PPPD - Lab. 07

Zadanie punktowane, lab 07, 2017/2018

Temat: Klątwa wymiarowości

Treść zadania

W trakcie zajęć poznaliśmy sposób obliczania pola powierzchni figury przy użyciu metody typu Monte Carlo (MMC). Polegała ona na losowaniu punktów z prostokątnego obszaru i zliczaniu, ile z nich spełnia podane warunki. Do tej pory metodę tę wykorzystywaliśmy do obliczania pola powierzchni figury będącej zbiorem punktów w $\mathbb{R}^2$, jednak – jak wiemy – nie żyjemy w dwuwymiarowym świecie, a zjawiska, które chcemy badać, wymagają często umiejętności analizowania wielowymiarowych zależności.

Dzisiejsze zadanie będzie polegało na zaobserwowaniu, jak zachowują się odległości między punktami w n-wymiarowej przestrzeni $\mathbb{R}^n$ i dlaczego klątwa wymiarowości (ang. *curse of dimensionality*) zasłużyła sobie na swoją nazwę.

Funkcje `distance` i `random_point` [2 p.]

Napisz funkcję `distance(x, y)` obliczającą odległość euklidesową między dwoma punktami $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ (reprezentowanymi przy użyciu dwóch list liczbowych), gdzie $n \geqslant 1$, tj.:

$$\texttt{distance}(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}.$$

Następnie napisz funkcję `random_point(n, a=-1, b=1)` zwracającą n-elementową listę taką, że jej i-ty element jest pseudolosową wartością wygenerowaną z rozkładu jednostajnego na przedziale $[a, b]$ dla każdego możliwego i. Do generowania każdej wartości wykorzystaj funkcję `random.uniform()`.

Funkcje `volume_exact` i `unit_ball_ratio` [1 p.]

n-wymiarową kulą o środku $\mathbf{x} \in \mathbb{R}^n$ i promieniu r nazywamy zbiór:

$$B_n(\mathbf{x}, r) = \{\mathbf{y} \in \mathbb{R}^n : \texttt{distance}(\mathbf{x}, \mathbf{y}) < r\}.$$

Napisz funkcję `volume_exact(n)`, która wyznaczy dokładną wartość objętości n-wymiarowej kuli o środku $(0, \ldots, 0)$ i promieniu $r = 1$, tj. kuli jednostkowej.

Dokładną wartość można wyznaczyć, korzystając ze wzoru:

$$V_n(r) = \frac{\pi^{\frac{n}{2}}}{\Gamma\left(\frac{n}{2} + 1\right)} r^n,$$

gdzie:

- $\Gamma(x)$ jest funkcją gamma, zob. `math.gamma()`,
- π jest równe `math.pi`.

Okazuje się, że objętość n-wymiarowej jednostkowej kuli $B_n((0,\ldots,0),1)$ bardzo szybko maleje, a co za tym idzie metody typu Monte Carlo w wyższych wymiarach mogą stać się nieefektywne: będą potrzebowały dużej liczby punktów, by w ogóle trafić w badany obszar.

Aby zbadać to zjawisko, napisz wpierw funkcję `unit_ball_ratio(n)`, która dla danego n zwróci stosunek dokładnej objętości kuli $B_n((0,\ldots,0),1)$ do objętości kostki $[-1,1]^n$.

Funkcja `volume_approx` [1 p.]

Następnie napisz funkcję `volume_approx(n, m=10000)` obliczającą przybliżoną wartość objętości n-wymiarowej kuli jednostkowej $B_n((0,\ldots,0),1)$.

Przybliżoną objętość możemy obliczyć przy użyciu metody typu Monte Carlo:

1. Wylosuj `m` punktów $\mathbf{x} \in \mathbb{R}^n$ z rozkładu jednostajnego na kostce $[-1,1]^n$.
2. Wyznacz frakcję (proporcję) punktów oddalonych od punktu $(0,\ldots,0)$ o nie więcej niż 1 (odległość euklidesowa).
3. Uzyskany w punkcie 2. wynik odpowiednio przeskaluj, mnożąc go przez objętość kostki $[-1,1]^n$.

Rysunki ilustrujące zachowanie sie odległości w przestrzeniach o dużym wymiarze [1 p.]

Narysuj wykresy ilustrujące, jak zmienia się objętość kuli $B_n((0,\ldots,0),1)$ wraz z rosnącym n oraz jak zmienia się stosunek objętości kuli $B_n((0,\ldots,0),1)$ do objętości kostki $[-1,1]^n$.

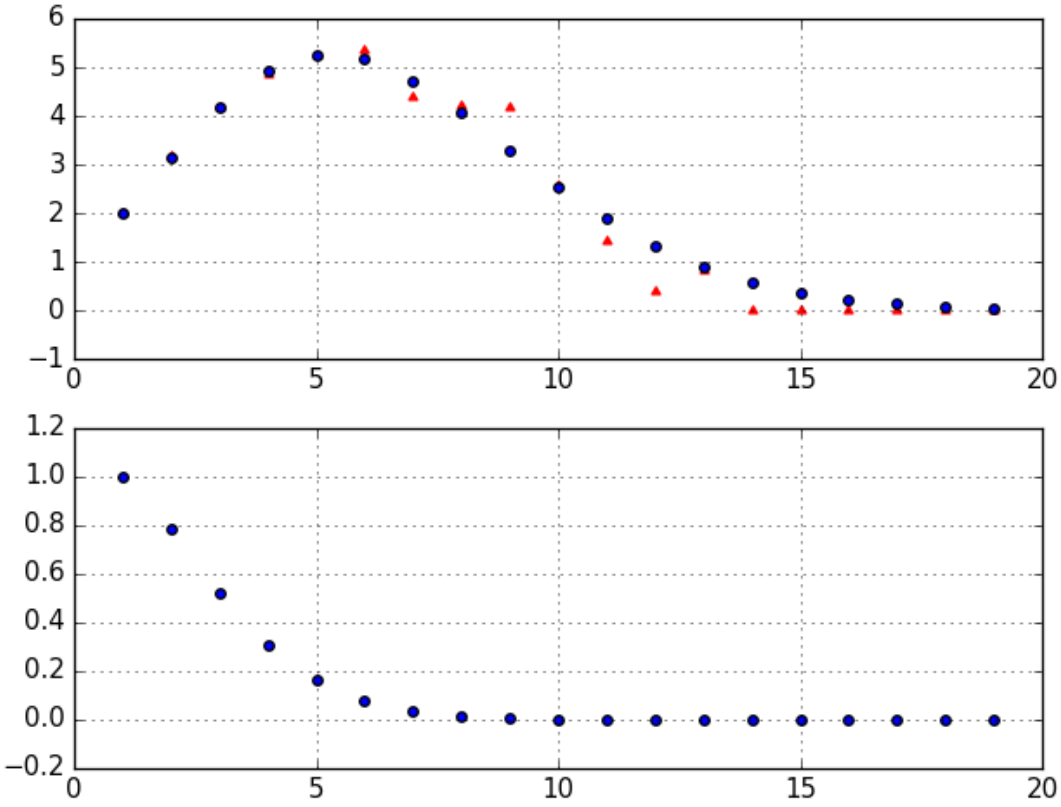
Wykorzystaj napisane przed chwilą funkcje `volume_exact`, `volume_approx` i `unit_ball_ratio` do utworzenia wektorow `y_exact`, `y_approx`, `y_ratio`, w taki sposób, że dla $i = 0, ..., 18$:

- `y_exact[i] = volume_exact(i+1)`
- `y_approx[i] = volume_approx(i+1)`
- `y_ratio[i] = unit_ball_ratio(i+1)`

Następnie skorzystaj z poniższego kodu, aby narysować ciekawe wykresy.

```
import matplotlib.pyplot as plt

fig = plt.figure()

x = range(1, 20)
y_exact = ... # volume_exact
y_approx = ... # volume_approx
y_ratio = ... # unit_ball_ratio


ax1 = fig.add_subplot(2, 1, 1)
ax1.scatter(x, y_approx, color="red", marker=(3,0,0))
ax1.scatter(x, y_exact)
ax1.grid()

ax2 = fig.add_subplot(2, 1, 2)
ax2.scatter(x, y_ratio)
ax2.grid()
```

```
fig.savefig('volume.png', dpi=90)
```



Rysunek 1: Objętość kuli

Funkcja `czas_do_sukcesu` oraz zapis wyników do pliku [1 p.]

Napisz funkcję `czas_do_sukcesu(n)`, która zwraca liczbę punktów, które trzeba wylosować, aby trafić po raz pierwszy w kulę. Oczywiście przy każdym jej wywołaniu możemy otrzymywać inną wartość.

Utwórz plik tekstowy `czas.txt` i zapisz do niego:

- oczekiwane (średnie) czasy (rozumiane jako liczba punktów, które trzeba wylosować) do pojawienia się pierwszego wylosowanego punktu, który będzie należał do kuli. Zauważ, że czas ten będzie wynosił $\frac{1}{p}$, gdzie p – prawdopodobieństwo wylosowania punktu należącego do kuli $B_n((0,\ldots,0),1)$, które znajdziemy przy użyciu funkcji `unit_ball_ratio(n)`. Na przykład dla $n = 2$ średni czas oczekiwania wyniesie około 2, natomiast dla $n = 10$ będzie to już ponad 400.
- czas dla przykładowego losowania (wywołanie funkcji `czas_do_sukcesu(n)`)

Ładnie sformatowany wynik dla wymiarów $n \in \{1,\ldots,15\}$ powinien wyglądać na przykład tak:

```
n |  oczekiwany | przykładowy
1 |        1.00 |           1
2 |        1.27 |           1
```

```
 3 |        1.91 |           4
 4 |        3.24 |           5
 5 |        6.08 |           9
 6 |       12.38 |          19
 7 |       27.09 |           2
 8 |       63.07 |          28
 9 |      155.22 |          59
10 |      401.54 |          95
11 |     1086.99 |        2668
12 |     3067.56 |        9742
13 |     8995.98 |        2465
14 |    27340.18 |       83407
15 |    85905.30 |       95391
```

Jakość kodu [1 p.]

Aby otrzymać pozostały, siódmy punkt za zadanie:

- przesłany skrypt musi wykonać się bez błędów,
- kod musi być napisany w sposób czytelny,
- kod musi być dobrze i dokładnie udokumentowany.