ex06-solution

June 27, 2021

1 Gaussian Process Regression

[1]:
```
# !pip install Optunity GPy sobol sobol-seq
```

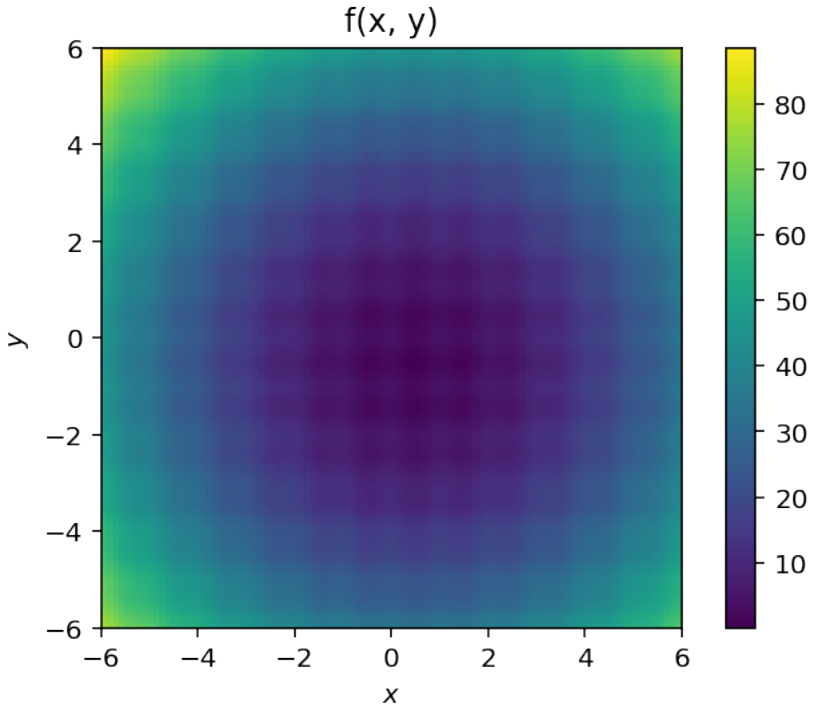
[2]:
```
import math
import pprint
import time
from collections import OrderedDict
import inspect
from IPython.display import Markdown

import GPy
import matplotlib.pyplot as plt
import numpy as np
import optunity
import optunity.metrics
import scipy
import scipy.linalg
import scipy.sparse
import scipy.sparse.linalg
import sobol
```

1.1 Exercise 1

[3]:
```
# 1.1: Implementation of toy function
def ras(x0, x1):
    return x0 * x0 - x0 + x1 * x1 + x1 - (np.cos(2 * math.pi * (x0 - .5)) + np.
 ↪cos(2 * math.pi * (x1 + .5))) + 2.5
```

[4]:
```
# 1.1: Plotting the toy function f which has local minima on a regular grid.

def plot_ras(x_range, y_range):
    X_ = np.linspace(*x_range, 100)
    Y_ = np.linspace(*y_range, 100)
    X_, Y_ = np.meshgrid(X_, Y_)
    Z = ras(X_, Y_)
```

```
    plt.imshow(Z, origin="lower", extent=(*x_range, *y_range))
    plt.colorbar()
    plt.gca().set_aspect(1)
    plt.xlabel("$x$")
    plt.ylabel("$y$")
    plt.title(f"f(x, y)")


x_range = y_range = (-6, 6)
plot_ras(x_range, y_range)
```



[5]:
```
# 1.1: Value at random points

rand_points = np.random.rand(60, 2)
values = ras(rand_points[:, 0], rand_points[:, 1])
desc = scipy.stats.describe(values)

Markdown(f"""Statistics of 60 random values from $f$:
- Minimum: {desc.minmax[0]:6.3f}
- Maximum: {desc.minmax[1]:6.3f}
- Mean:    {desc.mean:6.3f} ± {np.sqrt(desc.variance):6.3f}""")
```

[5]:
Statistics of 60 random values from f: - Minimum: 0.964 - Maximum: 6.383 - Mean: 3.293 ± 1.409

[6]:
```
# 1.3+1.4: Choose 30 initial points from a sobol sequence and compute f

def sobol_sample(count, limits, fn, offset=0):
    dim = len(limits)
    limits = np.array(limits)
    lower = limits[:, 0]
    upper = limits[:, 1]

    x = []
    y = []
    for i in range(count):
        new_x = sobol.i4_sobol(dim, i + offset)[0] * (upper - lower) + lower
        print("Point " + ", ".join(f"{v:6.2f}" for v in new_x), end=" ... ")
        x.append(new_x)
        start = time.time()
        new_y = fn(*new_x)
        print(f"{new_y:6.2f} in {time.time() - start:7.1f}s")
        y.append(new_y)

    return np.array(x), np.array(y).reshape(-1, 1)

toy_initialX, toy_initialY = sobol_sample(30, [x_range, y_range], ras)
```

```
Point  -6.00,  -6.00 …  76.50 in     0.0s
Point   0.00,   0.00 …   4.50 in     0.0s
Point   3.00,  -3.00 …  16.50 in     0.0s
Point  -3.00,   3.00 …  28.50 in     0.0s
Point  -1.50,  -1.50 …   5.00 in     0.0s
Point   4.50,   4.50 …  41.00 in     0.0s
Point   1.50,  -4.50 …  17.00 in     0.0s
Point  -4.50,   1.50 …  29.00 in     0.0s
Point  -3.75,  -2.25 …  23.12 in     0.0s
Point   2.25,   3.75 …  23.12 in     0.0s
Point   5.25,  -5.25 …  47.13 in     0.0s
Point  -0.75,   0.75 …   5.12 in     0.0s
Point  -2.25,  -3.75 …  20.13 in     0.0s
Point   3.75,   2.25 …  20.13 in     0.0s
Point   0.75,  -0.75 …   2.12 in     0.0s
Point  -5.25,   5.25 …  68.12 in     0.0s
Point  -4.88,  -0.38 …  30.91 in     0.0s
Point   1.12,   5.62 …  39.91 in     0.0s
Point   4.12,  -3.38 …  23.41 in     0.0s
Point  -1.88,   2.62 …  17.41 in     0.0s
Point  -0.38,  -4.88 …  21.91 in     0.0s
Point   5.62,   1.12 …  30.91 in     0.0s
Point   2.62,  -1.88 …   8.41 in     0.0s
Point  -3.38,   4.12 …  38.41 in     0.0s
```

```
Point  -4.12,  -4.12 …  37.95 in     0.0s
Point   1.88,   1.88 …  10.95 in     0.0s
Point   4.88,  -1.12 …  22.95 in     0.0s
Point  -1.12,   4.88 …  34.95 in     0.0s
Point  -2.62,  -2.62 …  14.87 in     0.0s
Point   3.38,   3.38 …  23.87 in     0.0s
```

[7]:
```
toy_ker = GPy.kern.RBF(2) + GPy.kern.White(2)
```

[8]:
```
# See 1.2

def expected_imp(model, x):
    pred, var = model.predict_noiseless(x)
    stddev = np.sqrt(var)
    fmin = np.min(model.Y)
    gamma = (fmin - pred) / stddev
    return stddev * (gamma * scipy.stats.norm.cdf(gamma) + scipy.stats.norm.
 ↪pdf(gamma))
```

[9]:
```
# Different utility functions for the toy problem

# Expected improvement
def toy_utility_ei(x0, x1, m):
    return expected_imp(m, np.array([[x0, x1]]))


# Variance of the GP. Maximizing this selects the point where the GP is "most␣
 ↪unsure"
# and leads to exploration of the optimization domain
def toy_utility_var(x0, x1, m):
    pred, var = m.predict_noiseless(np.array([[x0, x1]]))
    return var


# Negative Prediction of the GP. Maximizing this minimizes the prediction.
# This might be useful as a last step, but wasn't required in the exercise.
def toy_utility_f(x0, x1):
    pred, var = m.predict_noiseless(np.array([[x0, x1]]))
    return -pred
```

