AI505 – Optimization

Sheet 08, Spring 2025

Solution:

Included.

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$^+$ (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.

Solution:

The volume of an hypercube I^n is:

$$V = r^n$$

to find the hypercube of side s that fills half of V we need to compute:

$$r^n/2 = s^n \implies s = r\frac{1}{\sqrt[n]{2}} = r2^{-1/n}$$

As an example for $n = 3$ with $r = 1$, s must be 0.79.

Exercise 2$^+$ (13.2)

Suppose that you sample randomly inside a unit sphere in n dimensions. Compute the probability that a randomly sampled point is within ϵ distance from the surface of the sphere as $n \to \infty$. Hint: The volume of a sphere is $C(n)r^n$, where r is the radius and $C(n)$ is a function of the dimension n only.

Solution:

The probability that a randomly sampled point is within ϵ-distance from the surface is the ratio of the volumes. Thus:

$$P(\|\boldsymbol{x}\|_2 > 1 - \epsilon) = 1 - P(\|\boldsymbol{x}\|_2 \leq 1 - \epsilon) = 1 - (1 - \epsilon)^n$$

that tends to 1 as $n \to \infty$.

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.

Solution:

```
import numpy as np
import itertools

def samples_full_factorial_prod(a, b, m):
    ranges = [np.linspace(a[i], b[i], m[i]) for i in range(len(a))]
    F = itertools.productranges)return F
```

```
def samples_full_factorial_mesh(a, b, m):
    ranges = [np.linspace(a[i], b[i], m[i]) for i in range(len(a))]
    M = np.meshgridranges)return M
```

```
def uniform_projection_plan(m, n):
    perms = [np.random.permutation(m) for i in range(n)]
    L = np.array([[perms[i][j] for i in range(n)] for j in range(m)])
    return L
```

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?

Solution:

There are $(4!)^3$ Latin Hypercubes of dimension 3. For each Latin Hypercube there are

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.

Solution:

Let $x \in [a_1, b_1] \times [a_2, b_2] \times \ldots \times [a_n, b_n]$. We want to tranform to and from variables $x' \in [0, 1]^n$, ie, belonging to the hypercube $I^n = [0, 1]^n$.
For every dimension $i = 1, \ldots, n$:

$$x'_i = x_i \cdot (b_i - a_i) + a_i \qquad x_i = \frac{x'_i - a_i}{b_i - a_i}$$

This is also called *normalization* (which is not the same as *standardization*.
In Python terms, we can transform a sampling plan generated in the hypercube I^n to a sampling plan for the hyperrectangle $[a_1, b_1] \times [a_2, b_2] \times \ldots \times [a_n, b_n]$ as follows:

```
import numpy as np

def rev\_unit\_hypercube\_parameterization(x, a, b):
    return x*(b-a)+a
```

Alternatively, we can stay in the hypercube and transform the function to a function in the hypercube as follows:

```
def reparameterize\_to\_unit\_hypercube(f, a, b):
  delta = b - a
  return lambda x: f(x * delta + a)
```

Exercise 7* (13.3)

Suppose we have a sampling plan $X = \{x_1, ..., x_{10}\}$, 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 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.

Solution:

```python
import numpy as np

def pairwise_distances(X: np.ndarray, p: int = 2) -> np.ndarray:
    m = X.shape[0]
    dists = []
    for i in range(m - 1):
        for j in range(i + 1, m):
            d = np.linalg.norm(X[i] - X[j], ord=p)
            dists.append(d)
    return np.array(dists)

def phiq(X: np.ndarray, q: float = 1.0, p: int = 2) -> float:
    dists = pairwise_distances(X, p)
    return np.sum(dists ** (-q)) ** (1.0 / q)

# Create 10 points evenly spaced on the unit circle
X = np.array([
    [np.cos(2 * np.pi * i / 10), np.sin(2 * np.pi * i / 10)]
    for i in range(10)
])

print("phiq(X, 2):", phiq(X, q=2))


np.random.seed(42)
random_points = np.random.rand(10, 2)

print("phiq(X, 2):", phiq(random_points, q=2))
```

phiq(X, 2): 6.422616289332565
phiq(X, 2): 31.659636916479467

No. The Morris-Mitchell criterion is based entirely on pairwise distances. Shift- ing all of the points by the same amount does not change the pairwise distances and thus will not change $\Phi_2(X)$.

Exercise 8 (13.4)

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

Solution:
A rational number can be written as a fraction of two integers a/b. It follows that the sequence repeats every b iterations:

$$x_{k+1} = x_k + \frac{a}{b} \pmod 1$$

$$x_k = x_0 + k\frac{a}{b} \pmod 1$$

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.

Solution:
The unit square is $[0,1] \times [0,1]$. The equation of a circle of radius 1 inscribed in the square is

$$x^2 + y^2 = 1$$

Points internal to the circle satisfy $x^2 + y^2 \leq 1$. We denote the area under the curve by $\mathcal{H}$.
The area of the square is 1 and the one of the circle is $\pi r^2 = \pi$. The area inscribed in the square is $\pi/4$.

$$\frac{\#\mathcal{H}}{\#X} = r = \frac{\pi}{4}$$

Hence,

$$\pi = 4r$$

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import qmc

# Settings
max_power = 10 # up to 2^10 = 1024 points

# --- Sobol sequence ---
sobol_sampler = qmc.Sobol(d=2, scramble=True)
sobol_points = sobol_sampler.random_base2(m=max_power)
#sobol_points = 2 * sobol_points - 1
sobol_radii = np.sum(sobol_points**2, axis=1)

# --- Monte Carlo (uniform random) ---
np.random.seed(42)
random_points = np.random.rand(2**max_power, 2)
random_radii = np.sum(random_points**2, axis=1)

# Store estimates
```

```python
ns = []
pi_sobol = []
pi_random = []

# Compute estimates for each power of 2
for m in range(1, max_power + 1):
    n = 2**m
    ns.append(n)

    inside_sobol = np.sum(sobol_radii[:n] <= 1)
    inside_random = np.sum(random_radii[:n] <= 1)

    pi_sobol.append(4 * inside_sobol / n)
    pi_random.append(4 * inside_random / n)

# --- Plot ---
plt.figure(figsize=(10, 5))
plt.plot(ns, pi_sobol, marker='o', label='Sobol Estimate')
plt.plot(ns, pi_random, marker='x', linestyle='--', label='Monte Carlo Estimate', alpha
    =0.7)
plt.axhline(np.pi, color='red', linestyle=':', label=r'True $\pi$')
plt.xscale('log', base=2)
plt.xlabel("Number of Points (log scale)")
plt.ylabel(r"Estimated $\pi$")
plt.title(r"Comparison: Sobol vs Monte Carlo Estimation of $\pi$")
plt.grid(True, linestyle='--', linewidth=0.5)
plt.legend()
plt.tight_layout()
plt.savefig("pi_convergence.png")


# Plotting
fig, axs = plt.subplots(1, 2, figsize=(12, 6), sharex=True, sharey=True)

# Circle coordinates
theta = np.linspace(0, np.pi/2, 500)
circle_x = np.cos(theta)
circle_y = np.sin(theta)

# Sobol sequence
# axs[0].scatter(sobol_points[:, 0], sobol_points[:, 1], s=5, color='blue')
axs[0].scatter(sobol_points[sobol_radii > 1, 0], sobol_points[sobol_radii > 1, 1], s=5,
    color='gray', label='Outside Circle')
axs[0].scatter(sobol_points[sobol_radii <= 1, 0], sobol_points[sobol_radii <= 1, 1], s=5,
    color='blue', label='Inside Circle')
axs[0].plot(circle_x, circle_y, color='red', linewidth=1.5, label='Unit Circle')
axs[0].set_title("Sobol Sequence")
axs[0].set_xlabel("x")
axs[0].set_ylabel("y")
axs[0].axis("square")
axs[0].grid(True, linestyle='--', linewidth=0.5)

