Missile_4feats_pred_PlusSensitivity

January 16, 2023

[1]:
```
write_images = False
wirte_output_txt = False
```

[2]:
```
import numpy as np
```

[3]:
```
from emukit.core import ContinuousParameter, ParameterSpace
from emukit.core.initial_designs import RandomDesign

import GPy
from GPy.models import GPRegression
from emukit.model_wrappers import GPyModelWrapper
from emukit.sensitivity.monte_carlo import MonteCarloSensitivity

import matplotlib.pyplot as plt
import mlai.plot as plot
```

[4]:
```
%run Missile_utils_no_prints.ipynb
```

[ ]:
```

```

[5]:
```
simulation_output = 'range'
# We divide by 1000 to avoid dealing with too large numbers
```

[6]:
```
run_sensitivity_with_simulator = True
m4_evaluate = True
```

[ ]:
```

```

We consider missiles with only 1 stage

[7]:
```
basic_param_spaces = {
    'payload':  [10, 2410],
    'missilediam':  [0.1, 9.9],
    'rvdiam':  [0.1, 9.9],
    'estrange': [100, 4900],
    'fuelmass': [500, 6000], # [500, 7000],
    'drymass':  [1000, 3000],
    'Isp0':  [100, 800],# [100, 800],
    'thrust0':  [10000, 69000],
}
```

[8]:
```
from sklearn.metrics import mean_squared_error
import math
```

```python
def compute_rmse(y_actual, y_predicted):
    MSE = mean_squared_error(y_actual, y_predicted)
    RMSE = math.sqrt(MSE)

    return RMSE

def evaluate_prediction(y_actual, y_predicted):
    return compute_rmse(y_actual, y_predicted)
```

[ ]:

0.1 Sensitivity Analysis

[9]:
```python
rescale_0_1 = True

m4_param_1 = 'fuelmass'
m4_param_2 = 'Isp0'
m4_param_3 = 'drymass'
m4_param_4 = 'thrust0'
# m4_param_5 = 'payload'
# m4_param_6 = 'missilediam'

m4_domain_param_1 = basic_param_spaces[m4_param_1] # [500, 6000] # [5000,15000]
m4_domain_param_2 = basic_param_spaces[m4_param_2] # [200, 300] # [224, 228]
m4_domain_param_3 = basic_param_spaces[m4_param_3]
m4_domain_param_4 = basic_param_spaces[m4_param_4]
# m4_domain_param_5 = basic_param_spaces[m4_param_5]
# m4_domain_param_6 = basic_param_spaces[m4_param_6]

if rescale_0_1:
    m4_domain_param_1 = [0,1]
    m4_domain_param_2 = [0,1]
    m4_domain_param_3 = [0,1]
    m4_domain_param_4 = [0,1]
#     m4_domain_param_5 = [0,1]
#     m4_domain_param_6 = [0,1]



m4_space = ParameterSpace(
          [ContinuousParameter(m4_param_1, *m4_domain_param_1),
           ContinuousParameter(m4_param_2, *m4_domain_param_2),
           ContinuousParameter(m4_param_3, *m4_domain_param_3),
           ContinuousParameter(m4_param_4, *m4_domain_param_4),
#            ContinuousParameter(m4_param_5, *m4_domain_param_5),
#            ContinuousParameter(m4_param_6, *m4_domain_param_6),
          ])
custom_param_names = [m4_param_1, m4_param_2,
                      m4_param_3, m4_param_4,
#                       m4_param_5, m4_param_6
```

```
                    ]
nr_custom_params = len(custom_param_names)
```

