Programowanie i metody numeryczne

Ćwiczenia 11.

Metody Monte Carlo.

Zadanie 1. `pimc` – Liczba π.

Napisz szablon funkcji

```
template<typename RNG>
double IntMonteCarlo(const auto &f, double a, double b, std::size_t N)
```

która posługując się metodą Monte Carlo znajduje przybliżoną wartość całki oznaczonej

$$I = \int_a^b f(x)\,\mathrm{d}x$$

całkowalnej funkcji $f : \mathbb{R} \supseteq X \to \mathbb{R}$ w przedziale $[a, b] \subset X$, wykorzystując N losowo wybranych punktów. Funkcja `IntMonteCarlo` przyjmuje następujące argumenty: `f` – implementację funkcji f; `a` i `b` – odpowiadające kolejno liczbom a i b; `N` – odpowiadający liczbie N. Parametr `RNG` szablonu ma być typem reprezentującym generator liczb losowych, który funkcja powinna wykorzystać do wylosowania punktów. Wartością zwracaną przez funkcję `IntMonteCarlo` powinna być obliczona przez nią przybliżona wartość całki oznaczonej I.

Korzystając z tego szablonu napisz program `pimc`, który oblicza metodą Monte Carlo przybliżoną wartość liczby π na podstawie zależności

$$\frac{\pi}{2} = \int_{-1}^{1} \sqrt{1 - x^2}\,\mathrm{d}x.$$

Program ma przyjmować jako argument wywołania dodatnią liczbę całkowitą określającą wartość N. Dla zadanej w ten sposób wartości N program powinien wyznaczać wartość liczby π stukrotnie, a następnie wypisywać na standardowe wyjście średnią wyznaczonych wartości oraz ich wariancję. Jako generator liczb losowych wykorzystaj *Mersenne Twister 19937* dostarczany przez plik nagłówkowy `random`.

Zadanie 2. `lemniscate` – Pole powierzchni ograniczonej lemniskatą Bernoulliego.

Napisz szablon funkcji

```
template<typename RNG>
double LemniscateArea(std::size_t N)
```

która posługując się metodą Monte Carlo znajduje przybliżoną wartość pola powierzchni ograniczonej lemniskatą Bernoulliego, opisaną równaniem

$$\left(x^2 + y^2\right)^2 \leqslant 2\left(x^2 - y^2\right),$$

wykorzystując N losowo wybranych punktów. Jako obszar całkowania przyjmij $\Omega = [-1,5,\ 1,5] \times [-0,6,\ 0,6]$. Funkcja `LemniscateArea` przyjmuje argument `N` odpowiadający liczbie N. Parametr `RNG` szablonu ma być

typem reprezentującym generator liczb losowych, który funkcja powinna wykorzystać do wylosowania punktów. Wartością zwracaną przez funkcję `LemniscateArea` powinna być obliczona przez nią przybliżona wartość szukanego pola powierzchni.

Korzystając z tego szablonu napisz program `lemniscate`, który przyjmuje jako argument wywołania dodatnią liczbę całkowitą określającą wartość N. Program powinien wypisywać na standardowe wyjście obliczoną przez funkcję `:emniscateArea` wartość pola powierzchni ograniczonej lemniskatą Bernoulliego. Jako generator liczb losowych wykorzystaj *Mersenne Twister 19937* dostarczany przez plik nagłówkowy `random`.

Zadanie 3. `ball` – Objętość n–wymiarowej kuli jednostkowej.

Napisz szablon funkcji

```
template<typename RNG>
double BallVolume(std::size_t n, std::size_t N)
```

która posługując się metodą Monte Carlo znajduje przybliżoną objętość kuli jednostkowej w n–wymiarowej przestrzeni euklidesowej, wykorzystując N losowo wybranych punktów. Funkcja `BallVolume` przyjmuje argumenty `n` i `N` odpowiadające kolejno liczbom n i N. Parametr `RNG` szablonu ma być typem reprezentującym generator liczb losowych, który funkcja powinna wykorzystać do wylosowania punktów. Wartością zwracaną przez funkcję `BallVolume` powinna być obliczona przez nią przybliżona objętość n–wymiarowej kuli jednostkowej.

Korzystając z tego szablonu napisz program `ball`, który przyjmuje jako argumenty wywołania dwie dodatnie liczby całkowite k i N. Program powinien obliczać objętości n–wymiarowych kul jednostkowych dla $n = 1, 2, \ldots, k$ wykorzystując N losowo wybranych punktów, a następnie przygotować dane niezbędne do wykreślenia zależności objętości kuli jednostkowej w n–wymiarowej przestrzeni euklidesowej od wymiaru przestrzeni n. Jako generator liczb losowych wykorzystaj *Mersenne Twister 19937* dostarczany przez plik nagłówkowy `random`.

Wykorzystując odpowiednią bibliotekę graficzną (np. Matplotlib-cpp lub ImPlot) lub zapisując przygotowane dane do pliku i korzystając z zewnętrznego narzędzia (np. skryptu napisanego w języku Python) sporządź wykres zależności objętości kuli jednostkowej w n–wymiarowej przestrzeni euklidesowej od wymiaru przestrzeni n.

Zadanie 4. `demere` – Paradoks kawalera de Méré.

Paradoks kawalera de Méré to problem z dziedziny rachunku prawdopodobieństwa postawiony przez francuskiego matematyka–amatora Antoine'a Gombauda (1607 – 1684). Dotyczy on serii rzutów symetrycznymi sześciennymi kostkami do gry i polega na rozstrzygnięciu, które spośród następujących dwóch zdarzeń jest bardziej prawdopodobne:

- zdarzenie A: wyrzucenie przynajmniej jednej szóstki w czterech rzutach jedną kostką,
- zdarzenie B: wyrzucenie przynajmniej jednej pary szóstek w dwudziestu czterech rzutach dwiema kostkami.

Analityczne rozwiązanie tego problemu podał Blaise Pascal (1623 – 1662).

Napisz szablon funkcji

```
template<typename RNG>
void DeMere(double &pA, double &pB, std::size_t N)
```

która posługując się metodą Monte Carlo znajduje przybliżone prawdopodobieństwa opisanych wyżej zdarzeń A i B. Funkcja `DeMere` przyjmuje argumenty `pA` i `pB` stanowiące referencje do zmiennych typu `double` oraz argument `N` odpowiadający liczbie N. Parametr `RNG` szablonu ma być typem reprezentującym generator liczb

losowych, który funkcja powinna wykorzystać do wylosowania punktów. Funkcja `DeMere` powinna przypisać obliczone prawdopodobieństwa zdarzeń A i B do zmiennych, odpowiednio, `pA` i `pB`.

Korzystając z tego szablonu napisz program `demere`, który czterokrotnie wypisze na standardowe wyjście prawdopodobieństwa zdarzeń A i B: za pierwszym razem program powinien podać prawdopodobieństwa znalezione przez Ciebie w literaturze, a następnie ich przybliżone wartości obliczone przez funkcję `DeMere` za pomocą trzech generatorów liczb losowych dostarczanych przez plik nagłówkowy `random`:

- `minstd_rand` – liniowy generator kongruentny *MINSTD*,
- `mt19937` – 32-bitowy generator *Mersenne Twister 19937*,
- `ranlux24` – 24-bitowy generator *RANLUX*.

Opracowanie: Bartłomiej Zglinicki.