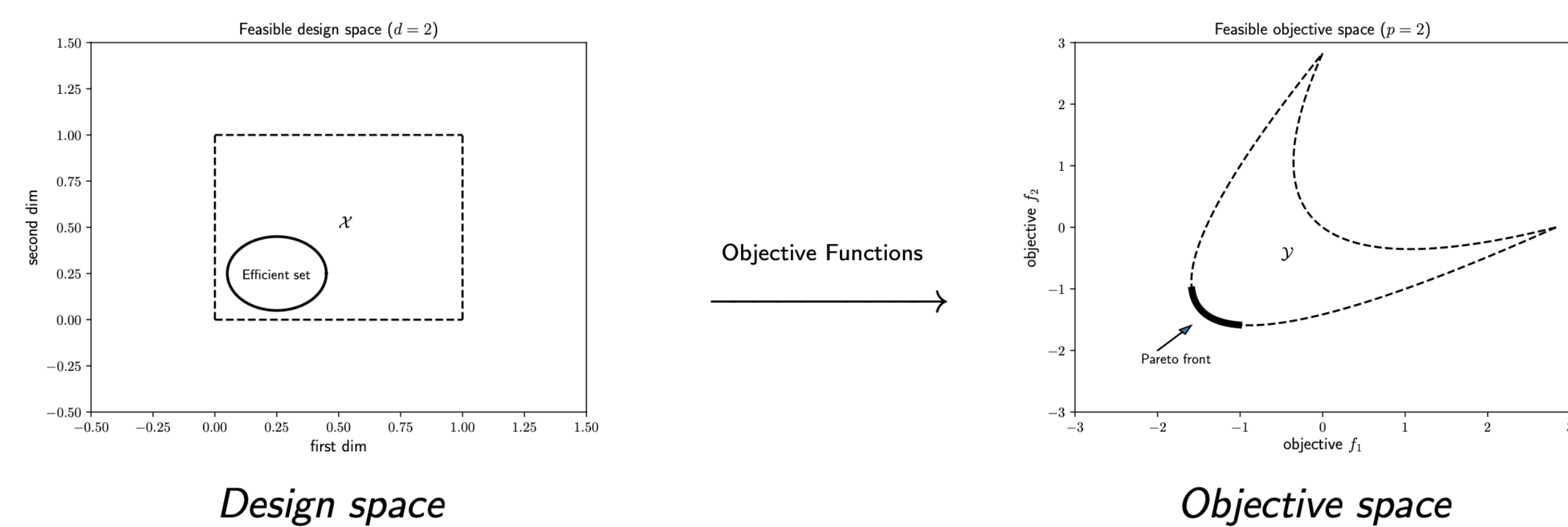


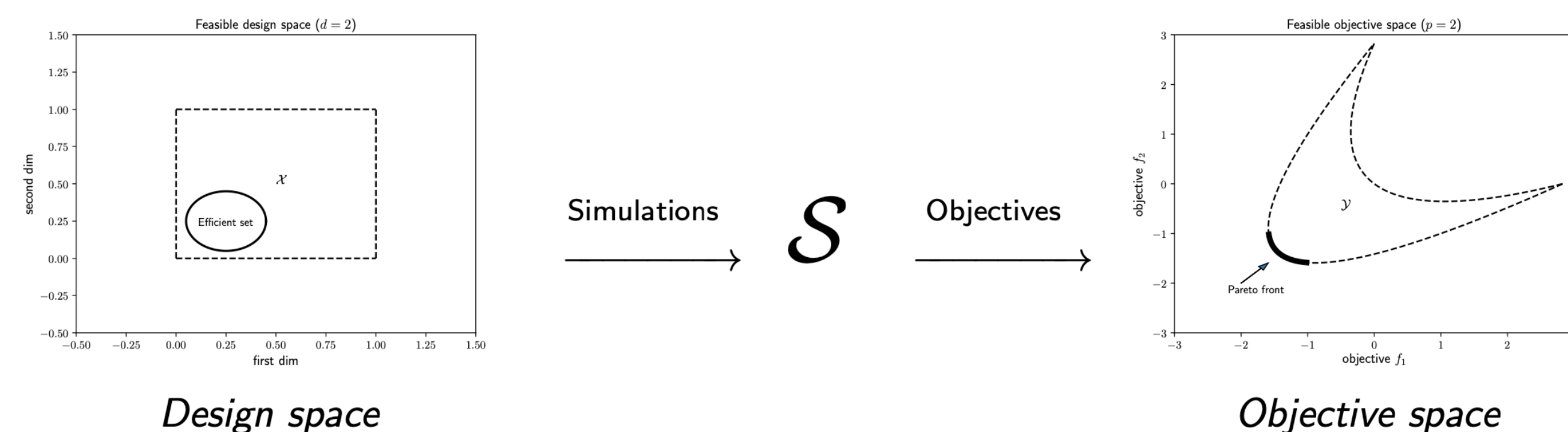
## Multiobjective Optimization

minimize  $F(\mathbf{x})$   $F(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \dots, f_o(\mathbf{x}))$   
subject to  $G(\mathbf{x}) \leq 0$   $G(\mathbf{x}) = (g_1(\mathbf{x}), g_2(\mathbf{x}), \dots, g_p(\mathbf{x}))$



## Multiobjective \*Simulation\* Optimization

minimize  $F(\mathbf{x}, \mathbf{S}(\mathbf{x}))$   $\mathbf{S}(\mathbf{x}) = (s_1(\mathbf{x}), s_2(\mathbf{x}), \dots, s_m(\mathbf{x}))$   
subject to  $G(\mathbf{x}, \mathbf{S}(\mathbf{x})) \leq 0$



## Common Simulation-based Structures

**Sum-of-squared simulation outputs:**

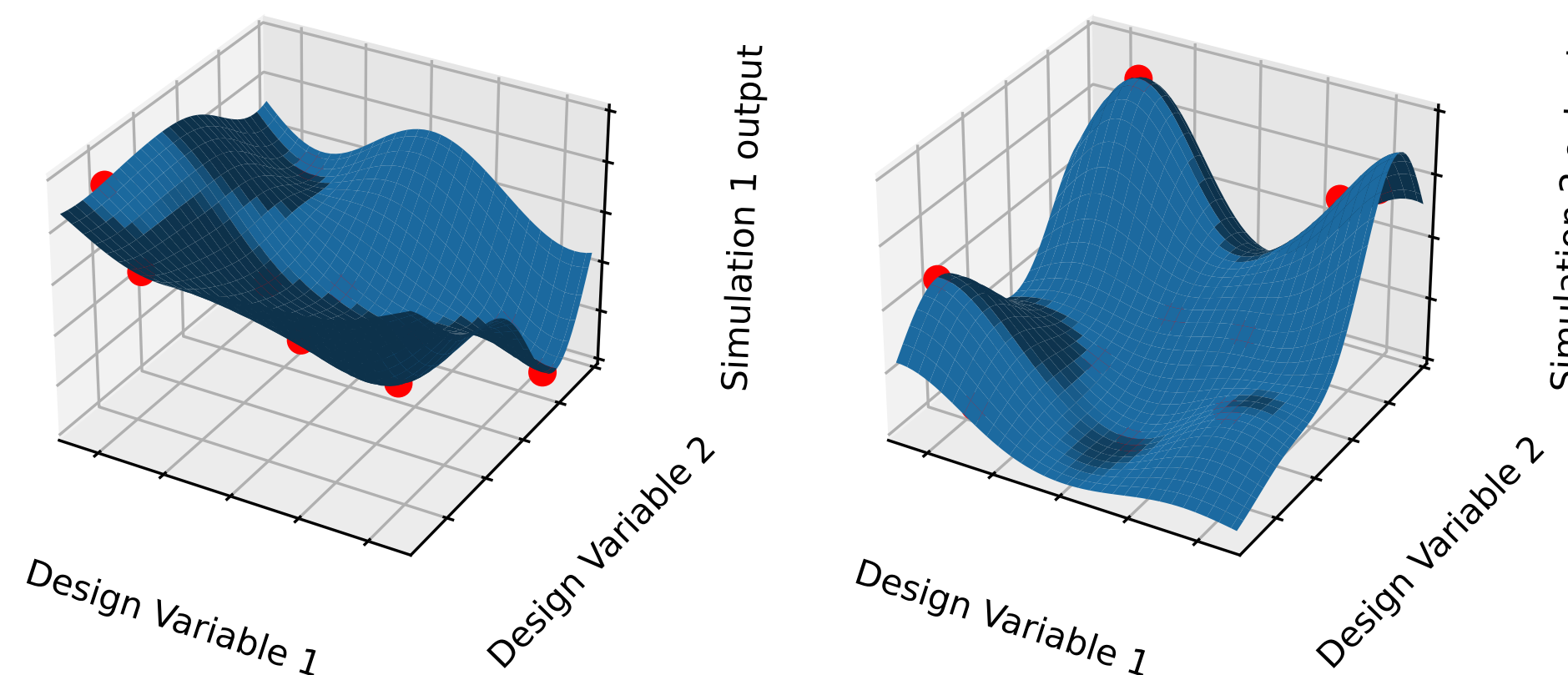
$$\min_{\mathbf{x} \in \mathcal{X}} \left( \sum_{i=1}^{m_1} S_i(\mathbf{x})^2, \sum_{j=1}^{m_2} S_j(\mathbf{x})^2 \right)$$