[10]:
```
def run_missile_sim(custom_params):
    """
    Recives in input an array of custom parameters.
    Each row represents a set of different parameters
    Each column is a different parameter (#cols = len(custom_param_names))
    """
    default_params_IRAQ = {
        'payload':500,
        'missilediam':0.88,
        'rvdiam':0,
        'estrange':600,
        'numstages':1,
        'fuelmass':[0,5600],
        'drymass':[0,1200],
        'Isp0':[0,226],
        'thrust0':[0,9177.4]
    }


    y = np.zeros((custom_params.shape[0], 1))
    for i in range(custom_params.shape[0]):
        params_to_use = default_params_IRAQ
        # Overwrite default param variables
        for j in range(custom_params.shape[1]):
            param_name = custom_param_names[j]
            if param_name in ['fuelmass', 'drymass', 'Isp0', 'thrust0']:
                params_to_use[param_name][1] = custom_params[i,j]
            else:
                params_to_use[param_name] = custom_params[i, j]

            if j==0:
                print('\nNew simulation: i= \n', i)
            str_to_print = param_name + ': ' + str(custom_params[i,j])
            print(str_to_print)

        # Run simulation
        output_path = 'results/results_' + str(i) + '.txt'
        sim_output = run_one_sim(
            numstages=params_to_use["numstages"],
            fuelmass=params_to_use["fuelmass"],
            drymass=params_to_use["drymass"],
            thrust0=params_to_use["thrust0"],
            Isp0=params_to_use["Isp0"],
            payload=params_to_use["payload"],
            missilediam=params_to_use["missilediam"],
            rvdiam=params_to_use["rvdiam"],
            est_range=params_to_use["estrange"],
            output_path=output_path,
            simulation_output=simulation_output,
        )
```

```
        y[i, 0] = sim_output
    return y


if rescale_0_1:
    def scale_back_original(param_value, param_name):
        original_val_ranges = basic_param_spaces[param_name]
        return original_val_ranges[0] + param_value␣
↪*(original_val_ranges[1]-original_val_ranges[0])


    def run_missile_sim(custom_params):
        """
        Recives in input an array of custom parameters.
        Each row represents a set of different parameters
        Each column is a different parameter (#cols = len(custom_param_names))
        """
        default_params_IRAQ = {
            'payload':500,
            'missilediam':0.88,
            'rvdiam':0,
            'estrange':600,
            'numstages':1,
            'fuelmass':[0,5600],
            'drymass':[0,1200],
            'Isp0':[0,226],
            'thrust0':[0,9177.4]
        }


        y = np.zeros((custom_params.shape[0], 1))
        for i in range(custom_params.shape[0]):
            params_to_use = default_params_IRAQ
            # Overwrite default param variables
            for j in range(custom_params.shape[1]):
                param_name = custom_param_names[j]
                if param_name in ['fuelmass', 'drymass', 'Isp0', 'thrust0']:
                    params_to_use[param_name][1] =␣
↪scale_back_original(custom_params[i,j], param_name)
                else:
                    params_to_use[param_name] =␣
↪scale_back_original(custom_params[i,j], param_name)

                if j==0:
#                       print('\nNew simulation \n')
#                   str_to_print = param_name + ': ' +␣
↪str(scale_back_original(custom_params[i,j], param_name))
#                   print(str_to_print)
                    if i%10000==0:
                        print(i)

            # Run simulation
```

```
            output_path = 'results/results_' + str(i) + '.txt' # TODO Define better␣
↪identifier
            sim_output = run_one_sim(
                numstages=params_to_use["numstages"],
                fuelmass=params_to_use["fuelmass"],
                drymass=params_to_use["drymass"],
                thrust0=params_to_use["thrust0"],
                Isp0=params_to_use["Isp0"],
                payload=params_to_use["payload"],
                missilediam=params_to_use["missilediam"],
                rvdiam=params_to_use["rvdiam"],
                est_range=params_to_use["estrange"],
                output_path=output_path,
                simulation_output=simulation_output,
            )

            y[i, 0] = sim_output
        return y
```

