# Zasady programowania strukturalnego II – projektowanie

|  |  |  |  |
| --- | --- | --- | --- |
| Imię i Nazwisko | Łukasz Jaśkowski | Grupa | 14 |
| Prowadzący | Mgr inż. Adam Piechna | | |
| Temat projektu | Projekt 7 - Algorytm genetyczny | | |

1. ***Opis projektu***

Program implementujący algorytm genetyczny do rozwiązania problemu optymalizacji – znalezienia minimum wybranej funkcji w określonym przedziale. Program wykorzystuje strukturę dynamiczną danych opartą o ręcznie tworzone listy, a także operacje na plikach(odczyt i zapis).

***1a. Algorytm***

Idea algorytmu opiera się na zjawisku doboru naturalnego i dziedziczenia znanego z nauk przyrodniczych. Zgodnie z tym zjawiskiem osobniki lepiej przystosowane do warunków środowiska mają większą szansę na przetrwanie i przekazanie swojej informacji genetycznej potomkom. Krzyżowanie materiału genetycznego między osobnikami prowadzi do coraz to lepszego przeciętnego dostosowania się ich kolejnych pokoleń do warunków środowiska. Ewolucja jest więc pewnym procesem optymalizacji.

Przekładając to na temat projektu – pokolenie jest zbiorem osobników o określonej liczebności. Osobnik z kolei reprezentuje pewną liczbę(punkt w przypadku funkcji wielu zmiennych) zawartą w rozpatrywanym w zadaniu przedziale. Genotyp osobnika jest zakodowanym zapisem tej liczby(punktu) w systemie binarnym. Pierwszym pokoleniem jest pewna ilość osobników, których materiał genetyczny został wylosowany. Funkcja określająca przystosowanie osobnika jest nazywana inaczej funkcją oceny. W tym przypadku jest to funkcja, której minimum szukamy. Podając zdekodowany materiał genetyczny osobnika(czyli de facto liczbę lub punkt, który on reprezentuje) jako argument funkcji celu otrzymujemy wartość funkcji dla tej liczby(punktu). Osobniki dla których funkcja celu przyjmuje niższe wartości mają większą szansę na przekazanie swojego materiału genetycznego dalej, natomiast te gorzej przystosowane(czyli z większymi wartościami funkcji oceny) mają tę szansę odpowiednio mniejszą. Następnie pomiędzy osobnikami następuje zjawisko krzyżowania(zdefiniowane jako wymiana określonej liczby genów między nimi) z czego powstają nowe osobniki. Dość rzadko, ale tak samo jak w naturze występuje zjawisko mutacji – w projekcie reprezentowane jest poprzez zmianę kilku losowych bitów osobnika na przeciwny. Po przejściu jednego takiego cyklu liczba osobników pozostaje niezmienna, a ich średnie przystosowanie powinno być lepsze niż w pokoleniu poprzedzającym. Następnie pokolenie drugie zostaje poddane takiemu samemu procesowi jak pokolenie pierwsze. Taki cykl zachodzi określoną ilość razy. Na samym końcu z ostatniej populacji wybierany jest osobnik o najlepszym przystosowaniu. Jeżeli wykonaliśmy odpowiednią ilość cykli rozwiązanie to powinno być równe bądź bardzo bliskie szukanemu minimum.

1. ***Opis formatu danych wejściowych/wyjściowych***

Program przewiduje podanie danych wejściowych do programu w dwojaki sposób. Użytkownik może zarówno wprowadzić je z klawiatury jak i wczytać z pliku. W drugim przypadku powinny one być zapisane w pliku tekstowym o rozszerzeniu \*.txt. Nazwa pliku jest dowolna – użytkownik programu będzie w odpowiednim momencie poproszony o jej podanie. W pliku powinno znaleźć się 5 liczb całkowitych oddzielonych spacjami, które będą oznaczać kolejno:

* początek przedziału, w którym będziemy szukać minimum,
* koniec przedziału,
* liczbę bitów na których będzie zapisywany materiał genetyczny jednego osobnika; należy pamiętać, że im mniejsza będzie ta liczba tym mniej dokładny będzie wynik(rozdzielczość wyniku okreslona jest wzorem: (koniec\_przedzialu-poczatek\_przedzialu)/(2^liczba\_bitow-1) ), ale podanie jej zbyt dużej spowoduje dłuższe działanie programu,
* liczbę osobników w jednej populacji,
* liczbę pokoleń;