# Random sampling
#axs[1].scatter(random_points[:, 0], random_points[:, 1], s=5, color='orange')
axs[1].scatter(random_points[random_radii > 1, 0], random_points[random_radii > 1, 1], s
    =5, color='gray', label='Outside Circle')
axs[1].scatter(random_points[random_radii <= 1, 0], random_points[random_radii <= 1, 1], s
    =5, color='blue', label='Inside Circle')
axs[1].plot(circle_x, circle_y, color='red', linewidth=1.5, label='Unit Circle')
axs[1].set_title("Monte Carlo (Uniform Random)")
axs[1].set_xlabel("x")
axs[1].axis("square")
axs[1].grid(True, linestyle='--', linewidth=0.5)
```
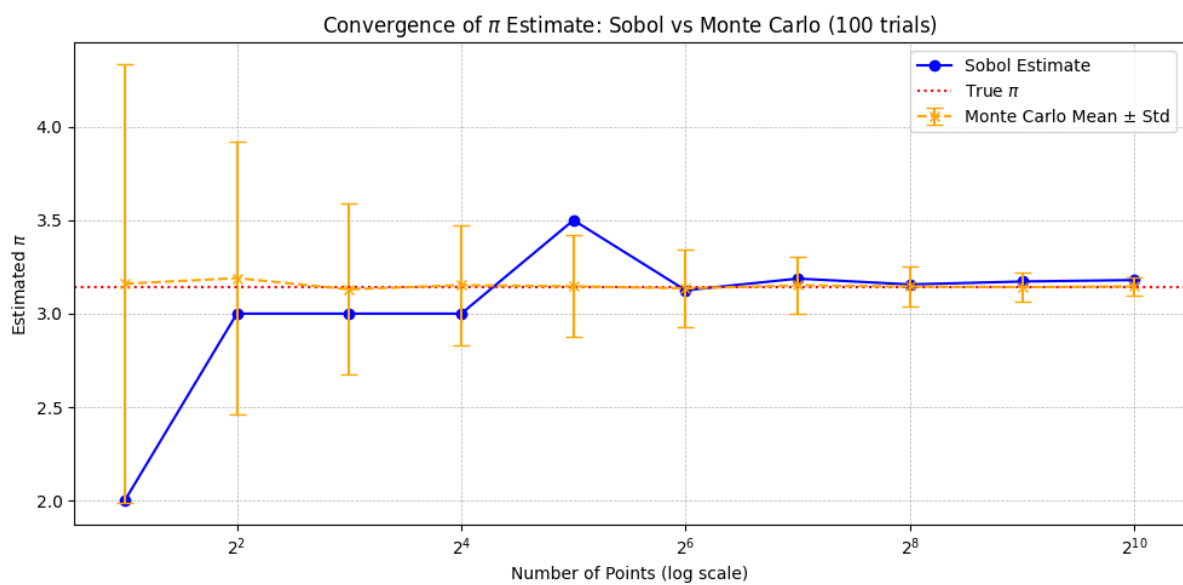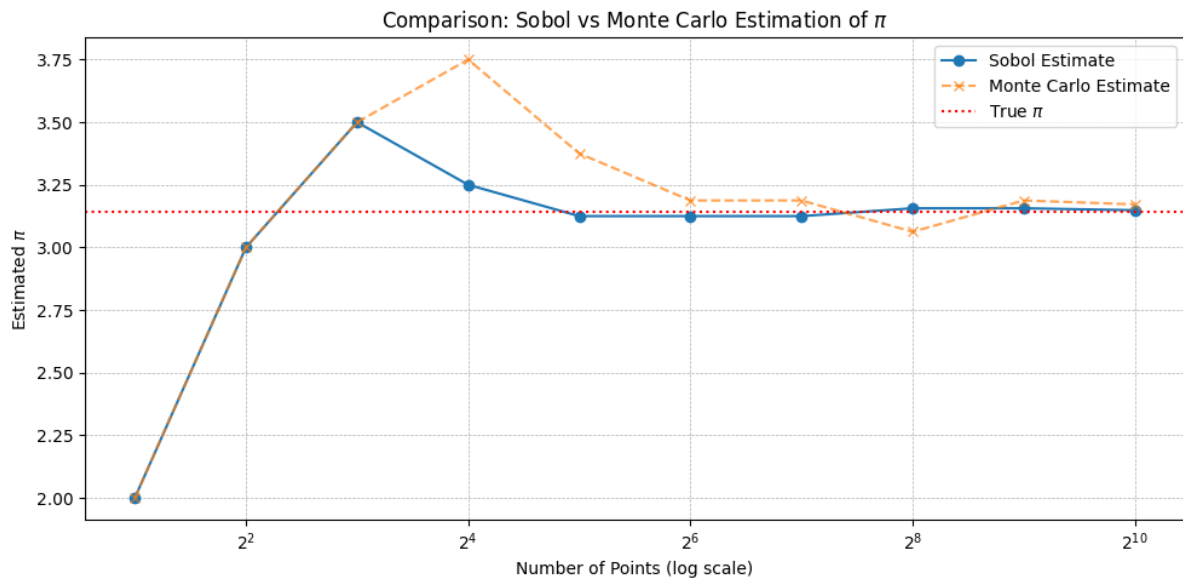
5

Comparison: Sobol vs Monte Carlo Estimation of π

Convergence of π Estimate: Sobol vs Monte Carlo (100 trials)

```
plt.suptitle(f"Distribution of {n} Points: Sobol vs Monte Carlo", fontsize=14)
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.savefig("pi_points.png")
```

Distribution of 1024 Points: Sobol vs Monte Carlo