[10]:
```
# 1.5 - 1.8

def optim(initialX, initialY, fn, kernel, utilities, iterations, ranges,␣
 ↪max_f_eval=2000):
    optimizedX = initialX.copy()
    optimizedY = initialY.copy()
```

```
    optimization_log = {"prediction": [], "result": [], "best": [], "worst": []}
    optimization_params = []
    # 100 iterations show that the process actually gets stuck, the exercise␣
↪only required 30.
    for i in range(iterations):
        print(f"Iteration: {i:3d}:", end="" if len(utilities) <= 1 else "\n")
        for utility in utilities:
            # 1.5:
            m = GPy.models.GPRegression(optimizedX, optimizedY, kernel)
            m.optimize(messages=False, max_f_eval=max_f_eval)

            # Add "m" argument to utility
            if "m" in inspect.getargspec(utility).args:
                opt_fun = lambda **kwargs: utility(m=m, **kwargs)
            else:
                opt_fun = utility

            # 1.6: Use optunity to maximize the expected improvement
            optimal_par_dict, info, _ = optunity.maximize(opt_fun,␣
↪num_evals=max_f_eval, **ranges)
            optimal_pars = np.array([[optimal_par_dict[key] for key in ranges]])
            optimizedX = np.concatenate((optimizedX, optimal_pars))

            # 1.7: Evaluate f at the new chosen point
            new_target = fn(**optimal_par_dict)
            optimizedY = np.concatenate((optimizedY, np.array([[new_target]])))

            print(
                f" {utility.__name__:>16} -> new value {new_target:6.3f} @ " +
                ", ".join(f"{key}: {value:6.3f}" for key, value in␣
↪optimal_par_dict.items()) +
                "."
            )
            prediction, var = m.predict_noiseless(optimal_pars)
            bestValue = np.min(optimizedY)
            worstValue = np.max(optimizedY)

            optimization_log["prediction"].append(prediction)
            optimization_log["result"].append(new_target)
            optimization_log["best"].append(bestValue)
            optimization_log["worst"].append(worstValue)
            optimization_params.append(optimal_pars)

    return m, optimizedX, optimizedY, optimization_log, optimization_params
```

[11]:
```
(
    toy_m,
    toy_optimizedX, toy_optimizedY,
    toy_optimization_log, toy_optimization_params
) = optim(
    toy_initialX, toy_initialY, ras,
    toy_ker, [toy_utility_ei], 100, OrderedDict([
        ("x0", x_range),
        ("x1", y_range)
    ])
)
```

```
Iteration:   0:   toy_utility_ei -> new value  0.342 @ x0:  0.401, x1: -0.585.
Iteration:   1:
```

```
 /home/felix/miniconda3/envs/optimal-transport/lib/python3.7/site-
packages/paramz/transformations.py:111: RuntimeWarning:overflow encountered in
expm1
```

```
   toy_utility_ei -> new value  0.380 @ x0:  0.390, x1: -0.582.
Iteration:   2:   toy_utility_ei -> new value  0.323 @ x0:  0.409, x1: -0.588.
Iteration:   3:   toy_utility_ei -> new value  0.280 @ x0:  0.402, x1: -0.565.
Iteration:   4:   toy_utility_ei -> new value  0.396 @ x0:  0.388, x1: -0.585.
Iteration:   5:   toy_utility_ei -> new value  0.207 @ x0:  0.419, x1: -0.560.
Iteration:   6:   toy_utility_ei -> new value  0.196 @ x0:  0.433, x1: -0.572.
Iteration:   7:   toy_utility_ei -> new value  0.291 @ x0:  0.405, x1: -0.573.
Iteration:   8:   toy_utility_ei -> new value  0.250 @ x0:  0.413, x1: -0.569.
Iteration:   9:   toy_utility_ei -> new value  0.325 @ x0:  0.422, x1: -0.600.
Iteration:  10:   toy_utility_ei -> new value  0.330 @ x0:  0.408, x1: -0.589.
Iteration:  11:   toy_utility_ei -> new value  0.253 @ x0:  0.424, x1: -0.582.
Iteration:  12:   toy_utility_ei -> new value  0.242 @ x0:  0.415, x1: -0.568.
Iteration:  13:   toy_utility_ei -> new value  0.314 @ x0:  0.420, x1: -0.595.
Iteration:  14:   toy_utility_ei -> new value  0.302 @ x0:  0.417, x1: -0.590.
Iteration:  15:   toy_utility_ei -> new value  0.365 @ x0:  0.395, x1: -0.585.
Iteration:  16:   toy_utility_ei -> new value  0.202 @ x0:  0.419, x1: -0.558.
Iteration:  17:   toy_utility_ei -> new value  0.343 @ x0:  0.395, x1: -0.578.
Iteration:  18:   toy_utility_ei -> new value  0.223 @ x0:  0.433, x1: -0.581.
Iteration:  19:   toy_utility_ei -> new value  0.186 @ x0:  0.424, x1: -0.558.
Iteration:  20:   toy_utility_ei -> new value  0.253 @ x0:  0.431, x1: -0.588.
Iteration:  21:   toy_utility_ei -> new value  0.273 @ x0:  0.418, x1: -0.582.
Iteration:  22:   toy_utility_ei -> new value  0.275 @ x0:  0.413, x1: -0.577.
Iteration:  23:   toy_utility_ei -> new value  0.324 @ x0:  0.407, x1: -0.586.
Iteration:  24:   toy_utility_ei -> new value  0.325 @ x0:  0.397, x1: -0.574.
Iteration:  25:   toy_utility_ei -> new value  0.292 @ x0:  0.421, x1: -0.590.
Iteration:  26:   toy_utility_ei -> new value  0.250 @ x0:  0.415, x1: -0.572.
Iteration:  27:   toy_utility_ei -> new value  0.321 @ x0:  0.414, x1: -0.592.
Iteration:  28:   toy_utility_ei -> new value  0.231 @ x0:  0.425, x1: -0.576.
Iteration:  29:   toy_utility_ei -> new value  0.405 @ x0:  0.401, x1: -0.602.
```