**One simulation, one algebraic objective:**

$$\min_{\mathbf{x} \in \mathcal{X}} \left( \mathbf{S}(\mathbf{x}), \sum_{i=1}^n \mathbf{x}^2 \right)$$

## Response Surface Methodology

- **Search/sample** data for raw **simulations outputs**
- Use **surrogates** to model **simulations**, **not objectives**
- Separately define **objectives and constraints**
- Scalarize **objectives** using **acquisition functions**
- **Solve scalarized surrogate problems** and iterate



## Design Principles

### Mix-and-match

- Initial search (design-of-experiments)
- Surrogate models
- Acquisition/scalarization functions
- Scalar optimization solvers

### Easy for users and developers

- Support for variety of design vars and simulations
- Support various scientific workflows
- Embed/extract problems from unit cube

### Flexible problem definitions

- Add design vars, sims, objs, + constraints
- Add searches, surrogates, acquisitions, optimizer
- Solve serially or in parallel using libEnsemble

## A Sample Script

```
import numpy as np
from parmoo import MOOP, optimizers, searches, \
    surrogates, acquisitions

# Create a MOOP object with LocalGPS solver
moop = MOOP(optimizers.LocalGPS)

# Add 2 design variables
moop.addDesign({"name": "x1", "des_type": "continuous",
    "lb": 0.0, "ub": 1.0})
moop.addDesign({"name": "x2", "des_type": "categorical",
    "levels": 3})

# Define and add 1 simulation with 2 outputs
def sim(x): return np.array([x["x1"]**2 + x["x2"],
    (x["x1"] - 1)**2 + x["x2"]])

moop.addSimulation({"name": "MySim", "m": 2,
    "sim_func": sim, "search": searches.LatinHypercube,
    "surrogate": surrogates.GaussRBF})

# Add 2 objectives: minimize each sim output
def f1(x, s): return s["MySim"][0]
def f2(x, s): return s["MySim"][1]
moop.addObjective({"name": "f1", "obj_func": f1})
moop.addObjective({"name": "f2", "obj_func": f2})

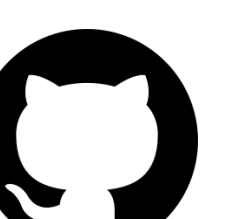
# Add 1 constraint: x["x1"] <= 0.1
def g1(x, s): return 0.1 - x["x1"]
moop.addConstraint({"name": "g", "constraint": g1})

# Add 3 acquisition functions
for i in range(3):
    moop.addAcquisition({"acquisition":
        acquisitions.UniformWeights})

# Solve with 5 iterations and print the solution
moop.solve(5)
print(moop.getPF())
```

## Download ParMOO

- git clone <https://github.com/parmoo/parmoo>
- pip install parmoo



## Continuing Work

- Continue to add new solvers and techniques
- Support wider variety of problems & workflows