[11]:
```
wirte_output_txt = False

if m4_evaluate:
    # Create grid
    nr_points_plot = 21
    m4_param_1_x_plot = np.linspace(m4_space.parameters[0].min, m4_space.parameters[0].
↪max, nr_points_plot)[:, None]
    m4_param_2_x_plot = np.linspace(m4_space.parameters[1].min, m4_space.parameters[1].
↪max, nr_points_plot)[:, None]
    m4_param_3_x_plot = np.linspace(m4_space.parameters[2].min, m4_space.parameters[2].
↪max, nr_points_plot)[:, None]
    m4_param_4_x_plot = np.linspace(m4_space.parameters[3].min, m4_space.parameters[3].
↪max, nr_points_plot)[:, None]

    m4_param_1_x_plot_mesh, m4_param_2_x_plot_mesh, m4_param_3_x_plot_mesh,␣
↪m4_param_4_x_plot_mesh = \
        np.meshgrid(m4_param_1_x_plot,
                    m4_param_2_x_plot,
                    m4_param_3_x_plot,
                    m4_param_4_x_plot)

    m4_x_plot = (np.array([m4_param_1_x_plot_mesh, m4_param_2_x_plot_mesh,
                           m4_param_3_x_plot_mesh, m4_param_4_x_plot_mesh])).T.
↪reshape(-1,4)
    print("Shape m4_x_plot: ", m4_x_plot.shape)
    # Compute simulated values
    # - Actually compute them (first time)
#     m4_y_plot = run_missile_sim(m4_x_plot) # TAKES LONG TIME
#     np.savetxt('m4_y_plot.txt', m4_y_plot, fmt='%f')

    # - Load the precomputed ones (after first time)
    m4_y_plot = np.loadtxt('m4_y_plot.txt', dtype=float)[:,None]
    print("Shape m4_y_plot: ", m4_y_plot.shape)
```

```
Shape m4_x_plot:  (194481, 4)
Shape m4_y_plot:  (194481, 1)
```

[ ]:

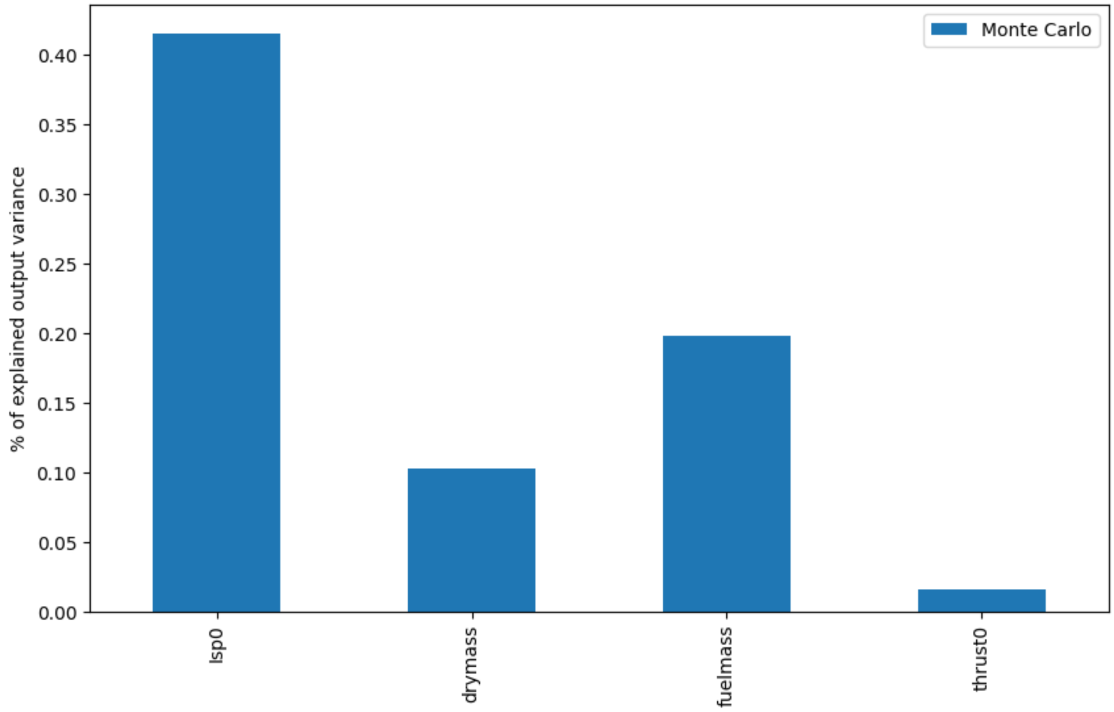
0.1.1 1. On the simulator

[12]:
```
class sim_model:
    def __init__(self):
        pass
    def scale_back_original(self,param_value, param_name):
        original_val_ranges = basic_param_spaces[param_name]
        return original_val_ranges[0] + param_value␣
↪*(original_val_ranges[1]-original_val_ranges[0])

    def run_missile_sim(self,custom_params):
        """
        Recives in input an array of custom parameters.
        Each row represents a set of different parameters
        Each column is a different parameter (#cols = len(custom_param_names))
        """
        default_params_IRAQ = {
            'payload':500,
            'missilediam':0.88,
            'rvdiam':0,
            'estrange':600,
            'numstages':1,
            'fuelmass':[0,5600],
            'drymass':[0,1200],
            'Isp0':[0,226],
            'thrust0':[0,9177.4]
        }


        y = np.zeros((custom_params.shape[0], 1))
        for i in range(custom_params.shape[0]):
            params_to_use = default_params_IRAQ
            # Overwrite default param variables
            for j in range(custom_params.shape[1]):
                param_name = custom_param_names[j]
                if param_name in ['fuelmass', 'drymass', 'Isp0', 'thrust0']:
                    params_to_use[param_name][1] = self.
↪scale_back_original(custom_params[i,j], param_name)
                else:
                    params_to_use[param_name] = self.
↪scale_back_original(custom_params[i,j], param_name)

                if j==0:
                    print('\nNew simulation \n')
                str_to_print = param_name + ': ' + str(self.
↪scale_back_original(custom_params[i,j], param_name))
                print(str_to_print)
                ##
```

```
            # Run simulation
            output_path = 'results/results_' + str(i) + '.txt'
            sim_output = run_one_sim(
                numstages=params_to_use["numstages"],
                fuelmass=params_to_use["fuelmass"],
                drymass=params_to_use["drymass"],
                thrust0=params_to_use["thrust0"],
                Isp0=params_to_use["Isp0"],
                payload=params_to_use["payload"],
                missilediam=params_to_use["missilediam"],
                rvdiam=params_to_use["rvdiam"],
                est_range=params_to_use["estrange"],
                output_path=output_path,
                simulation_output=simulation_output,
            )

            y[i, 0] = sim_output
        return y


    def predict(self,x):
        return (self.run_missile_sim(x), 0)


model = sim_model()
```

[13]:
```
# Long to run
wirte_output_txt = False

if run_sensitivity_with_simulator:
    num_mc = 1000 # Probably better to reduce
    senstivity = MonteCarloSensitivity(model = model, input_domain = m4_space)
    main_effects, total_effects, _ = senstivity.compute_effects(num_monte_carlo_points␣
 ↪= num_mc)
```

[14]:
```
import pandas as pd
if run_sensitivity_with_simulator:
    fig, ax = plt.subplots(figsize=plot.big_wide_figsize)

    main_effects_plot = {ivar: main_effects[ivar][0] for ivar in main_effects}

    d = {'Monte Carlo':main_effects_plot}

    pd.DataFrame(d).plot(kind='bar', ax=ax)
    plt.ylabel('% of explained output variance')

    if write_images:
        mlai.write_figure(filename='first-order-sobol-indices-missile.svg',␣
 ↪directory='./uq')
```