Dane wyjściowe mają być docelowo zapisywane w pliku tekstowym(użytkownik zostanie poproszony o podanie jego nazwy) i zawierać dwie liczby zmiennoprzecinkowe:

* liczbę określającą współrzędną x-ową szukanego minimum,
* wartość funkcji f(x) dla argumentu z powyższego punktu – czyli szukane minimum funkcji;

1. ***Podział na moduły***

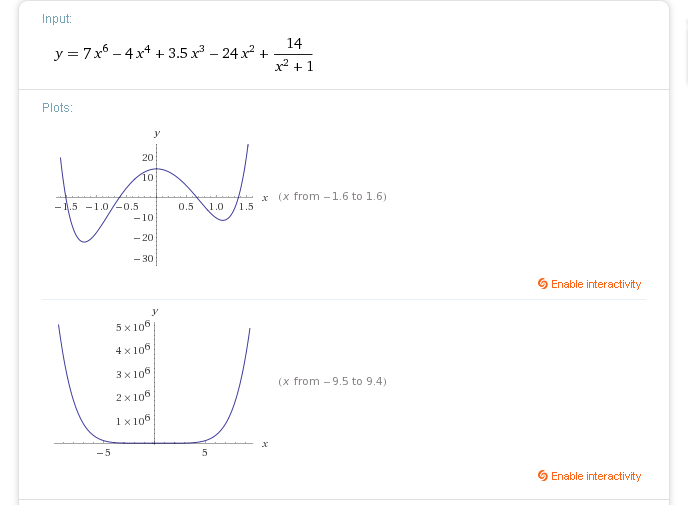
Program(w wersji konsolowej) składa się z czterech modułów:

* + 1. **Interfejs**(moduł odpowiedzialny za komunikację z użytkownikiem). Składa się z jednej funkcji:
* **void start()** – funkcja bezargumentowa, obsługuje cały interfejs programu, ustawia punkt startowy generatora pseudolosowego(potrzebny przy późniejszym losowaniu pierwszej populacji), odpowiada za pierwszy kontakt z użytkownikiem i pobranie od niego danych wejściowych, następnie uruchamia funkcję operującą na tych danych, a otrzymane wyniki zapisuje do pliku;
  + 1. **Dane**(moduł, a jednocześnie nazwa klasy, odpowiedzialny za pobranie i przechowywanie danych wejściowych od użytkownika). Klasa składa się z 8 prywatnych pól i 14 publicznych metod. Są to kolejno:
* pięć pól typu **int** (**początek\_przedzialu\_**, **koniec\_przedzialu\_**, **liczba\_bitow\_**, **liczba\_osobnikow\_**, **liczba\_pokolen\_**) - są to wartości pobierane od użytkownika na starcie programu,
* **double rozdzielczosc\_** - pole uzupełniane w trakcie działania programu, przyjmuje wartość zależną od danych wejściowych zgodnie ze wzorem podanym wyżej(w punkcie dotyczącym formatu danych wejściowych); pole dodane dla wygody użytkowania, ponieważ nie musimy obliczać jej wartości za każdym razem(jest to wartość niezmienna w trakcie działania całego programu i używana nader często),
* **double xMIN\_,yMIN\_** - pole uzupełniane tuż przed zapisem danych do pliku, dodane dla wygody użytkowania, oznacza kolejno współrzędną x-ową i y-ową szukanego minimum funkcji,
* **Dane()** – konstruktor klasy,
* **int Poczatek(),Koniec(),LBitow(),LOsobnikow(),LPokolen()** – metody zwracające kolejne pola prywatne danego obiektu, dodane w celu używania ich w funkcjach innych niż zaprzyjaźnione(pozwalają odczytać wartości pól prywatnych danego obiektu),
* **double Rozdzielczosc(),xMIN(),yMIN()** – metody o działaniu analogicznym do pięciu poprzednich,
* **bool wczytajDaneZPliku()** – metoda odpowiedzialna za wczytanie danych z pliku, odporna na podanie nieprawidłowej nazwy pliku bądź nieprawidłowych danych wejściowych(o czym informuje użytkownika),
* **bool wczytajDaneZKlawiatury()** – metoda odpowiedzialna za wczytanie danych z klawiatury, odporna na podanie nieprawidłowych danych wejściowych(o czym informuje użytkownika),
* **void zapiszDoPliku()** – metoda odpowiedzialna za zapisanie współrzędnej x-owej i y-owej do pliku o nazwie podanej przez użytkownika, w przypadku gdy dany plik nie istnieje – zostanie on stworzony,
* **void ustawRozdzielczosc()** – metoda odpowiedzialna za ustawienie pola rozdzielczosc\_, jest wywoływana tuż po poprawnym pobraniu danych od użytkownika,
* **void ustawWyjscie(double xMIN, double yMIN)** – metoda odpowiedzialna za ustawienie pól xMIN i yMIN, wywoływana po stworzeniu ostatniej populacji, tuż przed zapisem danych do pliku,
  + 1. **Osobnik**(moduł(a jednocześnie nazwa klasy) odpowiedzialny za operacje na poszczególnych osobnikach populacji). Klasa składa się z 5 prywatnych pól i 11publicznych metod(a także 4 funkcji zaprzyjaźnionych). Są to kolejno:
* **bool \*genotyp\_ -** wskaźnik na tablicę typu bool reprezentującą genotyp danego osobnika jako liczbę zapisaną w systemie binarnym,
* **int rozmiar\_ -** liczba określająca rozmiar powyższej tablicy typu bool,
* **double zdekodowany\_ -** wartość obliczana w trakcie programu na podstawie genotypu danego osobnika i danych wejściowych, określa wartość współrzędnej x-owej reprezentowanej przez danego osobnika,
* **double przystosowanie\_** - wartość obliczana w trakcie działania programu, równa jest wartości funkcji, której minimum szukamy przy argumencie równym polu zdekodowany\_,
* **Osobnik\* następny\_osobnik\_** - wskaźnik na kolejny obiekt typu Osobnik; używany, ponieważ każda populacja jest zapisywana w formie listy jednokierunkowej,
* **Osobnik(int x)**  - konstruktor klasy, ustawia pole rozmiar\_ równe argumentowi funkcji oraz rezerwuje dynamicznie miejsce w pamięci na genotyp danego osobnika(tablica typu bool o długości równej x),
* **~Osobnik()** – destruktor klasy, zajmuje się zwolnieniem pamięci po dynamicznie zarezerwowanej tablicy dla genotypu danego osobnika,
* **bool\* zwrocGenotyp(), double zdekodowany(), double przystosowanie(), Osobnik\* adresNastepnego()** – metody zwracające wartości bądź wskaźniki na odpowiednie prywatne pola obiektu; wykorzystywane w funkcjach innych niż zaprzyjaźnione,
* **void wyswietlPopulacje() –** metoda wyświetlająca kolejno pola genotyp\_, zdekodowany\_ i przystosowanie\_ wszystkich osobników, aż do końca listy,
* **void losuj() –** metoda losująca materiał genetyczny danego osobnika(tablica genotyp\_ uzupełniana jest losowo zerami i jedynkami, używana podczas tworzenia pierwszej populacji,
* **void mutuj() –** metoda losująca i negująca losowy bit z genotypu osobnika,
* **void sortuj()** – metoda odpowiedzialna za posortowanie listy osobników malejąco wg wartości pola zdekodowany\_,
* **Osobnik\* zwrocAdresITegoElementu(int x) –** metoda zwracająca wskaźnik na element listy jednokierunkowej o numerze równym x;
