AI505/AI801, Optimization – Exercise Sheet 06

2026-03-15

Exercises with the symbol $^+$ are to be done at home before the class. Exercises with the symbol * will be tackled in class. The remaining exercises are left for self training after the exercise class. Some exercises are from the text book and the number is reported. They have the solution at the end of the book.

Exercise 1 *

Using real-valued encoding for the solution representation, implement in Python a plain version of one of the following metaheuristic optimization algorithms:

- Genetic Algorithms (GA),
- Differential Evolution (DE),
- Particle Swarm Optimization (PSO).

Then, modify your implementation to produce a hybrid version of the algorithm by adding a local search method. You can use any method for the exploitation of design points studied in the previous parts of this course and Lamarkian or Baldwinian learning.

Test and compare your implementations of the plain and the hybrid versions on a few benchmark functions from the module landscapes.

Exercise 2 $^+$ (13.1)

Filling a multidimensional space requires exponentially more points as the number of dimensions increases. To help build this intuition, determine the side lengths of an n-dimensional hypercube such that it fills half of the volume of the n-dimensional unit hypercube.

Exercise 3 $^+$

Generate a full factorial set of design points in Python using `numpy.meshgrid` for two dimensions and plot the generated points. Find a function from the benchmark suite and evaluate the function in those points.

Repeat the same process for a function with $n = 3$ and for $n > 3$ dimensions.

Exercise 4 *

Construct in Python a Latin Hypercube design plan with numbers 1 through $m = 4$ for 3 dimensions. How many different Latin Hypercubes are there? How many different uniform projection plans are there for a single Latin Hypercube? How many different uniform projection plans can be generated from all Latin Hypercubes of m numbers in 3 dimensions?

Exercise 5

Consult the documentation of the quasi-Monte Carlo python submodule `scipy.stats.qmc` and repeat the previous exercises using those functions.

Exercise 6 *

Quasi-random sequences are typically constructed for the unit n-dimensional hypercube, $[0, 1]^n$. Any multidimensional function with bounds on each variable can be transformed into such a hypercube. Show how.

Exercise 7 * (13.3)

Suppose we have a sampling plan $X = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_{10}\}$, where

$$x_i = [\cos(2\pi i/10), \sin(2\pi i/10)]$$

Compute the Morris-Mitchell criterion for X using an L_2 norm when the parameter q is set to 2. In other words, evaluate $\Phi_2(X)$. If we add $[2, 3]$ to each $\boldsymbol{x}_i$, will $\Phi_2(X)$ change? Why or why not?

Compare the MM criterion for the points given above with a set of 10 uniformly sampled random points.

Exercise 8 (13.4)

Additive recurrence requires that the multiplicative factor c be irrational. Why can c not be rational?

Exercise 9 *

Consider the set X of random points given by:

```
np.random.seed(42)
X = np.random.rand(6, 2)
```

Plot them and:

- calculate the discrepancy of the sample plan.
- find the best space-filling subset of size 3.

Exercise 10 *

Consider the sets X and Y of random points given by:

```
np.random.seed(42)
X = np.random.rand(10, 2)
Y = np.random.rand(10, 2)
```

Plot them and decide which is the most space-filling on the basis of pairwise distances.

Exercise 11

Calculate an approximation of the value of π using the Monte Carlo method. Draw a circle inscribed in a unit square and use a sampling plan to create sample points in the square. Use the ratio between the number of points inside the circle and the total number of points to approximate π.

Compare the convergence rate of uniform random sampling against a quasi-Monte Carlo low discrepancy method like Sobol method.

Plot the sample plans generated.