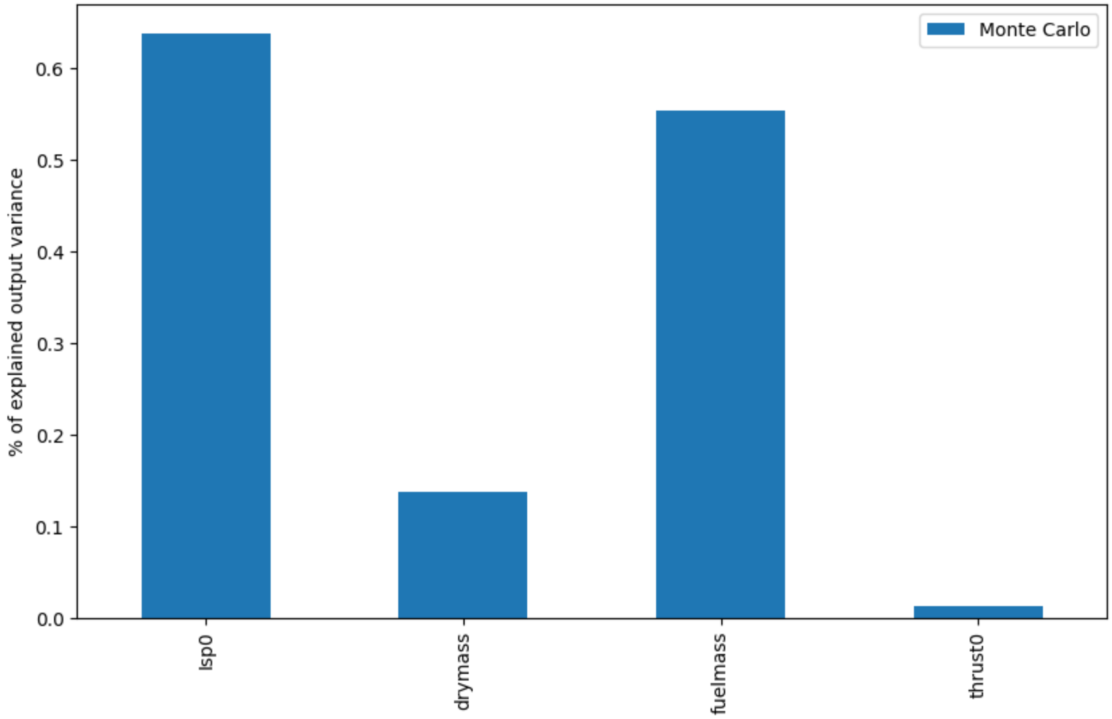


[15]:
```
if run_sensitivity_with_simulator:
    sob_sum = 0
    for var, sob_ind in main_effects.items():
        str_to_print = var + ' Sobol_index: ' + str(sob_ind[0])
        print(str_to_print)
        sob_sum += sob_ind[0]
    print('\n')
    print('Total sum: ', sob_sum)
```

```
fuelmass Sobol_index: 0.1980361789139976
Isp0 Sobol_index: 0.4154124616786544
drymass Sobol_index: 0.10302391606112503
thrust0 Sobol_index: 0.015654646647756742


Total sum:  0.7321272033015338
```

[16]:
```
if run_sensitivity_with_simulator:
    fig, ax = plt.subplots(figsize=plot.big_wide_figsize)

    total_effects_plot = {ivar: total_effects[ivar][0] for ivar in total_effects}

    d = {'Monte Carlo':total_effects_plot}

    pd.DataFrame(d).plot(kind='bar', ax=ax)
    ax.set_ylabel('% of explained output variance')

    if write_images:
```

```
        mlai.write_figure(filename='total-effects-sobol-indices-missile.svg',␣
↪directory='./uq')
```



[ ]:

```

```

0.1.2 2. On the emulator

Build emulator

[17]:

```
wirte_output_txt = True


# from emukit.core.initial_designs.latin_design import LatinDesign
# design = LatinDesign(parameter_space)

m4_design = RandomDesign(m4_space)
m4_x = m4_design.get_samples(3*4)
m4_y = run_missile_sim(m4_x)
```

[18]:

```
# Build model
m4_var_kernel = (100)**2
m4_lengthscale = 0.1 # 1
m4_var_noise = 1e-5 # small value

constrain_lengthscale = True

m4_rbf_kern = GPy.kern.RBF(input_dim=nr_custom_params, lengthscale=m4_lengthscale)
if constrain_lengthscale:
```

```
    m4_rbf_kern.lengthscale.constrain_bounded(m4_lengthscale, m4_lengthscale*1e12)

# m4_kern = m4_rbf_kern + \
#     GPy.kern.Linear(input_dim=nr_custom_params)
# m4_kern = (GPy.kern.RBF(input_dim=4, lengthscale=0.5) * \
#            GPy.kern.RBF(input_dim=4, lengthscale=0.1)) + \
#     GPy.kern.Linear(input_dim=nr_custom_params)
m4_kern = m4_rbf_kern + \
    GPy.kern.Linear(input_dim=nr_custom_params)


m4_model_gpy = GPRegression(m4_x,m4_y, kernel=m4_kern)
m4_model_gpy.kern.variance =  m4_var_kernel
m4_model_gpy.likelihood.variance.fix(m4_var_noise)

display(m4_model_gpy)
```

```
reconstraining parameters rbf.lengthscale
```

```
<GPy.models.gp_regression.GPRegression at 0x7f8f7ad03eb0>
```

[19]:
```
m4_model_emukit = GPyModelWrapper(m4_model_gpy)
m4_model_emukit.optimize()

display(m4_model_gpy)
```

```
<GPy.models.gp_regression.GPRegression at 0x7f8f7ad03eb0>
```

