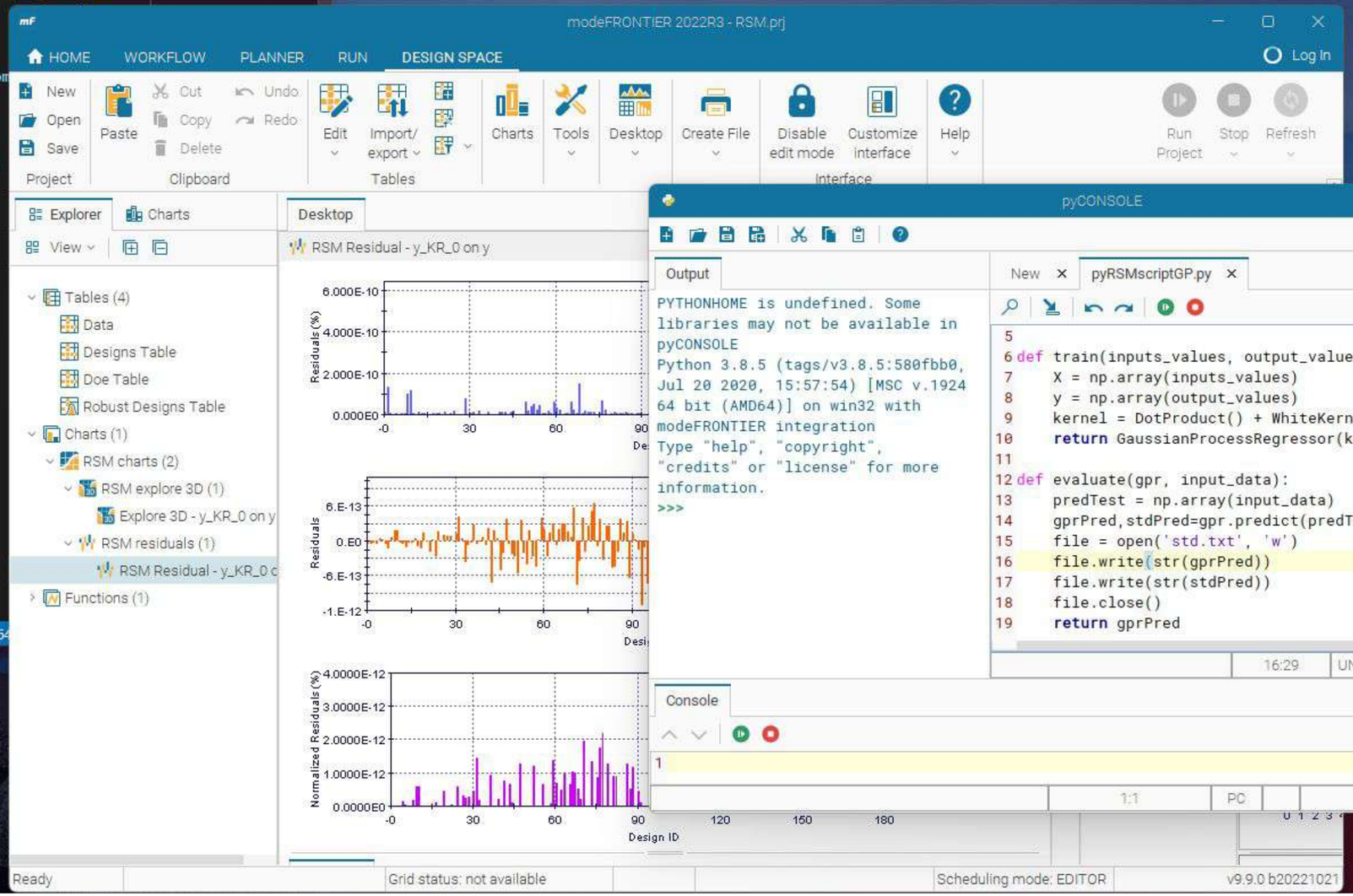
Python bridge for DoE
Run python DoE in modeFRONTIER