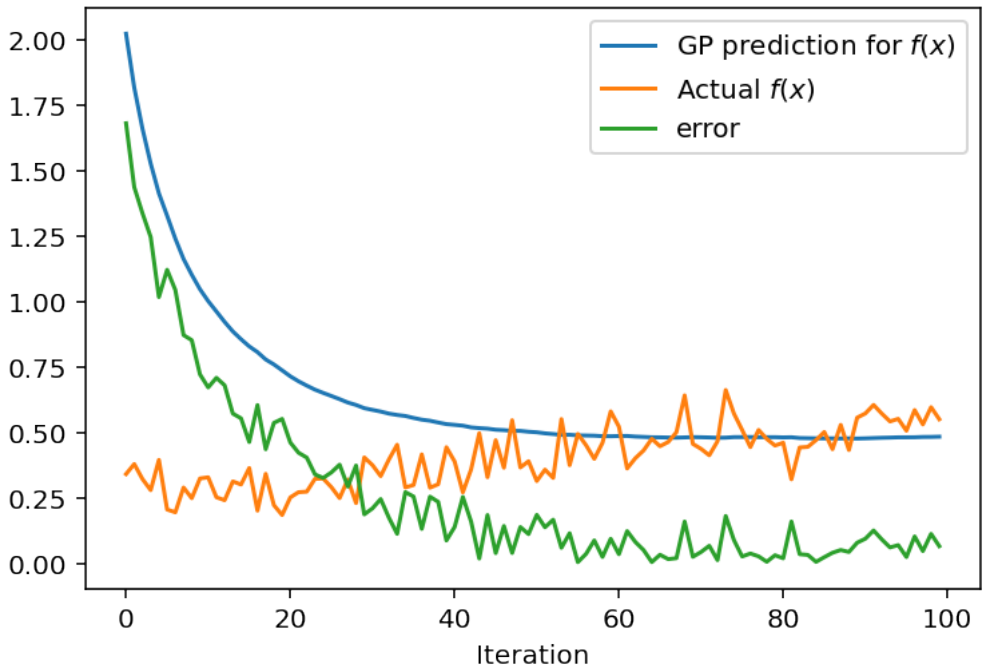
```
Iteration:  30:   toy_utility_ei -> new value  0.376 @ x0:  0.400, x1: -0.593.
Iteration:  31:   toy_utility_ei -> new value  0.334 @ x0:  0.417, x1: -0.598.
Iteration:  32:   toy_utility_ei -> new value  0.397 @ x0:  0.398, x1: -0.597.
Iteration:  33:   toy_utility_ei -> new value  0.454 @ x0:  0.387, x1: -0.599.
Iteration:  34:   toy_utility_ei -> new value  0.291 @ x0:  0.409, x1: -0.578.
Iteration:  35:   toy_utility_ei -> new value  0.300 @ x0:  0.403, x1: -0.574.
Iteration:  36:   toy_utility_ei -> new value  0.417 @ x0:  0.376, x1: -0.575.
Iteration:  37:   toy_utility_ei -> new value  0.290 @ x0:  0.402, x1: -0.569.
Iteration:  38:   toy_utility_ei -> new value  0.303 @ x0:  0.414, x1: -0.587.
Iteration:  39:   toy_utility_ei -> new value  0.444 @ x0:  0.395, x1: -0.605.
Iteration:  40:   toy_utility_ei -> new value  0.390 @ x0:  0.392, x1: -0.588.
Iteration:  41:   toy_utility_ei -> new value  0.272 @ x0:  0.415, x1: -0.579.
Iteration:  42:   toy_utility_ei -> new value  0.359 @ x0:  0.402, x1: -0.591.
Iteration:  43:   toy_utility_ei -> new value  0.498 @ x0:  0.401, x1: -0.624.
Iteration:  44:   toy_utility_ei -> new value  0.330 @ x0:  0.403, x1: -0.583.
Iteration:  45:   toy_utility_ei -> new value  0.471 @ x0:  0.394, x1: -0.611.
Iteration:  46:   toy_utility_ei -> new value  0.366 @ x0:  0.402, x1: -0.593.
Iteration:  47:   toy_utility_ei -> new value  0.548 @ x0:  0.373, x1: -0.607.
Iteration:  48:   toy_utility_ei -> new value  0.367 @ x0:  0.398, x1: -0.589.
Iteration:  49:   toy_utility_ei -> new value  0.391 @ x0:  0.392, x1: -0.589.
Iteration:  50:   toy_utility_ei -> new value  0.315 @ x0:  0.407, x1: -0.583.
Iteration:  51:   toy_utility_ei -> new value  0.359 @ x0:  0.401, x1: -0.590.
Iteration:  52:   toy_utility_ei -> new value  0.327 @ x0:  0.401, x1: -0.580.
Iteration:  53:   toy_utility_ei -> new value  0.552 @ x0:  0.374, x1: -0.609.
Iteration:  54:   toy_utility_ei -> new value  0.376 @ x0:  0.399, x1: -0.593.
Iteration:  55:   toy_utility_ei -> new value  0.495 @ x0:  0.383, x1: -0.605.
Iteration:  56:   toy_utility_ei -> new value  0.452 @ x0:  0.382, x1: -0.593.
Iteration:  57:   toy_utility_ei -> new value  0.400 @ x0:  0.382, x1: -0.577.
Iteration:  58:   toy_utility_ei -> new value  0.461 @ x0:  0.391, x1: -0.606.
Iteration:  59:   toy_utility_ei -> new value  0.581 @ x0:  0.374, x1: -0.616.
Iteration:  60:   toy_utility_ei -> new value  0.523 @ x0:  0.370, x1: -0.597.
Iteration:  61:   toy_utility_ei -> new value  0.363 @ x0:  0.400, x1: -0.590.
Iteration:  62:   toy_utility_ei -> new value  0.402 @ x0:  0.388, x1: -0.586.
Iteration:  63:   toy_utility_ei -> new value  0.432 @ x0:  0.397, x1: -0.605.
Iteration:  64:   toy_utility_ei -> new value  0.477 @ x0:  0.382, x1: -0.600.
Iteration:  65:   toy_utility_ei -> new value  0.448 @ x0:  0.379, x1: -0.589.
Iteration:  66:   toy_utility_ei -> new value  0.464 @ x0:  0.389, x1: -0.605.
Iteration:  67:   toy_utility_ei -> new value  0.502 @ x0:  0.374, x1: -0.596.
Iteration:  68:   toy_utility_ei -> new value  0.642 @ x0:  0.356, x1: -0.610.
Iteration:  69:   toy_utility_ei -> new value  0.457 @ x0:  0.374, x1: -0.584.
Iteration:  70:   toy_utility_ei -> new value  0.438 @ x0:  0.377, x1: -0.583.
Iteration:  71:   toy_utility_ei -> new value  0.414 @ x0:  0.382, x1: -0.582.
Iteration:  72:   toy_utility_ei -> new value  0.467 @ x0:  0.385, x1: -0.600.
Iteration:  73:   toy_utility_ei -> new value  0.663 @ x0:  0.355, x1: -0.613.
Iteration:  74:   toy_utility_ei -> new value  0.573 @ x0:  0.384, x1: -0.624.
Iteration:  75:   toy_utility_ei -> new value  0.510 @ x0:  0.373, x1: -0.598.
Iteration:  76:   toy_utility_ei -> new value  0.445 @ x0:  0.383, x1: -0.593.
Iteration:  77:   toy_utility_ei -> new value  0.510 @ x0:  0.379, x1: -0.605.
```

```
Iteration:  78:   toy_utility_ei -> new value  0.476 @ x0:  0.385, x1: -0.603.
Iteration:  79:   toy_utility_ei -> new value  0.451 @ x0:  0.394, x1: -0.606.
Iteration:  80:   toy_utility_ei -> new value  0.462 @ x0:  0.380, x1: -0.594.
Iteration:  81:   toy_utility_ei -> new value  0.322 @ x0:  0.400, x1: -0.577.
Iteration:  82:   toy_utility_ei -> new value  0.443 @ x0:  0.382, x1: -0.591.
Iteration:  83:   toy_utility_ei -> new value  0.446 @ x0:  0.381, x1: -0.591.
Iteration:  84:   toy_utility_ei -> new value  0.472 @ x0:  0.384, x1: -0.601.
Iteration:  85:   toy_utility_ei -> new value  0.503 @ x0:  0.377, x1: -0.600.
Iteration:  86:   toy_utility_ei -> new value  0.437 @ x0:  0.385, x1: -0.593.
Iteration:  87:   toy_utility_ei -> new value  0.530 @ x0:  0.365, x1: -0.593.
Iteration:  88:   toy_utility_ei -> new value  0.434 @ x0:  0.381, x1: -0.586.
Iteration:  89:   toy_utility_ei -> new value  0.557 @ x0:  0.370, x1: -0.606.
Iteration:  90:   toy_utility_ei -> new value  0.573 @ x0:  0.371, x1: -0.610.
Iteration:  91:   toy_utility_ei -> new value  0.606 @ x0:  0.357, x1: -0.602.
Iteration:  92:   toy_utility_ei -> new value  0.573 @ x0:  0.373, x1: -0.613.
Iteration:  93:   toy_utility_ei -> new value  0.543 @ x0:  0.375, x1: -0.608.
Iteration:  94:   toy_utility_ei -> new value  0.553 @ x0:  0.384, x1: -0.620.
Iteration:  95:   toy_utility_ei -> new value  0.507 @ x0:  0.381, x1: -0.606.
Iteration:  96:   toy_utility_ei -> new value  0.586 @ x0:  0.369, x1: -0.611.
Iteration:  97:   toy_utility_ei -> new value  0.531 @ x0:  0.360, x1: -0.586.
Iteration:  98:   toy_utility_ei -> new value  0.597 @ x0:  0.361, x1: -0.604.
Iteration:  99:   toy_utility_ei -> new value  0.551 @ x0:  0.366, x1: -0.600.
```