[20]:
```
if m4_evaluate:
    # Compute predictions through emulator
    m4_mu_plot_grid_pred, var_plot_grid_pred = m4_model_emukit.predict(m4_x_plot)

    m4_rmse = evaluate_prediction(y_actual=m4_y_plot, y_predicted=m4_mu_plot_grid_pred)
    print("RMSE m4 (pre experiment design loop): ", m4_rmse)
```

```
RMSE m4 (pre experiment design loop):  3042.9675231793794
```

[21]:
```
m4_2_model_emukit = m4_model_emukit
```

[22]:
```
# Experimental design to improve emulator
from emukit.experimental_design.acquisitions import IntegratedVarianceReduction,␣
 ↪ModelVariance
from emukit.experimental_design.experimental_design_loop import ExperimentalDesignLoop

wirte_output_txt = False

integrated_variance = IntegratedVarianceReduction(space=m4_space,
                                                  model=m4_2_model_emukit)
m4_ed = ExperimentalDesignLoop(space=m4_space,
                            model=m4_2_model_emukit,
                            acquisition = integrated_variance,
                            batch_size = 1)
# bach size is set to one in this example as we'll collect evaluations sequentially,
# but parallel evaluations are allowed
m4_ed.run_loop(user_function=run_missile_sim, stopping_condition=20)
```

```
/Users/ilariasartori/opt/anaconda3/envs/mlphysical/lib/python3.10/site-
packages/paramz/transformations.py:111: RuntimeWarning:overflow encountered in
expm1
```

```
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
```

[23]:
```
wirte_output_txt = False

if m4_evaluate:
    # Compute predictions through emulator
    m4_mu_plot_grid_pred2, var_plot_grid_pred = m4_2_model_emukit.predict(m4_x_plot)

    m4_2_rmse = evaluate_prediction(y_actual=m4_y_plot,␣
↪y_predicted=m4_mu_plot_grid_pred2)
    print("RMSE m4 (post first experiment design loop): ", m4_2_rmse)
```

```
RMSE m4 (post first experiment design loop):  1290.8813810774545
```

[ ]:
```

```

[ ]:
```

```

Run sensitivity

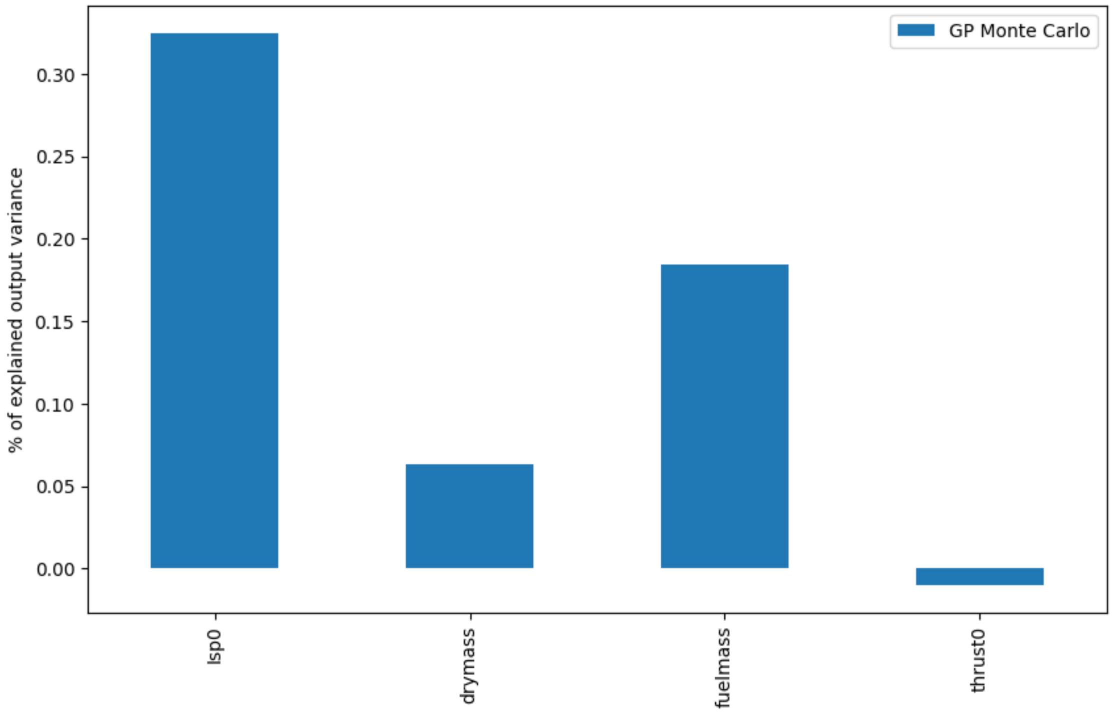
[24]:
```
num_mc = 1000
senstivity = MonteCarloSensitivity(model = m4_2_model_emukit, input_domain = m4_space)
main_effects_gp, total_effects_gp, _ = senstivity.
↪compute_effects(num_monte_carlo_points = num_mc)
```