Future improvements

pyCONSOLE as a server

Driving modeFRONTIER from outside



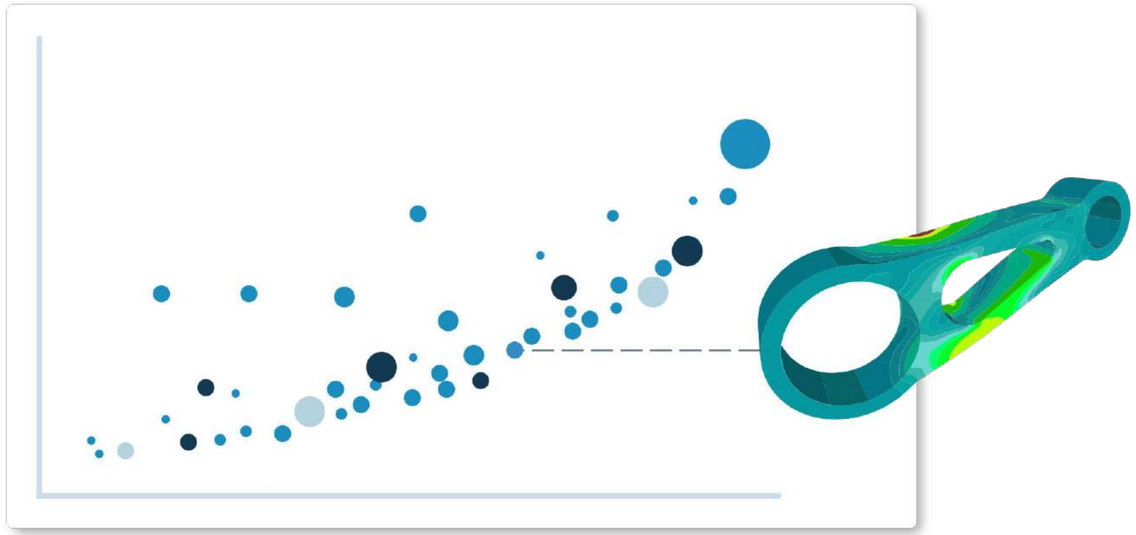


The evolution of pilOPT

This new optimization algorithm will build on the very successful pilOPT optimizer technology and will cover an even larger range of applications.

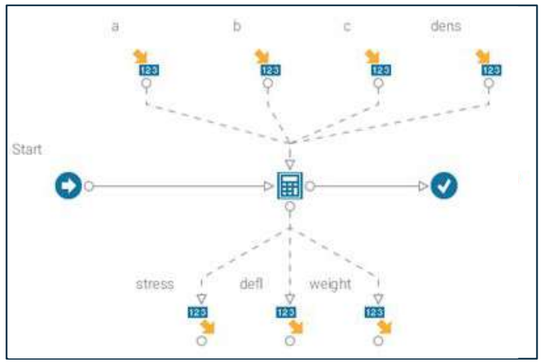
Focus on:

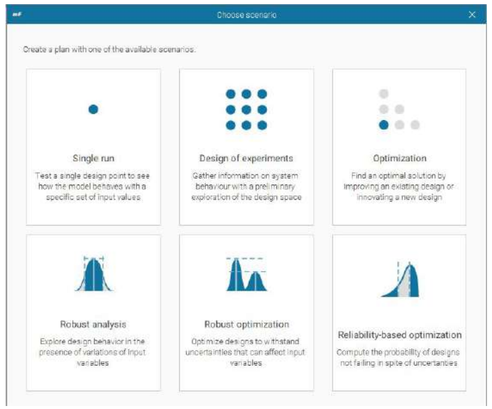
- High dimensional problems
- Intelligent use of existing information
- Effective Pause-Restart option



Plan task node

- This new node will make it possible for the programmatic execution of Plans in the context of modeFRONTIER workflow.
- Plans can be introspected and executed, also sequentially, in the workflow





Connectors SDK

A toolset to build your Connector

Python API

Introspection, input parsing, output generation

Build automation tool

Automatically validate and build the Connector

Documentation

Tutorials and reference material

Thank you!

esteco.com