[12]:
```python
## 1.9: Plot the predicted values, actual function values and error.

def plot_optim_curve(optimizedX, optimizedY, optimization_log,␣
↪selection=slice(None)):
    bestIndex = np.argmin(optimizedY)
    worstIndex = np.argmax(optimizedY)
    bestPoint = optimizedX[bestIndex]
    bestValue = optimizedY[bestIndex]
    worstPoint = optimizedX[worstIndex]
    worstValue = optimizedY[worstIndex]

    print(
        f"Best value:  {bestValue[0]:6.3f} @ {bestPoint[0]:6.3f},{bestPoint[1]:
↪6.3f}\n"
        f"Worst value: {worstValue[0]:6.3f} @ {worstPoint[0]:6.
↪3f},{worstPoint[1]:6.3f}"
    )

    optimization_log_new = {}
    for key, val in optimization_log.items():
        tmp = np.array(val)
        tmp = np.reshape(tmp, -1)
        optimization_log_new[key] = tmp[selection]
```

```
    plt.plot(optimization_log_new["prediction"], label="GP prediction for␣
↪$f(x)$")
    plt.plot(optimization_log_new["result"], label="Actual $f(x)$")
    plt.plot(np.abs(optimization_log_new["prediction"] -␣
↪optimization_log_new["result"]), label="error")

    plt.xlabel("Iteration")
    plt.legend()

plot_optim_curve(toy_optimizedX, toy_optimizedY, toy_optimization_log)
```

```
Best value:   0.186 @  0.424,-0.558
Worst value: 76.500 @ -6.000,-6.000
```

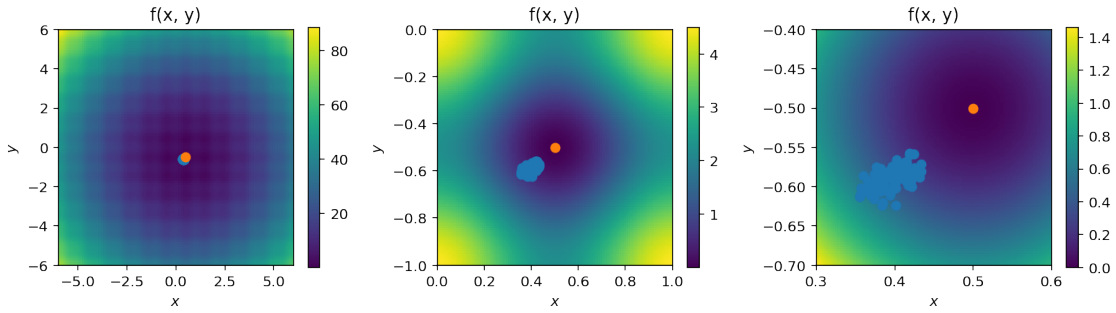


We observe that the GP nicely approximates the actual function value. However, new points proposed by minimising the utility produce points with a function value significantly higher than at the points given by the Sobol sequence.

In the next step, we look at the actually sampled points for a closer analysis:

[13]:
```
# 1.9: Plot selected points and global optimum.

fig, axes = plt.subplots(1, 3, figsize=(11, 3))
for ax, (x_plot_range, y_plot_range) in zip(axes, [
    ((-6, 6), (-6, 6)),
```

```
    ((0, 1), (-1, 0)),
    ((0.3, 0.6), (-.7, -.4))
]):
    plt.sca(ax)
    plot_ras(x_plot_range, y_plot_range)

    tmp = np.array(toy_optimization_params).reshape(-1, 2)
    plt.scatter(tmp[:, 0], tmp[:, 1])
    plt.scatter(.5, -.5)
plt.tight_layout()
```



Observation: All chosen points are clustered bottom left of the global optimum.

We are only ever sampling points that minimise the expected utility as predicted by the values we have seen so far. This encourages proposing points like the best points seen until that point, and refines the estimate of the GP around this particular region.

1.1.1 Second attempt: Alternating utility functions

[14]:
```
(
    toy2_m,
    toy2_optimizedX, toy2_optimizedY,
    toy2_optimization_log, toy2_optimization_params
) = optim(
    toy_initialX, toy_initialY, ras,
    toy_ker, [toy_utility_ei, toy_utility_var], 15, OrderedDict([
        ("x0", x_range),
        ("x1", y_range)
    ])
)
```

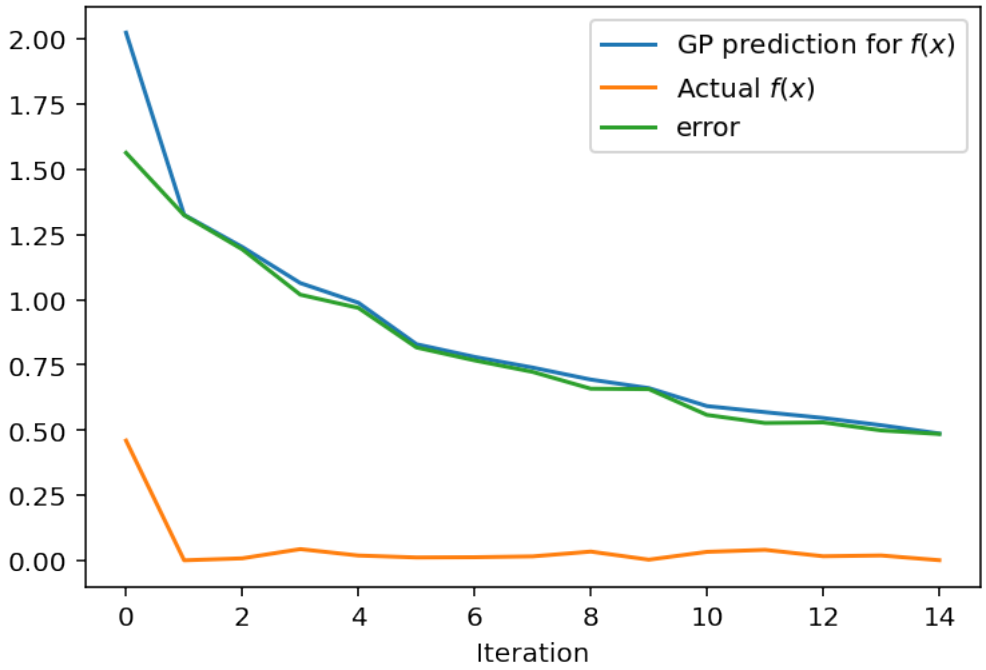
```
Iteration:   0:

 /home/felix/miniconda3/envs/optimal-transport/lib/python3.7/site-
packages/paramz/transformations.py:111: RuntimeWarning:overflow encountered in
expm1
```

```
   toy_utility_ei -> new value  0.460 @ x0:  0.377, x1: -0.589.
  toy_utility_var -> new value 75.319 @ x0:  5.937, x1:  5.969.
Iteration:   1:
   toy_utility_ei -> new value  0.001 @ x0:  0.492, x1: -0.498.
  toy_utility_var -> new value 32.438 @ x0:  0.400, x1: -5.999.
Iteration:   2:
   toy_utility_ei -> new value  0.009 @ x0:  0.485, x1: -0.515.
  toy_utility_var -> new value 87.991 @ x0: -5.974, x1:  5.988.
Iteration:   3:
   toy_utility_ei -> new value  0.044 @ x0:  0.454, x1: -0.498.
  toy_utility_var -> new value 45.214 @ x0: -5.996, x1:  0.391.
Iteration:   4:
   toy_utility_ei -> new value  0.020 @ x0:  0.478, x1: -0.478.
  toy_utility_var -> new value 63.605 @ x0:  5.971, x1: -5.953.
Iteration:   5:
   toy_utility_ei -> new value  0.012 @ x0:  0.477, x1: -0.492.
  toy_utility_var -> new value 33.293 @ x0:  6.000, x1:  0.380.
Iteration:   6:
   toy_utility_ei -> new value  0.013 @ x0:  0.476, x1: -0.492.
  toy_utility_var -> new value 48.442 @ x0: -1.200, x1:  5.999.
Iteration:   7:
   toy_utility_ei -> new value  0.017 @ x0:  0.481, x1: -0.521.
  toy_utility_var -> new value 75.763 @ x0: -5.992, x1: -5.947.
Iteration:   8:
   toy_utility_ei -> new value  0.035 @ x0:  0.468, x1: -0.474.
  toy_utility_var -> new value 37.613 @ x0: -1.685, x1: -5.999.
Iteration:   9:
   toy_utility_ei -> new value  0.004 @ x0:  0.486, x1: -0.498.
  toy_utility_var -> new value 76.179 @ x0:  5.986, x1:  5.988.
Iteration:  10:
   toy_utility_ei -> new value  0.034 @ x0:  0.468, x1: -0.476.
  toy_utility_var -> new value 37.879 @ x0:  5.993, x1: -2.717.
Iteration:  11:
   toy_utility_ei -> new value  0.042 @ x0:  0.484, x1: -0.458.
  toy_utility_var -> new value 44.387 @ x0: -5.999, x1: -0.415.
Iteration:  12:
   toy_utility_ei -> new value  0.017 @ x0:  0.473, x1: -0.489.
  toy_utility_var -> new value 87.125 @ x0: -5.989, x1:  5.916.
Iteration:  13:
   toy_utility_ei -> new value  0.020 @ x0:  0.469, x1: -0.496.
  toy_utility_var -> new value 63.545 @ x0:  5.988, x1: -5.933.
Iteration:  14:
   toy_utility_ei -> new value  0.002 @ x0:  0.491, x1: -0.502.
  toy_utility_var -> new value 48.194 @ x0:  1.931, x1:  5.999.
```

