A general cheat sheet for GEMSEO

Design space

Read a design space from a file and handle it:

```
from gemseo.api import read_design_space
design_space = read_design_space('file.txt')
design_space.filter(['x', 'y']) # Keep x & y variables
design_space.add_variable('z', l_b=-3, u_b=2)
del design_space['x']
print(design_space) # Pretty table view
```

Create a design space from scratch and handle it:

```
from gemseo.api import create_design_space
design_space = create_design_space()
design_space.add_variable('z', size=2, l_b=-3, u_b=2)
```

Export a design space to a text or HDF file:

```
from gemseo.api import export_design_space
export_design_space(design_space, 'file.txt')
```

Discipline

```
from gemseo.api import create_discipline
```

Instantiate a discipline from an internal or external module:

```
discipline = create_discipline('Sellar1')
```

Create a discipline from a python function:

```
def py_func(x=0., y=0.):
    z = x + 2*y
    return z
discipline = create_discipline('AutoPyDiscipline', py_func=py_func)
```

Create an analytic discipline from a dictionary of expressions.

```
expressions = {'y_1': '2*x**2', 'y_2': '5+3*x**2+z**3'}
discipline = create_discipline('AnalyticDiscipline', name='my_func',
                               expressions=expressions)
```

Coupling

Save or show the N2 chart:

```
from gemseo.api import generate_n2_plot
disciplines_names = ['disc1', 'disc2', 'disc3']
disciplines = create_discipline(disciplines_names, 'ext_path')
generate_n2_plot(disciplines, save=True, show=False)
```

Save the coupling graph:

```
from gemseo.api import generate_coupling_graph, create_discipline
disciplines = create_discipline(['disc1', 'disc2', 'disc3'])
generate_coupling_graph(disciplines)
```

Get all the inputs or outputs:

```
from gemseo.api import get_all_inputs, get_all_outputs
get_all_inputs(disciplines)
get_all_outputs(disciplines, recursive=True)
```

Surrogate discipline

Create a surrogate discipline from a dataset:

```
from gemseo.api import create_surrogate
surrogate = create_surrogate('LinearRegression', dataset)
```

Cache

```
from gemseo.api import create_discipline
discipline = create_discipline('disc')
```

Set the cache policy to store all executions:

```
discipline.set_cache_policy('HDF5Cache', cache_hdf_file='file.h5') # on
discipline.set_cache_policy('MemoryFullCache') # in memory
```

Set the simple cache policy to store the last execution in memory:

```
discipline.set_cache_policy('SimpleCache') # default option
```

Export cache to dataset:

```
dataset = discipline.cache.export_to_dataset()
```

Cache inputs and outputs in an HDF5 file:

```
from gemseo.core.cache import HDF5Cache
in_data = {'x':array([1.]), 'y':array([2.,3.])}
out_data = {'z': array([-6])}
cache = HDF5Cache('file.h5', 'node')
cache[in_data] = out_data
```

Get cached data:

```
last_entry = cache.last_entry
last_cached_inputs = last_entry.inputs
last_cached_outputs = last_entry.outputs
len(cache)
```

Get outputs and jacobian if data are cached, else None:

```
_, out_data, jac_data = cache[in_data]
```

Scenario

Instantiate an MDO or DOE scenario:

```
from gemseo.api import (create_scenario, create_discipline,
                        read_design_space)
disc_names = ['disc1', 'disc2', 'disc3']
disciplines = create_discipline(disc_names, 'ext_path')
d_space = read_design_space('file.txt')
scenario_type = 'MDO' # or 'DOE'
scenario = create_scenario(disciplines,
                           formulation='MDF',
                           objective_name='obj',
                           design_space=d_space,
                           name='my_scenario',
                           scenario_type=scenario_type,
                           **formulation_options)
scenario.add_constraint('cstr1', 'ineq') # <=0
scenario.add_constraint('cstr2', 'ineq', positive=True) # >=0
scenario.add_constraint('cstr3', 'ineq', value=1.) # <=1
scenario.add_constraint('cstr4', 'eq') # =0
scenario.xdsmize() # Build the XDSM graph to check it.
```

Execute the scenario:

```
scenario.execute({'algo': 'SLSQP', 'max_iter': 50,
                  algo_options={'xtol_rel':1e-3}}) # if MDO
scenario.execute({'algo': 'LHS', 'n_samples': 30}) # if DOE
optimum = scenario.get_optimum()
```

Save the optimisation history:

```
optimum_result = scenario.save_optimization_history('file.h5')
```

Visualization

Post-process the results from an MDO or DOE scenario:

```
from gemseo.api import execute_post
execute_post('OptHistoryView', scenario)
execute_post('OptHistoryView', optproblem)
execute_post('OptHistoryView', hdf_optproblem)
```

Availables

Get the available methods:

```
from gemseo.api import *
get_available_disciplines()
get_available_scenario_types()
get_available_formulations()
get_available_mdas()
get_available_opt_algorithms()
get_available_doe_algorithms()
get_available_post_processings()
get_available_surrogates()
```

New discipline

Create a new discipline from scratch:

```
from gemseo.api import MDODiscipline
from numpy import array

class NewDiscipline(MDODiscipline):

    def __init__(self):
        super(NewDiscipline, self).__init__()
        self.input_grammar.update(['x', 'z'])
        self.output_grammar.update(['f'])
        self.default_inputs = {'x': array([0.]), 'z': array([0.])}

    def _run(self):
        x, z = self.get_inputs_by_name(['x', 'z'])
        f = array([x[0]*z[0]])
        g = array([x[0]*(z[0]+1.)**2])
        self.store_local_data(f=f)
        self.store_local_data(g=g)

    def _compute_jacobian(self, inputs=None, outputs=None):
        self._init_jacobian(with_zeros=True)
        x, z = self.get_inputs_by_name(['x', 'z'])
        dfdx = z
        dfdz = x
        dgdx = array([(z[0]+1.)**2])
        dgdz = array([2*x[0]*z[0]*(z[0]+1.)])
        self.jac['f'] = {}
        self.jac['f']['x'] = atleast_2d(dfdx)
        self.jac['f']['z'] = atleast_2d(dfdz)
        self.jac['g'] = {}
        self.jac['g']['x'] = atleast_2d(dgdx)
        self.jac['g']['z'] = atleast_2d(dgdz)
```