Bartosz Kirker
Daniel Szura
Igor Skiba

Liczenie całki ze stringa

Założenia Programu:

1. Scałkowanie podanego algorytmu metodą prostokątów.
2. Wykorzystanie metody Monte-Carlo.
3. Wyliczenie podanego algorytmu.

Teoria:

ONP - **Odwrotna notacja polska -** sposób zapisu wyrażeń arytmetycznych, w którym znak wykonywanej operacji umieszczony jest po operandach (zapis postfiksowy), a nie pomiędzy nimi jak w konwencjonalnym zapisie algebraicznym (zapis infiksowy) lub przed operandami jak w zwykłej notacji polskiej (zapis prefiksowy). Zapis ten pozwala na całkowitą rezygnację z użycia nawiasów w wyrażeniach, jako że jednoznacznie określa kolejność wykonywanych działań.

```
(2+3)*5
```

w ONP wygląda tak:

```
2 3 + 5 *
```

natomiast:

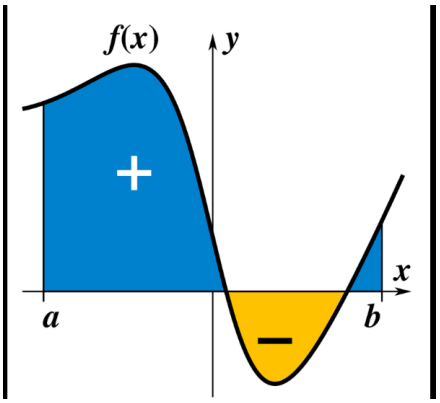
```
((2+7)/3+(14-3)*4)/2
```

zapisuje się następująco:

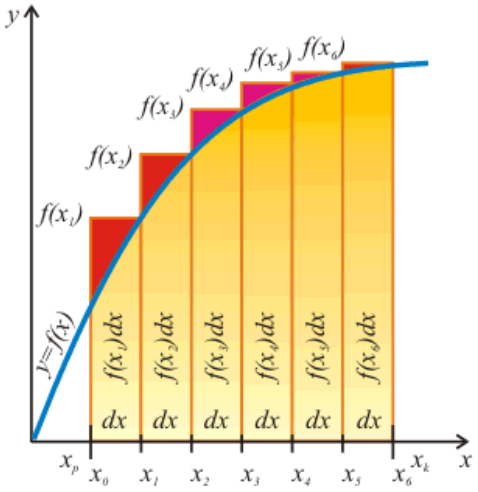
```
2 7 + 3 / 14 3 - 4 * + 2 /
```

Monte-Carlo - jest stosowana do modelowania matematycznego procesów zbyt złożonych aby można było przewidzieć ich wyniki za pomocą podejścia analitycznego. Istotną rolę w metodzie MC odgrywa losowanie wielkości charakteryzujących proces, przy czym losowanie dokonywane jest zgodnie z rozkładem, który musi być znany.

Całka - ogólne określenie wielu różnych, choć powiązanych ze sobą pojęć analizy matematycznej. Najczęściej przez „całkę” rozumie się całkę oznaczoną lub całkę nieoznaczoną, choć istnieje wiele innych odmian całki. Ścisłe definicje można znaleźć w artykułach dotyczących poszczególnych całek.



Metoda prostokątów - W metodzie prostokątów korzystamy z definicji całki oznaczonej Riemanna , w której wartość całki interpretowana jest jako suma pól obszarów pod wykresem krzywej w zadanym przedziale całkowania $<x_p,x_k>$. Sumę tę przybliżamy przy pomocy sumy pól odpowiednio dobranych prostokątów.



Program:

Program jest napisany w języku programowania c++11. C++11 jest to zaktualizowane wersja języka c++. Zostały do niej dodane nowe elementy jak na przykład operator „auto” co pozwala na automatyczną definicję typu zmiennej. Program został napisany w Microsoft Visual Studio Community 2015.

Przykładowy output programu:

```
Hello World
0,54030230586814

-5,41619081605703
```

Program zawiera strukturę klasową. Oznacza to, że posiada nadrzędną klasę, po której dziedziczą pozostałe klasy.

```
ref class Op : IDisposable{
    public:
        double virtual licz(double x, double y){
            return 0;
        }
        !Op(){
        }
        ~Op(){
        }
};
```

Następnie mamy klasy, które definiują podstawowe działania arytmetyczne:

```
ref class Liczba :public Op{
private:
    double x;
public:
    Liczba(double x){
        this->x = x;
    }
    double virtual licz(double x, double y)override {
        //double wynik = this->x->licz(x,y) + this->y->licz(x,y);
        //Console::WriteLine("liczba"+ x);
        return x;
    }
};
```

Kolejny etapem struktury programu, jest prasowanie wyrażeń matematycznych takich jak: sin, cos, pow, sqrt:

```
ref class Sqrt: public Op{
private:
    Op ^ x, ^ y;
public:
    Sqrt(Op ^ x, Op ^ y){
        this->x = x;
        this->y = y;
    }
    double virtual licz(double x, double y) override{
        return Math::Sqrt(this->x->licz(x,y));
    }
};
```

Każda klasa programu używa przeładowanej metody „Licz” z klasy nadrzędnej. Elementem użytym z języka c++11 jest symbol „ref”, który każe zwracać z klasy wartość oraz „override”, który mówi, że ta metoda jest przeładowaną metodą z klasy bazowej.

Następnie mamy budowę stosu na który w odpowiedni sposób ładujemy elementu algorytmu podanego w stringu oraz następnie te elementy są z niego zrzucane i zostają podane obliczeniu:

```
Op ^ toTree( String^ str ){
    Op toTree;
    String^ delim = L" ";
    array<Char>^ delimiter = delim->ToCharArray( );
    array<String^>^ words;

    words = str->Split( delimiter );
    //Op *tmp0 = nullptr;
    Stack ^stack = gcnew Stack();

    Double wynik;
    for (int word=0; word<words->Length; word++){
        String^ w = words[word];
        if( Double::TryParse(w, wynik )== false ){
            if( w == "+" ){
                Op ^tmp = (Op^)stack->Peek();
                stack->Pop();
                Op ^tmp1 = (Op^)stack->Peek();
                stack->Pop();
                stack->Push(( gcnew Dodaj( tmp, tmp1 )));
            }else if( w == "-" ){
                Op ^tmp = (Op^)stack->Peek();
                stack->Pop();
```

Metoda „toTree” wskazuje na typ „Op”. Jest to nasza klasa bazowa. Operator „^” jest typem wskaźnikowym, który mówi nam że metoda, zmiana jest tworzona dynamicznie. Zmiennę zainicjowane za pomocą operatora „^” tworzymy za pomocą „gcnew”, a nie zwykłym ”new”, ponieważ ten operator automatycznie zwalnia pamięć zarezerwowaną przez wcześniej zdefiniowaną zmienną. Jest to taki prosty „garbage colletion”.

Ostatnim elementem kodu jest funkcja licząca pole pod całką za pomocą metody prostokątów:

```
double pole( Op ^f, int n, double xs, double xk, double ys, double yk ){
    int trafien = 0;
    double xl = xk - xs;
    double yl = yk - ys;
    double fs = 0;
    Random r;
    double dx = xk - xs;
    double dy = yk - ys;
    for( int i = 0; i < n; i++ ){
        double x = xs + r.NextDouble()*dx;
        double y = ys + r.NextDouble()*dy;
        fs += f->licz(x,y) / n;
    }
    return dx*dy * fs;
}
```

Metoda wykorzystuje metode Monte Carlo poprzez korzystanie z losowych zmiennych.

Metoda Main wywołuje metody programu:

```
int main(array<System::String ^> ^args)
{
    Console::WriteLine(L"Hello World");
    Op ^ function = toTree( L" X cos ");
    double x = 1;
    double y = 1;
    Console::WriteLine( function->licz(x,y) + L"\n");
    Console::WriteLine( pole(function,1000000,0,10,0,10));
    getch();
    return 0;
}
```

W stringu podajemy już algorytm w zapisię ONP.

Jedyną wadą programu jest to, że trzeba podać algorytm w konkretny sposób. Gdyż nie zdążyliśmy zaprogramować kolejności wykonywania działań przez program. Dlatego w stringu trzeba podać w poprawnej kolejności wykonywanie działań matematycznych.