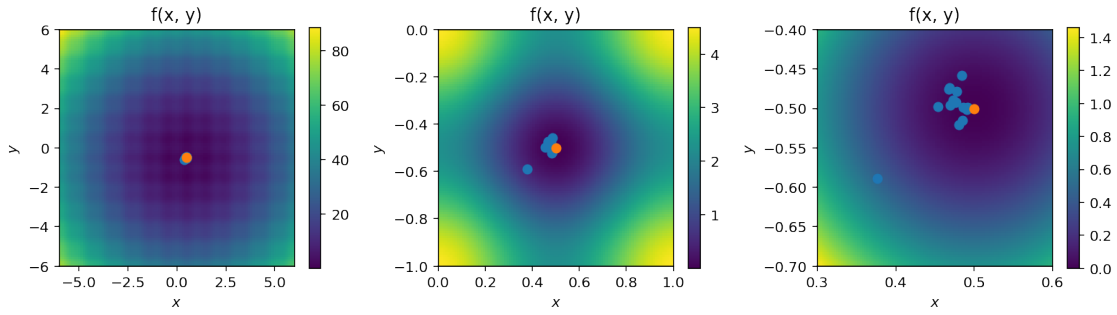
[15]:
```
# Plot the steps where expected improvement was optimized
```

```
plot_optim_curve(toy2_optimizedX, toy2_optimizedY, toy2_optimization_log,␣
 ↪slice(None, None, 2))
```

```
Best value:   0.001 @  0.492,-0.498
Worst value: 87.991 @ -5.974, 5.988
```

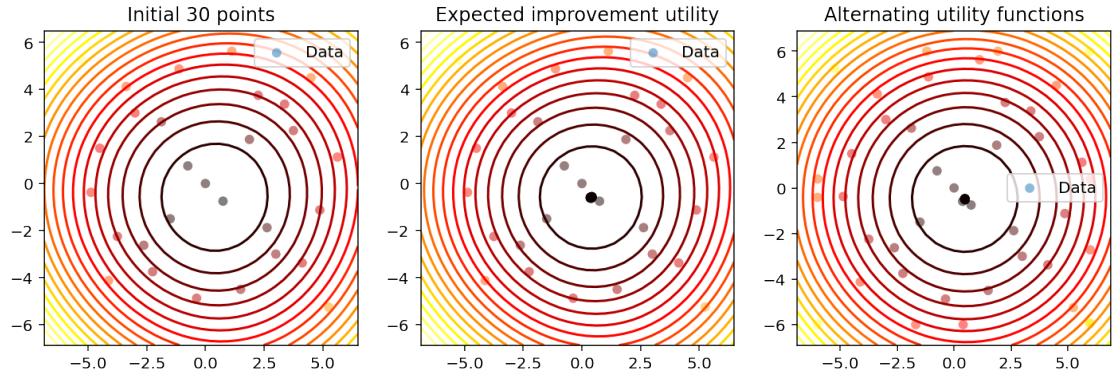


[16]:
```
# 1.9: Plot selected points and global optimum.

fig, axes = plt.subplots(1, 3, figsize=(11, 3))
for ax, (x_plot_range, y_plot_range) in zip(axes, [
    ((-6, 6), (-6, 6)),
    ((0, 1), (-1, 0)),
    ((0.3, 0.6), (-.7, -.4))
]):
    plt.sca(ax)
    plot_ras(x_plot_range, y_plot_range)

    tmp = np.array(toy2_optimization_params).reshape(-1, 2)
    plt.scatter(tmp[::2, 0], tmp[::2, 1])
    plt.scatter(.5, -.5)

plt.tight_layout()
```



[17]:
```
# 1.11: Plot the GP for the different scenarios

_, axes = plt.subplots(1, 3, figsize=(12, 4))

for ax, (title, x, y) in zip(axes, [
    ("Initial 30 points", toy_initialX, toy_initialY),
    ("Expected improvement utility", toy_optimizedX, toy_optimizedY),
    ("Alternating utility functions", toy2_optimizedX, toy2_optimizedY),
]):
    m = GPy.models.GPRegression(x, y, toy_ker)
    m.optimize(messages=False, max_f_eval=1000)
    m.plot(ax=ax)
    plt.sca(ax)
    plt.title(title)
    ax.set_aspect(1)
```

```
 /home/felix/miniconda3/envs/optimal-transport/lib/python3.7/site-
packages/paramz/transformations.py:111: RuntimeWarning:overflow encountered in
expm1
 /home/felix/miniconda3/envs/optimal-transport/lib/python3.7/site-
packages/GPy/plotting/matplot_dep/plot_definitions.py:184: UserWarning:The
following kwargs were not used by contour: 'label', 'linewidth'
```



1.12:

The Gaussian Process only captures the global structure of the function, the local structure is invisible. This is due to the kernel having a characteristic length scale of 1. (If the kernel had a smaller characteristic length scale, finer structures could be resolved. In this case, more samples would be needed.)

Thus, the function f has to be smooth enough for the GP approach to work.

Not required:

We use `Optunity` to minimize the toy function ras. We optimize 100 times using 60 evaluations per experiment and show statistics of the resulting function values (not required).

[18]:
```
values = []
num_evals = 60
for i in range(100):
    optimal_pars, info, _ = optunity.minimize(ras, num_evals=num_evals, x0=[-6,␣
 ↪6], x1=[-6, 6])
    values.append(ras(optimal_pars['x0'], optimal_pars['x1']))

values = np.array(values)
desc = scipy.stats.describe(values)

print(f"""Statistics of results found by Optunity, each using {num_evals}␣
 ↪evaluations of ras:
- Minimum: {desc.minmax[0]:6.3f}
- Maximum: {desc.minmax[1]:6.3f}
- Mean:    {desc.mean:6.3f} ± {np.sqrt(desc.variance):6.3f}""")
```

```
Statistics of results found by Optunity, each using 60 evaluations of ras:
- Minimum:  0.013
- Maximum:  2.240
- Mean:     0.934 ±  0.525
```

1.2 Exercise 2

In comparison to `ridge_regression.py`, we added γ to the game.

