Programowanie i metody numeryczne
zestaw zadań 10
Algebra liniowa, Całkowanie II: metody Monte Carlo

Opracowanie *Jędrzej Wardyn*

13 maja 2024

Polecam: kody C++, jak ktoś potrzebuje sobie C++ powtórzyć: Rozwiązania na githubie Kurs na youtubie

1 Całkowanie II: metodami Monte Carlo

Całkując metodą Monte Carlo przybliżamy całkę przez sumę::

$$I = \int_a^b f(x)\mathrm{d}x \approx \langle f \rangle = \frac{1}{N}\sum_{i=1}^{N} f(x_i)$$

dla odwzorowania $y = f(x)$.Z wikipedii nieco ogólniej

Napisz szablon do różnych metod całkowania za pomocą templatki:

```
template <typename T>
T integrate2D(T f, T xbegin, T xend, T ybegin, T yend, int N, mt19937& rng)
```

gdzie `T` oznacza nazwę zastosowanego typu (na przykład `double`, tylko nie macie go wpisywać w templatkę, a jedynie w funkcji main używacie konkretnego typu na wartości np `xbegin,xend` i wtedy kod przyjmie te wartości jako `T`.) `mt19937&` to typ generatora liczb losowych z `<random>` do zastosowania Masz do policzenia :

1. Dla sprawdzenia czy dobrze liczy: całkę z funkcji $f(x_i, y_i) = 1$ w przedziałach $x_i \in [0, 2]$ do $y_i \in [0, 3]$, pole prostokąta powinno nam dać po prostu $P = 2 \times 3 = 6$

2. Lemniskaty

$$(x^2 + y^2)^2 \leq 2(x^2 - y^2)$$

   w przedziale całkowania $\Omega = [-1.5, 1.5] \times [-0.6, 0.6]$. Następnie spróbuj zadać inny przedział całkowania.

3. Kardioidy której brzeg leży na: $(ax + x^2 + y^2)^2 = a^2(x^2 + y^2)$, gdzie $a = 1$, przy czym zdecyduj jaki zadasz przedział całkowania oraz ile całek policzysz (możemy podzielić tę figurę na prostokąty).

Aby oszacować przybliżony w sposób statystyczny błąd liczenia całek możemy użyć formuły:

$$\sigma_M = \sigma / \sqrt{N},$$

gdzie N-liczba punktów, a:

$$\sigma = \langle f^2 \rangle - \langle f \rangle^2$$

Tutaj więcej informacji

2 Algebra Liniowa i powtórka z klas C++

: Autor: Z.B.

Napisz szablon klasy

```
template <typename T>
class Vector
```

reprezentującej wektor n-wymiarowy, którego skladowe są liczbami reprezentowanymi przez typ T. W klasie tej zaimplementuj:

- konstruktor jednoargumentowy, ustalający wartość n równą liczbie przekazanej mu jako argument
- metodę Length zwracającą długość wektora (liczbę reprezentowaną przez typ T),
- operator== porównywania wektorów,
- jednoargumentowy operator zmiany znaku-,
- operator +dodawania wektorów,
- operator * mnozenia wektora przez liczbę (reprezentowaną przez typ T),
- operator* iloczynu skalarnego wektorów (wynik powinien być typu T)
- operator«, wypisujący do strumienia typu ostream wektor reprezentowany przez obiekt, oraz operator
- `>>` wczytujący odpowiedni wektor ze strumienia typu istream; przyjmij dowolny, wygodny dla Ciebie sposób tekstowego reprezentowania wektora,

Napisz szablon klasy

```
template <typename T>
class Matrix
```

reprezentującej macierz nxn, której składowe są liczbami reprezentowanymi przez typ T. W klasie tej zaimplementuj:

- konstruktor jednoargumentowy, ustalający wartość n równą liczbie przekazanej mu jako argument
- metodę Length zwracającą długość wektora (liczbę reprezentowaną przez typ T),
- operator `==` porównywania macierzy,
- jednoargumentowy operator zmiany znaku `-` ,
- operator `+` dodawania macierzy,
- operator `*` mnozenia macierzy przez liczbę (reprezentowaną przez typ T),
- operator `*` iloczynu skalarnego macierzy (wynik powinien być typu T)
- operator `<<`, wypisujący do strumienia typu ostream macierz reprezentowaną przez obiekt, oraz operator `>>` wczytujący odpowiedni wektor ze strumienia typu istream; przyjmij dowolny, wygodny dla Ciebie sposób tekstowego reprezentowania macierzy,

Metody:

1. metodę do LU decomposition

   ```
   pair<Matrix<T>, Matrix<T>> DecomposeLU(const Matrix<T> \&C)
   ```

   zwracającą parę macierzy złożoną z macierzy górnotrójkątnej i dolnotrójkątnej, których iloczyn jest równy macierzy przekazanej jako argument metody,

2. ```
   template <typename T>
   Vector<T> SolveU(const Matrix<T><<U,const Vector<T><<y)
   ```

   zwracającej wektor x stanowiący rozwiązanie równania $Ux = y$ przy założeniu, że macierz U jest dolnotrójkątna

3. ```
   template <typename T>
   Vector<T> SolveL(const Matrix<T> \&L, const Vector<T> \&y)
   ```

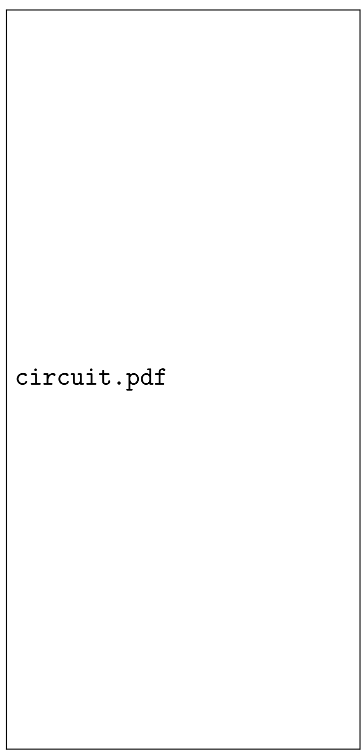
   zwracającej wektor x stanowiący rozwiązanie równania $Lx = y$ przy założeniu, ze macierz L jest górnotrójkątna,

4. ```
   template <typename T>
   Vector<T> Solve(const Matrix<T> &C, const Vector<T> &y)
   ```

   zwracającej wektor x stanowiący rozwiązanie równania $Cx = y$ dla dowolnej macierzy C; funkcja ta powinna wykorzystywać rozkład LU macierzy C.

Korzystając z tych szablonów, napisz program eqsolver, który wczytuje ze standardowego wejścia macierz C oraz wektor y, a następnie wypisuje na standardowe wyjście wektor x stanowiący rozwiązanie równania $Cx = y$.

2.1 Rozwiązywanie układu oporników, ale nie ręcznie: programem



Rysunek 1: Układ rezystorów, do którego podłączone jest stałe źródło napięcia +10V

Napisz program rozwiązujący i zastosuj go do powyższego układu rezystorów (patrz Rysunek 2.1).
   ```

Wystarczy, że zapiszesz 6 równań dla 6 oczek (pętli/loopów), które są widoczne [czyli nie trzeba równania dla kilku naraz]. Na przykład pierwsze oczko to:

$$10\text{V} = (1\Omega)I_1 + (25\Omega)(I_1 - I_2) + (50\Omega)(I_1 - I_3)$$

(Nie trzeba zapisywać równań dla węzłów, wystarczą oczka.) Mając zapisaną macierz rezystancji oraz wektor napięć dokonaj:

1. Rozkładu **LU** wykorzystując metodę Dolittle
2. Policz wyznacznik znając macierz **U**
3. Na macierzy **U** dokonaj eliminacji Gaussa i znajdź wektor natężeń prądu

Jak wykonać rozkład **LU** i eliminację gaussa? Linki:
LU metodą Dolittle
Eliminacja Gaussa