* **friend void usunPopulacje(Osobnik \*&wsk) –** funkcja odpowiedzialna za usunięcie całej populacji i przyjmująca jako argument wskaźnik na pierwszego osobnika z listy,
* **friend Osobnik\* tworzPopulacje(Dane\* wejscie)** – funkcja odpowiedzialna za stworzenie pierwszej populacji, materiał genetyczny osobników jest losowany, obliczane i ustawiane są pola zdekodowany\_ i przystosowanie\_,
* **friend void tworzKolejnaPopulacje(Osobnik\* &stara, Dane\* wejscie, int sumaPrzystosowan, int\* tablicaPrzystosowan) –** funkcja odpowiedzialna za stworzenie nowej populacji na podstawie starej; wykorzystywana jest tutaj metoda selekcji rankingu liniowego – z populacji poprzedniej losowanych jest dwóch osobników(ich szanse na wylosowanie rosną liniowo zgodnie z zajmowanym na posortowanej liście miejscem), następnie następuje krzyżowanie ich genotypu w połowie długości, powstają dwa osobniki potomne, z których każdy zawiera połowę informacji pochodzącej od jednego z rodziców i połowę od drugiego; później następuje zjawisko mutacji(dla każdego z osobników populacji potomnej prawdopodobieństwo jego wystąpienia wynosi 10%); po przeprowadzeniu tych operacji zostają wyliczone i ustawione pola zdekodowany\_ i przystosowanie\_ dla każdego z osobników,
* **friend void krzyzuj(Osobnik \*wsk1, Osobnik \*wsk2, Osobnik \*docelowy)** – funkcja odpowiedzialna za krzyżowanie materiału genetycznego między dwoma osobnikami, osobnik docelowy posiada pierwszą połowę materiału genetycznego od osobnika wsk1 i drugą od wsk2,
  + 1. **Algorytmy**(moduł, w którym zawarte są wszelkie inne funkcje(funkcja celu, dekodowanie genotypu, przeprowadzenie cyklu etc.). Moduł składa się z 9 funkcji. Są to kolejno:
* **void operujNaDanych(Dane\* wejście) –** funkcja przeprowadzająca większość operacji na danych wejściowych poprzez wywoływanie odpowiednich funkcji i metod, odpowiedzialna jest za ustawienie pola rozdzielczosc\_ w zmiennej podanej jako argument, tworzenie pierwszej populacji, sortowanie i wyświetlenie jej, wywołanie funkcji tworzącej kolejne populacje oraz ostatecznie ustawienie pól xMIN\_ i yMIN\_ zmiennej podanej jako argument i usunięcie ostatniej populacji,
* **void cykl(Osobnik\* &populacja, Dane\* wejście)** – funkcja odpowiedzialna jest za stworzenie tzw. tablicy przystosowań(jej zastosowanie opisane jest później), a także tworzenie nowych populacji, sortowanie i wyświetlanie ich na konsoli; pierwszy z argumentów funkcji jest wskaźnikiem na pierwszą populację stworzoną wcześniej i na niej przeprowadzane są kolejne operacje, drugi z argumentów zawiera dane dotyczące m.in. liczby cykli wykonywanych w tej funkcji czy ilości osobników danej populacji,
* **int binToDec(bool \*tab\_bool, int liczba\_bitow)** – funkcja przyjmująca jako argument wskaźnik na tablicę typu bool, a także jej długość; odpowiedzialna jest za rozkodowanie liczby zapisanej w systemie binarnym(jako ciąg 0 i 1) na liczbę w systemie dziesiętnym, która jest następnie zwracana,
* **double rozdzielczosc(int poczatek\_przedzialu, int koniec\_przedzialu, int liczba\_bitow)** – funkcja oblicza rozdzielczość zgodnie ze wzorem podanym wyżej, a następnie ją zwraca; wywoływana w programie raz – przy ustawianiu pola rozdzielczosc\_ klasy Dane;
* **double dekoduj(bool\* tab\_bool, int poczatek\_przedzialu, int liczba\_bitow, double rozdzielczosc)** – funkcja odpowiedzialna za zdekodowanie liczby zapisanej w systemie binarnym(jako ciąg 0 i 1 w postaci tablicy typu bool) na wartość, którą reprezentuje w danym przedziale oraz jej zwrócenie,
* **double f1(double x) –** funkcja(**R->R**), której minimum szukamy, jako argument przyjmuje jedną liczbę x zmiennoprzecinkową i oblicza wartość funkcji y=f1(x) zgodnie ze wzorem f1(x)=7x^6-4x^4+3,5x^3-24\*x^2+14/(x^2+1), a następnie ją zwraca; szczegółowy opis wybranej funkcji znajduje się na samym końcu dokumentacji,
* **void zamien(bool &x1, bool &x2)** – funkcja zamieniająca wartościami dwie zmienne typu bool(wywoływana podczas operacji krzyżowania osobników,
* **int sumaPrzystosowan(int LOsobnikow) –** funkcja licząca sumę przystosowań – wartość potrzebną przy wybranej przeze mnie metodzie selekcji osobników(metoda selekcji rankingowej), oblicza wartość sumy liczb od 1 do n(wartość równa argumentowi funkcji) zgodnie ze wzorem ((1+n)/2)\*n;
* **int\* tablicaPrzystosowan(int LOsobnikow)** – funkcja tworząca tablicę przystosowań i zwracająca wskaźnik na nią, kolejne elementy tablicy mają wartość równą sumie liczb od 1 do (x+1), gdzie x reprezentuje indeks danego elementu, tablica wykorzystywana jest podczas metody selekcji rankingu liniowego – losowana jest liczba z zakresu od 1 do sumaPrzystosowan(sP), a następnie sprawdzane jest w jakim zakresie się ona mieści(między którymi polami powyższej tablicy), wybierany jest przez to odpowiedni osobnik do krzyżowania; zgodnie z definicją metody rankingu liniowego pierwszy osobnik(najgorzej przystosowany) ma szansę na wylosowanie 1/sP, drugi 2/sP, kolejny 3/sP itd., szansa na ich wybranie rośnie liniowo(w przypadku 5 osobników jest to kolejno(1/15, 2/15, 3/15, 4/15 i 5/15),

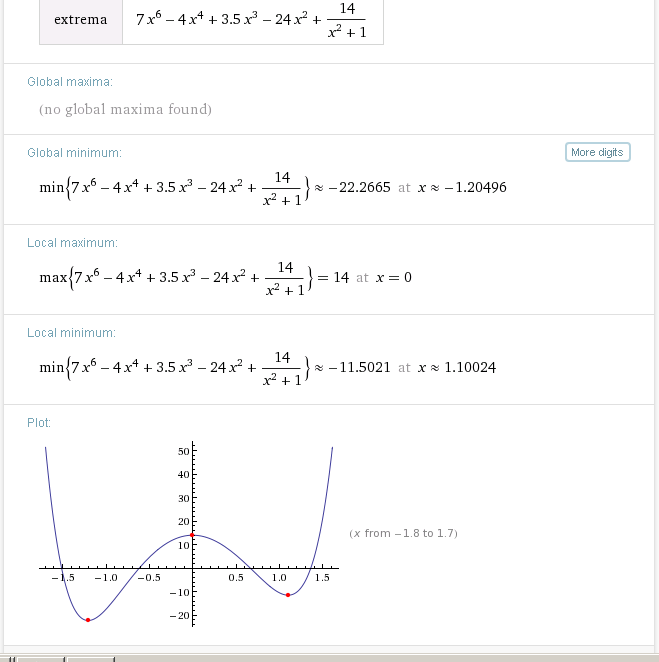
1. ***Uwagi końcowe***

Funkcja, której minimum szukam określona jest wzorem :

W trakcie testów programu rozpatrywałem szukanie jej minimum na przedziale (-10;10). Funkcja została wybrana celowo ponieważ posiada dwa bliskie sobie ekstrema minimum, co prezentuje poniższy wykres.



Rys.1. Wykres badanej funkcji wykonany przy użyciu programu WolframAlpha



Rys.2. Extrema funkcji obliczone za pomocą programu WolframAlpha

Jak widać funkcja posiada dwa ekstrema minimum o zbliżonych sobie wartościach. Znacząco utrudnia to szukanie minimum globalnego, ponieważ algorytm może zbiegać do minimum lokalnego. Na drodze prób i błędów postanowiłem zwiększyć zakres mutacji od negacji jednego bitu osobnika do negacji połowy jego bitów. Pozwoliło to „uciekać” z minimum lokalnego i zbiegać do szukanego minimum globalnego.

Wykonując wiele testów na przedziale (-10;10) doszedłem do wniosku, że aby algorytm znajdował właściwe ekstremum tej funkcji z rozsądną dokładnością należy przeprowadzić operacje na populacji ok. 25 osobników o genotypie ok. 25 bitowym. Ilość cykli, które dają dziewiędziesięcio-kilkuprocentową skuteczność to ok.500. Czasami jednak algorytm znajduje minimum globalne w mniej niż 100 cyklach.

.