[19]:
```
import time

import matplotlib.pyplot as plt
import numpy as np
import scipy.sparse
import scipy.sparse.linalg
import scipy.spatial
```

```
opt_print = lambda *args, **kwargs: None  # silence output
# opt_print = print # for debugging

def gaussian_kernel(data, sigma, gamma, max_distance):
    """Compute the gaussian kernel matrix.

    :param data: data matrix
    :param sigma: parameter sigma of the gaussian kernel
    :return: gaussian kernel matrix
    """
    assert len(data.shape) == 2
    assert sigma > 0

    limit = np.exp(-.5 * np.power(2 * (max_distance / sigma) ** 2, gamma / 2))
    # Find the pairwise squared distances and compute the Gaussian kernel.
    K = []
    for k in data:
        d = np.exp(-.5 * np.power(np.sum(((data - k) / sigma) ** 2, axis=1),␣
↪gamma / 2))
        d[d < limit] = 0.0  # truncate the Gaussian
        d = scipy.sparse.csc_matrix(d[:, None])
        K.append(d)
    K = scipy.sparse.hstack(K)
    return K


def compute_alpha(train_x, train_y, tau, sigma, gamma, max_distance,
                  verbose=False):
    """Compute the alpha vector of the ridge regressor.

    :param train_x: training x data
    :param train_y: training y data
    :param tau: parameter tau of the ridge regressor
    :param sigma: parameter sigma of the gaussian kernel
    :param verbose: Print debugging information?
    :return: alpha vector
    """
    if verbose: print("Building input kernel matrix")
    K = gaussian_kernel(train_x, sigma, gamma, max_distance)
    if verbose: print("Sparsity is: %.2f%%" % (
            float(100 * K.nnz) / (K.shape[0] * K.shape[1])))
    M = K + tau * scipy.sparse.identity(train_x.shape[0])
    y = scipy.sparse.csc_matrix(train_y[:, None])
    if verbose: print("Solving sparse system")
    alpha = scipy.sparse.linalg.cg(M, train_y)
    if verbose: print("Done computing alpha")
    return alpha[0]
```

```
class KernelRidgeRegressor(object):
    """Kernel Ridge Regressor.
    """

    def __init__(self, tau, sigma, gamma):
        self.dim = None
        self.train_x = None
        self.alpha = None
        self.mean_y = None
        self.std_y = None
        self.tau = tau
        self.sigma = sigma
        self.gamma = gamma
        self.scale = -0.5 / sigma ** 2
        self.max_distance = 4.0 * sigma

    def train(self, train_x, train_y, verbose=False):
        """Train the kernel ridge regressor.

        :param train_x: training x data
        :param train_y: training y data
        """
        assert len(train_x.shape) == 2
        assert len(train_y.shape) == 1
        assert train_x.shape[0] == train_y.shape[0]

        self.dim = train_x.shape[1]
        self.train_x = train_x.astype(np.float32)
        self.tree = scipy.spatial.cKDTree(self.train_x)

        self.mean_y = train_y.mean()
        self.std_y = train_y.std()
        train_y_std = (train_y - self.mean_y) / self.std_y

        self.alpha = compute_alpha(self.train_x, train_y_std, self.tau,
                                   self.sigma, self.gamma, self.max_distance,
                                   verbose=verbose)

    def predict_single(self, pred_x):
        """Predict the value of a single instance.

        :param pred_x: x data
        :return: predicted value of pred_x
        """
        assert len(pred_x.shape) == 1
```

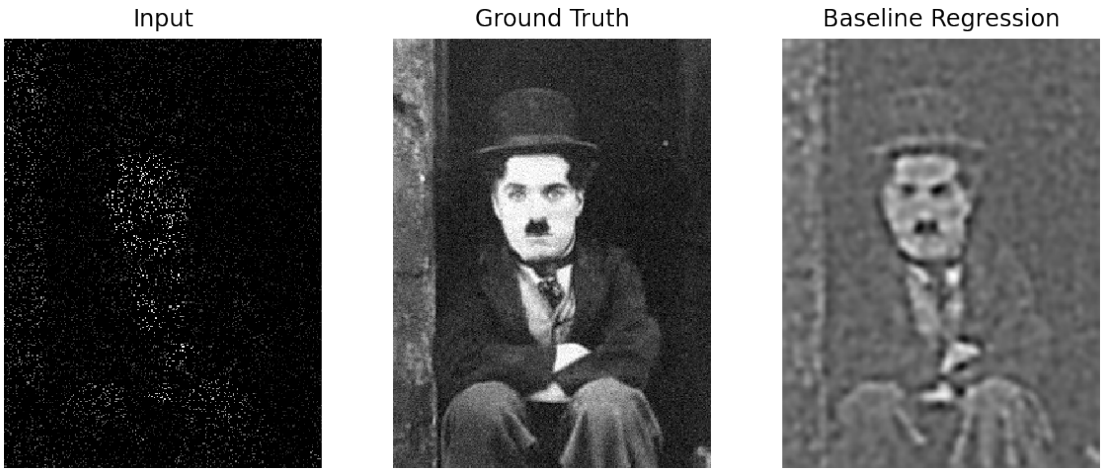
```
        assert pred_x.shape[0] == self.dim
        indices = np.asarray(self.tree.query_ball_point(pred_x, self.
↪max_distance), dtype=np.dtype("i8"))
        dist = np.sum((self.train_x[indices]-pred_x)**2, axis=1)
        kappa = np.exp(self.scale*dist)
        pred_y = np.dot(kappa, self.alpha[indices])
        return self.std_y * pred_y + self.mean_y

    def predict(self, pred_x):
        """Predict the values of pred_x.

        :param pred_x: x data
        :return: predicted values of pred_x
        """
        assert len(pred_x.shape) == 2
        assert pred_x.shape[1] == self.dim
        pred_x = pred_x.astype(np.float32)
        return np.array([self.predict_single(x) for x in pred_x])


def kernel_ridge_regression(im_orig, tau, sigma, gamma, verbose=False):
    # Make a copy, so both the original and the regressed image can be shown␣
↪afterwards.
    im = np.array(im_orig)

    # Find the known pixels and the pixels that shall be predicted.
    known_ind = np.where(im != 0)
    # predict everywhere, use im == 0 to predict only unfilled values
    unknown_ind = np.where(im >= 0)
    known_x = np.array(known_ind).transpose()
    known_y = np.array(im[known_ind])
    pred_x = np.array(unknown_ind).transpose()

    # Train and predict with the given regressor.
    start = time.time()
    if verbose: print("Training...")
    r = KernelRidgeRegressor(tau, sigma, gamma)
    r.train(known_x, known_y, verbose=verbose)
    if verbose: print("Done training.")

    if verbose: print("Predicting... ", end="")
    pred_y = r.predict(pred_x)
    if verbose: print("Done.")

    # Write the predicted values back into the image and show the result.
    im[unknown_ind] = pred_y
    stop = time.time()
```

```
    if verbose: print("Train and predict took %.02f seconds." % (stop - start))
    if verbose: print(im.shape)

    return im
```

[20]:
```
# Use default parameters as given by the sheet
im_orig = np.squeeze(plt.imread("cc_90.png"))
im_gt = plt.imread('charlie-chaplin.jpg') / 255
im_bl = kernel_ridge_regression(im_orig, 0.8, 3.0, 1.0, verbose=True)
```

```
Training…
Building input kernel matrix
Sparsity is: 1.01%
Solving sparse system
Done computing alpha
Done training.
Predicting… Done.
Train and predict took 4.68 seconds.
(338, 250)
```

[21]:
```
_, axes = plt.subplots(1, 3, figsize=(10, 4))

for ax, (img, title) in zip(axes, [
    (im_orig, "Input"),
    (im_gt, "Ground Truth"),
    (im_bl, "Baseline Regression")
]):
    plt.sca(ax)
    plt.title(title)
    plt.imshow(
        img,
        cmap='gray',
        interpolation='nearest'
    )
    plt.axis("off")
```



On the questions:

- Conjugate Gradient iteration is used to to solve the inverse of K
- `query_ball_point` is used to restrict the reconstruction to pixels close to the queried point