[25]:
```
import matplotlib.pyplot as plt
import mlai.plot as plot
import mlai
```

[26]:
```
import pandas as pd
```

[27]:

```
fig, ax = plt.subplots(figsize=plot.big_wide_figsize)

main_effects_gp_plot = {ivar: main_effects_gp[ivar][0] for ivar in main_effects_gp}

d = {'GP Monte Carlo':main_effects_gp_plot}

pd.DataFrame(d).plot(kind='bar', ax=ax)
plt.ylabel('% of explained output variance')

if write_images:
    mlai.write_figure(filename='first-order-sobol-indices-gp-missile.svg', directory='.
 ↪/uq')
```



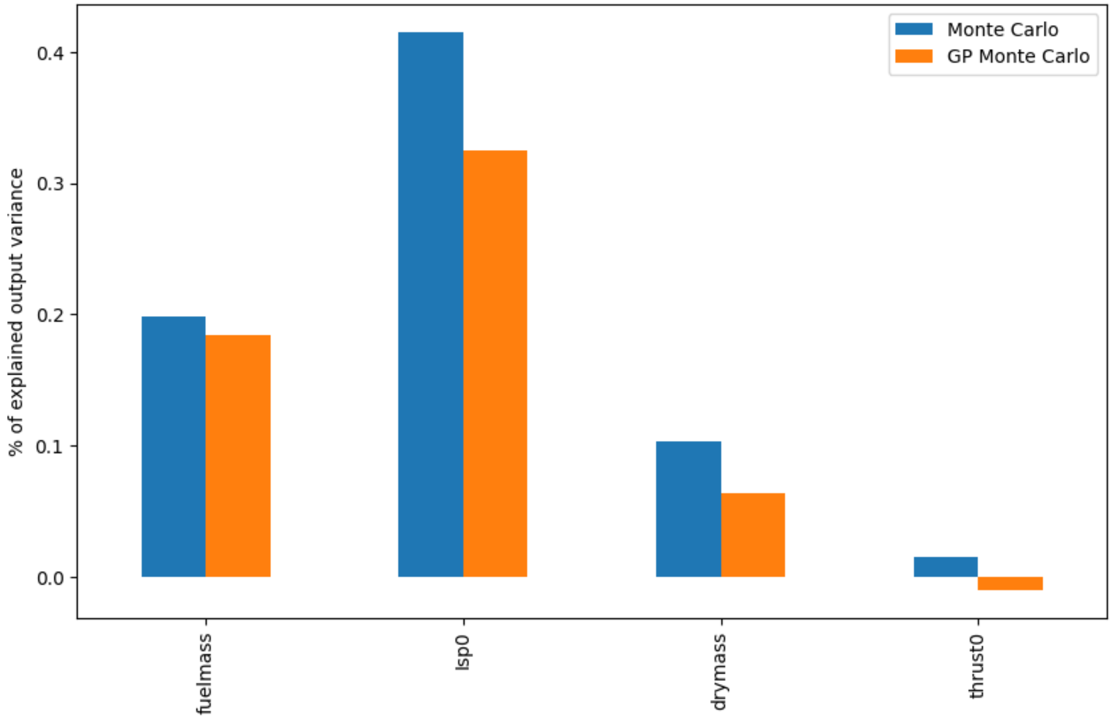
[ ]:

```
```

[28]:

```
if run_sensitivity_with_simulator:
    fig, ax = plt.subplots(figsize=plot.big_wide_figsize)

    main_effects_gp_plot = {ivar: main_effects_gp[ivar][0] for ivar in main_effects_gp}

    d = {'Monte Carlo': main_effects_plot,
         'GP Monte Carlo':main_effects_gp_plot}

    pd.DataFrame(d).plot(kind='bar', ax=ax)
    plt.ylabel('% of explained output variance')

    if write_images:
```

```
    mlai.write_figure(filename='first-order-sobol-indices-gp-missile.svg',␣
↪directory='./uq')
```



[29]:

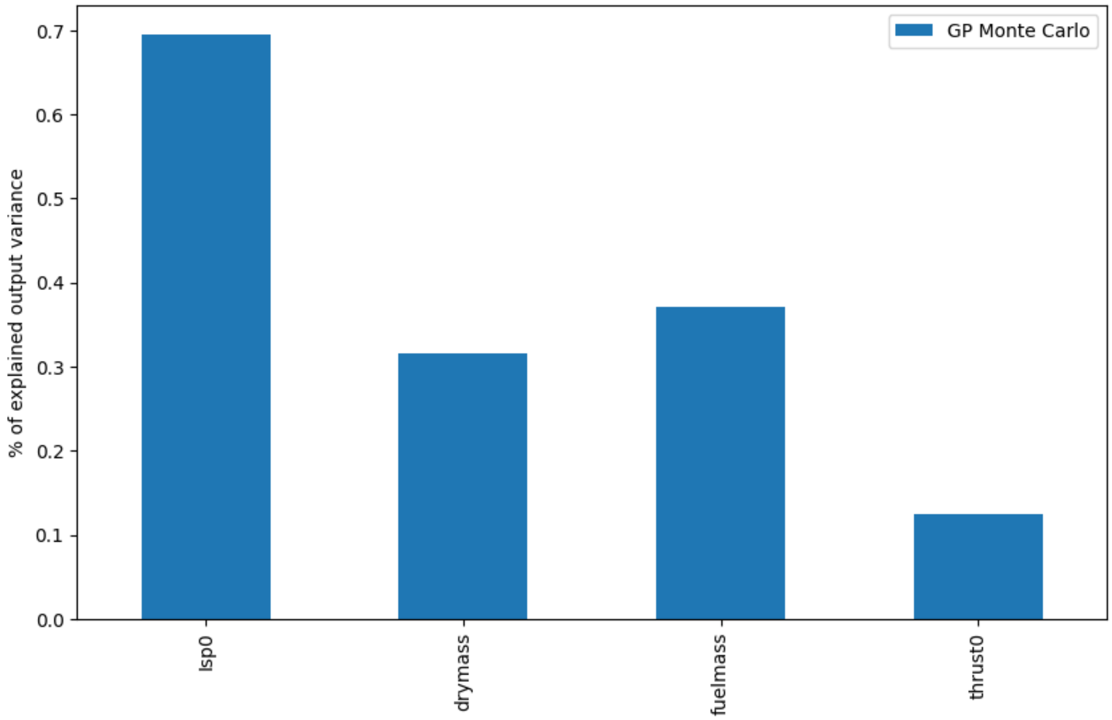
```
sob_sum = 0
for var, sob_ind in main_effects_gp.items():
    str_to_print = var + ' Sobol_index: ' + str(sob_ind[0])
    print(str_to_print)
    sob_sum += sob_ind[0]
print('\n')
print('Total sum: ', sob_sum)
```

```
fuelmass Sobol_index: 0.18419578072260942
Isp0 Sobol_index: 0.3248806974775271
drymass Sobol_index: 0.06354582916469083
thrust0 Sobol_index: -0.00990307577679336


Total sum:  0.562719231588034
```

[30]:

```
fig, ax = plt.subplots(figsize=plot.big_wide_figsize)

total_effects_gp_plot = {ivar: total_effects_gp[ivar][0] for ivar in total_effects_gp}

d = {'GP Monte Carlo':total_effects_gp_plot}

pd.DataFrame(d).plot(kind='bar', ax=ax)
ax.set_ylabel('% of explained output variance')
```

```
if write_images:
    mlai.write_figure(filename='total-effects-sobol-indices-gp-missile.svg',␣
 ↪directory='./uq')
```




```
[ ]:
```