1.3 Exercise 3: Bayesian Optimization of Hyperparameters

[22]:
```
#3.1: Correlation function

def COR(img, gt):
    img = img.astype('float64')
    img -= np.mean(img)
    gt -= np.mean(gt)
    varGT = np.sum(np.sum(gt * gt))
    varIm = np.sum(np.sum(img * img))
    cor = np.sum(np.sum(gt * img))
    return cor / math.sqrt(varGT * varIm)


def target(tau, sigma, gamma):
    img = kernel_ridge_regression(im_orig, tau, sigma, gamma)
    return 1.0 - COR(img, im_gt)

print(f"The correlation between the ground truth and the baseline is:␣
↪{COR(im_bl, im_gt)*100:.1f}%")
```

```
The correlation between the ground truth and the baseline is: 77.3%
```

[23]:
```
# 3.2: Sobol samples

tau_range = [0.005, 1]
```

```
sigma_range = [1, 7]
gamma_range = [1, 4]

initialX, initialY = sobol_sample(30, [tau_range, sigma_range, gamma_range],␣
↪target)
```

```
Point   0.01,   1.00,   1.00 …   0.31 in     4.3s
Point   0.50,   4.00,   2.50 …   0.07 in     4.9s
Point   0.75,   2.50,   3.25 …   0.08 in     4.5s
Point   0.25,   5.50,   1.75 …   0.08 in     5.7s
Point   0.38,   3.25,   2.88 …   0.07 in     4.8s
Point   0.88,   6.25,   1.38 …   0.14 in     6.0s
Point   0.63,   1.75,   2.12 …   0.11 in     4.4s
Point   0.13,   4.75,   3.62 …   0.90 in   190.3s
Point   0.19,   2.88,   1.94 …   0.07 in     4.6s
Point   0.69,   5.88,   3.44 …   0.92 in    89.2s
Point   0.94,   1.38,   2.69 …   0.16 in     4.4s
Point   0.44,   4.38,   1.19 …   0.19 in     5.1s
Point   0.32,   2.12,   3.81 …   0.57 in     5.2s
Point   0.81,   5.12,   2.31 …   0.07 in     5.4s
Point   0.56,   3.62,   1.56 …   0.08 in     4.9s
Point   0.07,   6.62,   3.06 …   0.88 in   606.4s
Point   0.10,   3.81,   3.53 …   0.97 in    94.0s
Point   0.60,   6.81,   2.03 …   0.08 in     6.3s
Point   0.84,   2.31,   1.28 …   0.11 in     4.6s
Point   0.35,   5.31,   2.78 …   0.82 in   209.4s
Point   0.47,   1.56,   2.41 …   0.12 in     4.5s
Point   0.97,   4.56,   3.91 …   0.09 in     5.3s
Point   0.72,   3.06,   3.16 …   0.07 in     4.7s
Point   0.22,   6.06,   1.66 …   0.09 in     6.1s
Point   0.16,   1.94,   2.59 …   0.09 in     4.5s
Point   0.66,   4.94,   1.09 …   0.24 in     5.3s
Point   0.91,   3.44,   1.84 …   0.07 in     4.8s
Point   0.41,   6.44,   3.34 …   0.80 in   216.4s
Point   0.28,   2.69,   1.47 …   0.10 in     4.6s
Point   0.78,   5.69,   2.97 …   0.09 in     6.3s
```

[24]:
```
# 3.2: Utilities

def utility_ei3(tau, sigma, gamma, m):
    return expected_imp(m, np.array([[tau, sigma, gamma]]))


def utility_var3(tau, sigma, gamma, m):
    pred, var = m.predict_noiseless(np.array([[tau, sigma, gamma]]))
    return var
```

```
def utility_f3(tau, sigma, gamma):
    pred, var = m.predict_noiseless(np.array([[tau, sigma, gamma]]))
    return -pred
```

[25]:
```
# 3.2: Kernel and Optimisation

ker = GPy.kern.Matern52(3)
(
    m,
    optimizedX, optimizedY,
    optimization_log, optimization_params
) = optim(
    initialX, initialY, target,
    ker, [utility_ei3, utility_var3], 15,
    OrderedDict([
        ("tau", tau_range),
        ("sigma", sigma_range),
        ("gamma", gamma_range)
    ]),
    max_f_eval=2000
)
```

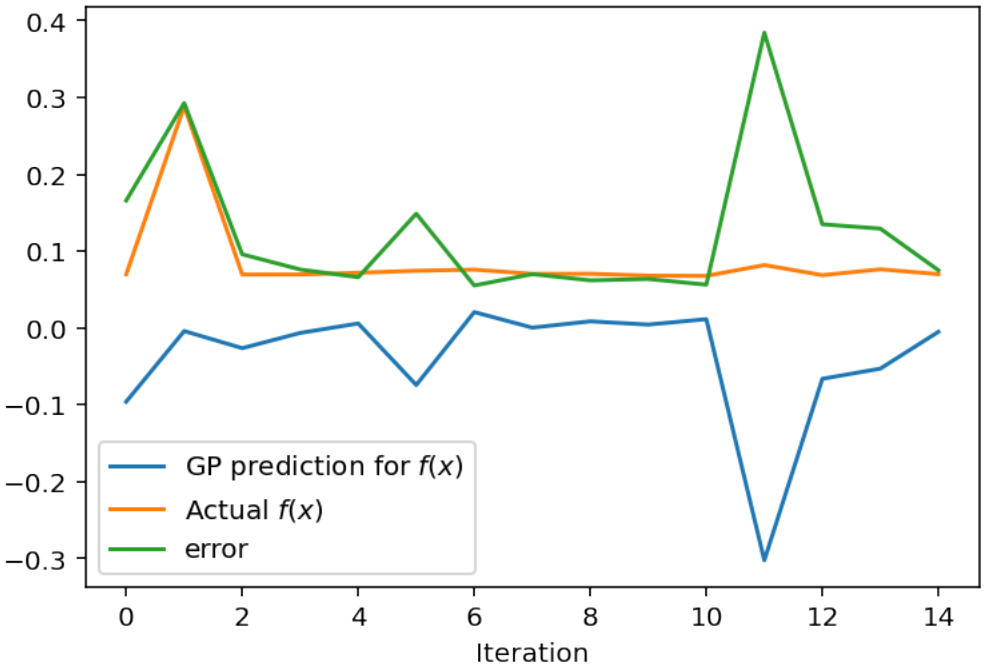
```
Iteration:   0:
      utility_ei3 -> new value  0.070 @ tau:  1.000, sigma:  3.175, gamma:
2.512.
     utility_var3 -> new value  0.251 @ tau:  0.654, sigma:  1.003, gamma:
3.991.
Iteration:   1:
      utility_ei3 -> new value  0.289 @ tau:  0.888, sigma:  6.992, gamma:
1.011.
     utility_var3 -> new value  0.675 @ tau:  0.667, sigma:  6.989, gamma:
3.995.
Iteration:   2:
      utility_ei3 -> new value  0.070 @ tau:  1.000, sigma:  3.100, gamma:
2.377.
     utility_var3 -> new value  0.075 @ tau:  0.998, sigma:  3.338, gamma:
3.995.
Iteration:   3:
      utility_ei3 -> new value  0.070 @ tau:  0.998, sigma:  3.646, gamma:
2.501.
     utility_var3 -> new value  0.237 @ tau:  0.987, sigma:  1.037, gamma:
1.261.
Iteration:   4:
      utility_ei3 -> new value  0.072 @ tau:  1.000, sigma:  2.887, gamma:
2.648.
     utility_var3 -> new value  0.907 @ tau:  0.107, sigma:  6.998, gamma:
```

```
1.016.
Iteration:   5:
      utility_ei3 -> new value  0.074 @ tau:  0.998, sigma:  5.791, gamma:
1.847.
     utility_var3 -> new value  0.758 @ tau:  0.010, sigma:  5.879, gamma:
3.991.
Iteration:   6:
      utility_ei3 -> new value  0.076 @ tau:  0.999, sigma:  5.161, gamma:
1.700.
     utility_var3 -> new value  0.908 @ tau:  0.009, sigma:  3.564, gamma:
1.016.
Iteration:   7:
      utility_ei3 -> new value  0.070 @ tau:  1.000, sigma:  2.867, gamma:
2.214.
     utility_var3 -> new value  0.258 @ tau:  0.017, sigma:  1.002, gamma:
3.582.
Iteration:   8:
      utility_ei3 -> new value  0.071 @ tau:  0.999, sigma:  2.912, gamma:
2.358.
     utility_var3 -> new value  0.960 @ tau:  0.011, sigma:  5.621, gamma:
1.001.
Iteration:   9:
      utility_ei3 -> new value  0.068 @ tau:  0.999, sigma:  3.280, gamma:
2.138.
     utility_var3 -> new value  0.876 @ tau:  0.998, sigma:  6.994, gamma:
3.991.
Iteration:  10:
      utility_ei3 -> new value  0.068 @ tau:  1.000, sigma:  3.492, gamma:
2.135.
     utility_var3 -> new value  0.961 @ tau:  0.014, sigma:  2.895, gamma:
3.995.
Iteration:  11:
      utility_ei3 -> new value  0.082 @ tau:  0.995, sigma:  5.780, gamma:
2.668.
     utility_var3 -> new value  0.083 @ tau:  0.006, sigma:  4.416, gamma:
1.944.
Iteration:  12:
      utility_ei3 -> new value  0.069 @ tau:  0.828, sigma:  4.334, gamma:
1.848.
     utility_var3 -> new value  0.239 @ tau:  0.122, sigma:  1.002, gamma:
2.171.
Iteration:  13:
      utility_ei3 -> new value  0.076 @ tau:  0.999, sigma:  6.347, gamma:
2.020.
     utility_var3 -> new value  0.269 @ tau:  0.132, sigma:  1.997, gamma:
1.021.
Iteration:  14:
      utility_ei3 -> new value  0.070 @ tau:  0.432, sigma:  2.626, gamma:
```

```
2.276.
     utility_var3 -> new value  0.104 @ tau:  0.994, sigma:  1.912, gamma:
3.996.
```

[26]:
```
# 3.3:
plot_optim_curve(optimizedX, optimizedY, optimization_log, slice(None, None, 2))
```

```
Best value:   0.068 @  1.000, 3.492
Worst value:  0.966 @  0.098, 3.812
```



[27]:
```
# 3.4: Reconstruct images
start = time.time()
im_best = kernel_ridge_regression(im_orig, *optimizedX[np.argmin(optimizedY)])
end = time.time()
print(f"Best image time:  {end - start:4.1f}s")
start = time.time()
im_worst = kernel_ridge_regression(im_orig, *optimizedX[np.argmax(optimizedY)])
end = time.time()
print(f"Worst image time: {end - start:4.1f}s")
```

```
Best image time:   4.9s
Worst image time: 99.8s
```

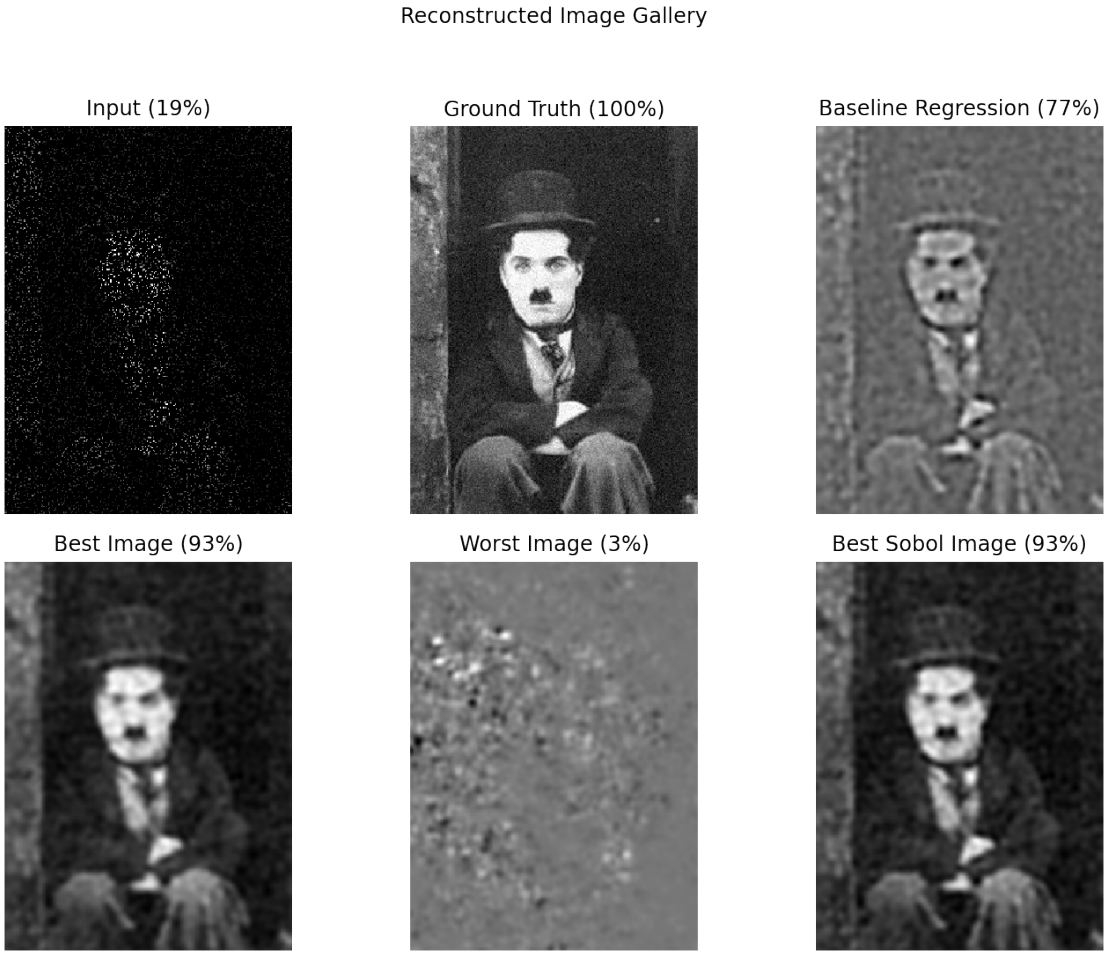
Not required:

Sample some more points from the Sobolev sequence as another baseline.

[28]:
```
moreX, moreY = sobol_sample(1, [tau_range, sigma_range, gamma_range], target,␣
↪offset=initialX.shape[0])

allX = np.concatenate([initialX, moreX])
allY = np.concatenate([initialY, moreY])

best_sobol = allX[np.argmin(allY)]
im_sobol = kernel_ridge_regression(im_orig, *best_sobol)
```

```
Point   0.53,   1.19,   3.72 …   0.20 in     4.5s
```

[29]:
```
# 3.4: Compare the different reconstructions

_, axes = plt.subplots(2, 3, figsize=(10, 8))

plt.suptitle("Reconstructed Image Gallery")

for ax, (img, title) in zip(axes.reshape(-1), [
    (im_orig,  "Input"),
    (im_gt,    "Ground Truth"),
    (im_bl,    "Baseline Regression"),
    (im_best,  "Best Image"),
    (im_worst, "Worst Image"),
    (im_sobol, "Best Sobol Image")
]):
    plt.sca(ax)
    plt.title(f"{title} ({COR(img, im_gt) * 100:.0f}%)")
    plt.imshow(
        img,
        cmap='gray',
        interpolation='nearest'
    )
    plt.axis("off")

plt.tight_layout(rect=[0, 0.03, 1, 0.95])
```


Reconstructed Image Gallery
Input (19%)
Ground Truth (100%)
Baseline Regression (77%)
Best Image (93%)
Worst Image (3%)
Best Sobol Image (93%)

[ ]: