Grupa projektowa: poniedziałek/N 11:15

Prowadząca: Dr.inż. Ewa Szlachcic

Data: 02.06.2020

Sprawozdanie

Teoria i metody optymalizacji

Zadanie znalezienia minimum funkcji nieliniowej ciągłej przy ograniczeniach nieliniowych

– metoda Powell / Gauss’a – Seidl’a

Dariusz Talik

Bartosz Lodek

1. Sformułowanie zadania optymalizacji

Zadaniem optymalizacji do zrealizowania w ramach projektu było poszukiwanie minimum funkcji f(x) nieliniowej, ciągłej z ograniczeniami, metodą Powella z metodą Gauss’a/Seidl’a. Należało do tego dobrać bezgradientową metodę minimalizacji w kierunku. W projekcie zostanie użyta metoda złotego podziału ze względu na charakteryzującą ją dobra zbieżność. Jest to optymalizacja statyczna.

Celem projektu było także stworzenie aplikacji implementującej algorytm optymalizacji z graficznym interfejsem użytkownika. Kolejne kroki tworzenia aplikacji oraz teoria, o którą oparto się w procesie tworzenia programu zostaną przedstawione w kolejnych punktach.

1. Opis zadania optymalizacji
   1. Programowanie nieliniowe

Problem, który należało rozwiązać w wykonanym przez nas projekcie polegał na znalezieniu wektora x̂ minimalizującego skalarną funkcję celu F= f(x) (gdzie x jest n-wymiarowym wektorem kolumnowym), z założeniem, że znamy postać analityczną funkcji.

Wartość f(x̂) jest ekstremum warunkowym funkcji f(x). Wartość x̂ jest rozwiązaniem zadania programowania.

* 1. Metoda bezgradientowa

Metody te nie wymagają znajomości gradientu (czyli pierwszej pochodnej) funkcji, lecz korzystają tylko z informacji o jej wartości. Często nazywane są one również metodami bezpośrednich poszukiwań (direct search).

Zarówno metody bezgradientowe, jak i gradientowe wykorzystują trzy podstawowe sposoby poszukiwania ekstremum w kierunku - są to metody: złotego podziału, interpolacji kwadratowej oraz interpolacji sześciennej. W projekcie zastosowano metodę złotego podziału.

* 1. Algorytm Powella z ograniczeniami

Istotą metody jest poszukiwanie ekstremum warunkowego przez ciąg kolejnych, minimalizacji bezwarunkowych zmodyfikowanej funkcji celu. W trakcie obliczeń “przesuwana” jest funkcja kary przy czym przesunięcie to uzależnił od wartości przekroczonych ograniczeń. Stąd też, przyjęto następującą postać modyfikacji funkcji celu

gdzie:

m = liczba ograniczeń

n = liczba argumentów

Przed przystąpieniem do algorytmu należy dobrać dane wejściowe takie jak:

* punkt startowy x0,
* maksymalną wartość przekroczenia ograniczenia w punkcie startowym c
* wymaganą dokładność uwzględnienia ograniczeń w chwili zakończenia działania procedury

Cmin

Przebieg algorytmu jest następujący:

Dokonaj minimalizacji funkcji oraz otrzymany punkt ekstremalny podstaw w miejsce , a ponadto c w miejsce .

Oblicz w punkcie wartość ograniczeń  dla i = 1,…,m oraz nową wartość zgodnie z zasadą

.

Zbadaj czy spełnione zostało kryterium na „minimum” tzn czy . Jeśli tak to zakończ działanie procedury, przejdź do metody Gauss’a-Seidl’a, natomiast jeśli nie, to przejdź do kolejnego punktu.

Zbadaj czy po minimalizacji (krok 2) nastąpiło zmniejszeni naruszenia ograniczeń tzn. czy . Jeśli tak, to przejdź do wykonania kroku 8, natomiast w przeciwnym razie podstaw na miejsce c jego wartość jego wartość przed minimalizacją tzn. .

Dla gdzie zmień wartość parametru według reguły

Przy czym 0 < <1 oraz są współczynnikami dobieranymi eksperymentalnie. Powell w swojej procedurze przyjął = ¼ oraz = 10.

Postaw k +1 w miejsce k oraz przyjmując ostatnio wyliczony jako nowy punkt startowy powtórz krok 2.

Jeśli k = 0 lub k-1 iteracji wykonywany był krok 6 to:

- (i) zmień wartość w myśl zasady

Oraz podstaw = dla i = 1,…,m, a następnie przejdź do wykonania kroku 7,

- (ii) natomiast w przeciwnym przypadku zbadaj warunek czy . Jeśli jest on

spełniony, to wykonaj czynności (i) kroku 8, a jeśli nie przejdź do wykonania kroku 6.

* 1. Algorytm Gaussa- Seidela

Algorytm Gaussa- Seidela opiera się na wykorzystaniu optymalizacji kierunkowych, realizowanych kolejno względem kierunków określonych przez wektory przyjętej bazy ortogonalnej. Polega na poruszaniu się w dziedzinie funkcji w taki sposób, aby osiągnąć szukane minimum za pomocą zmiany tylko jednej współrzędnej w jednym kroku. Długość kroku nie jest stała, określa ją algorytm minimalizacji wzdłuż kierunku. Jako bazę kierunków przyjęto wersory kartezjańskiego układu współrzędnych:

e 1= [1 0 0 … 0] e 2= [0 1 0 … 0]

Ponadto algorytm GausaSeidela jest zbieżny lokalnie, ponieważ potrafimy pokazać, że ciąg jest zbieżny gdy punkt początkowy x0 należy do stosownego otoczenia rozwiązania x0.

Kryterium stopu dla algorytmu miało wynosić:

- bezwzględna różnica z wartość funkcji aktualnego kroku i poprzedniego jest mniejsza od założonego epsilonu,

- norma różnicy aktualnego wektora i poprzedniego jest mniejsza od epsilonu

* 1. Metoda złotego podziału

Metoda złotego podziału charakteryzuje się dobrą zbieżnością przy prostocie obliczeń, co znacznie przyspiesza jej działanie w porównaniu do pozostałych opisanych metod. Jednak przy dużej dokładności szybkość zbieżności jest niewielka.

Działanie metody złotego przedziału opiera się na zmniejszaniu podprzedziały zawierającego poszukiwane przez nas minimum o określony współczynnik 𝞽 w każdej kolejnej iteracji, przy czym 𝞽 zawiera się w przedziale (0;1)

Pierwszym krokiem algorytmu jest wyznaczenie punktów pomocniczych a1 i a2 oraz obliczenie wartości funkcji minimalizowanej w tych punktach. Następnie, jeśli wartość funkcji w punkcie a1 jest większa niż w punkcie a2 to w następnej iteracji a = a1, natomiast b pozostaje takie samo. Ponadto kolejna wartość punktu a1 będzie równa poprzedniej wartości punktu a2, a aktualne a2 będzie wynosić a + 𝞽 (b - a). Natomiast jeżeli wartość funkcji minimalizowanej w punkcie a1 jest mniejsza lub równa wartości funkcji minimalizowanej w punkcie a2 to w następnej iteracji punkt a pozostaje bez zmian, a punkt b = a2. Ponadto wartość punktu a2 w kolejnej iteracji będzie równa poprzedniej wartości punktu a1, a aktualna wartość a1 będzie równa a + (1 - 𝞽) (b - a). Kroki powtarzamy do momentu aż różnica między wartościami b i a będzie mniejsza od ustalonej wcześniej wartości kryterium stopu ɛ.

1. Środowisko numeryczne

W projekcie wykorzystany został język Python, wybór został dokonany na podstawie prostoty i czytelności składni ułatwiającej używanie oraz rozumienie kodu. Zastosowano biblioteki takie jak Numpy, Py\_expression\_eval, Plot, Multiprocessing oraz Matplotlib.pyplot

Numpy - jest biblioteką Pythona służącą do obliczeń naukowych. Jako biblioteka NumPy dostarcza listę matematycznych funkcji użytecznych w takich zagadnieniach jak: algebra liniowa, transformacje Fouriera oraz generowanie liczb losowych,

Py\_expression\_eval – biblioteka dostarcza funkcje do analizy, oszacowywania oraz upraszczania wyrażeń matematycznych, tzw. Parser.

Plot – służy do wyświetlania wyników w postaci grafu,

Multiprocessing – jest to pakiet języka Python który wspiera tworzenie procesów za pomocą interfejsu API,

Matplotlib.pyplot – biblioteka umożliwia wyświetlanie danych w sposób zbliżony do środowiska MATLAB.

* 1. Struktura plików
* Model.py
* GoldenSection.py
* Plot.py
* restApi.py
* Client

Model.py - jest plikiem wykonawczym w który posiada w sobie funkcje reprezentujące kolejne kroki algorytmu. Funkcja celu, ograniczenia oraz parametry optymalizacji wprowadzane są w części '\_\_main\_\_'.

GoldenSection.py - zawiera logikę funkcji złotego podziału oraz zwraca wartość środkową.

Plot.py - odpowiada za rysowanie wykresów

restApi.py - jest wyprowadzeniem aplikacji w języku Python na aplikację webową napisaną w języku JavaScript React.

Client - zawiera w sobie front-end aplikacji

* 1. Interfejs aplikacji



Po wpisaniu adresu serwera z uwzględnieniem portu :3000, otrzymujemy podgląd panel głównego. W tym miejscu wprowadzamy funkcję celu, ograniczenia, wartości początkowe x1 i x2, dokładność epsilon, oraz ograniczenie liczby iteracji.

Następnie klikamy przycick GENERATE i otrzymujemy wyniki w postaci:

(wektor X) oraz wartość funkcji celu.

Otrzymujemy również informację na temat wartości parametrów i funkcji celu dla każdego kroku algorytmu.

1. Błędy wynikające z przyjętego środowiska

Liczby w języku Python, są tworzone za pomocą literałów liczbowych oraz zwracane jako wyniki przez operatory arytmetyczne i wbudowane funkcje arytmetyczne. Obiekty liczbowe są niezmienne - po utworzeniu nie zmieniają nigdy swojej wartości. Liczby w Pythonie są silnie powiązane z matematycznym pojęciem liczby, lecz nałożone są na nie pewne ograniczenia związane z reprezentacją liczb w komputerach.  Python rozróżnia liczby zmiennoprzecinkowe (floating point) które zapisywane są na 64 bitach. Mają one złożoną budowę oraz ograniczony zakres i precyzję. Do wad należą:

- Brak możliwości dokładnego zapisu niektórych liczb

- Im większa wartość, tym większy bezwzględny błąd

- Wyniki obliczeń mogą różnić się od spodziewanych na dalszych miejscach po przecinku

- Ze względu na ograniczoną pamięć komputera należało dobrać w projekcie odpowiednią wartość . Optymalną wartością okazało się być , dla wartości większych otrzymywaliśmy błędne wyniki.

1. Działanie programu

Za pomocą aplikacji webowej podajemy funkcję celu wraz z ograniczeniami oraz punkt startowy (x0, y0). Możemy również zmienić dokładność epsilon, długość kroku, modyfikować wartość kryterium stopu i określić maksymalną liczbę iteracji. Algorytm z wykorzystaniem parsera oblicza wartość funkcji f(x0,y0).

Następnie obliczamy wartość funkcji w kierunku. Po dokonanej minimalizacji obliczana jest wartość ograniczeń gi(x) dla i = 1,2,…,m, modyfikowana jest funkcja celu oraz obliczana nowa wartość c, w przypadku gdy nasze obliczone c jest mniejsze od ustalonego przez nas cmin, procedura zakańcza się (kryterium stopu dla metody Powella) i wracamy do metody Gauss’a-Seidl’a. Zmieniamy współrzędną i poruszamy się według tego samego schematu. Obliczenia kończą się w znalezionym minimum, czyli kiedy przesunięcie wzdłuż każdego z kierunków byłoby mniejsze od założonej przez nas dokładności (kryterium stopu dla metody Gauss’a-Seidl’a). Działanie programu przetestowano na funkcji celu:

Ograniczenia przyjmowały postać:

Parametry „sztywne”

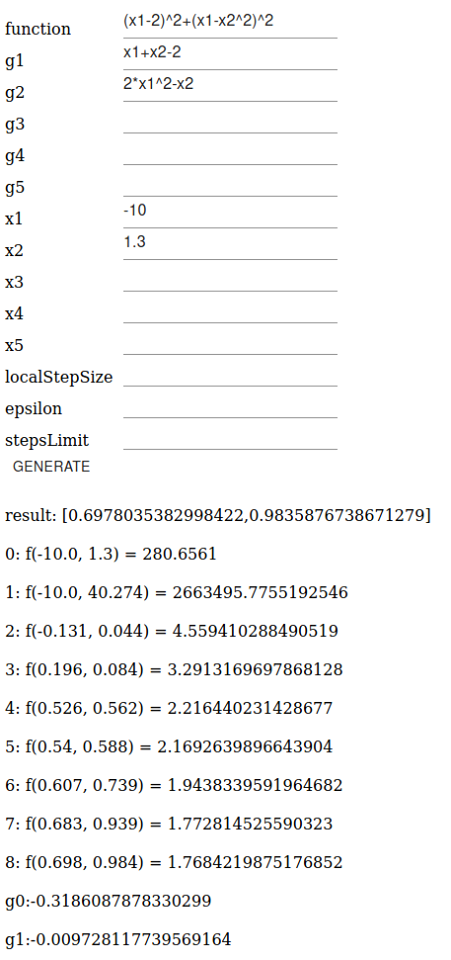
cmin =

c = 1

m1 =0.25

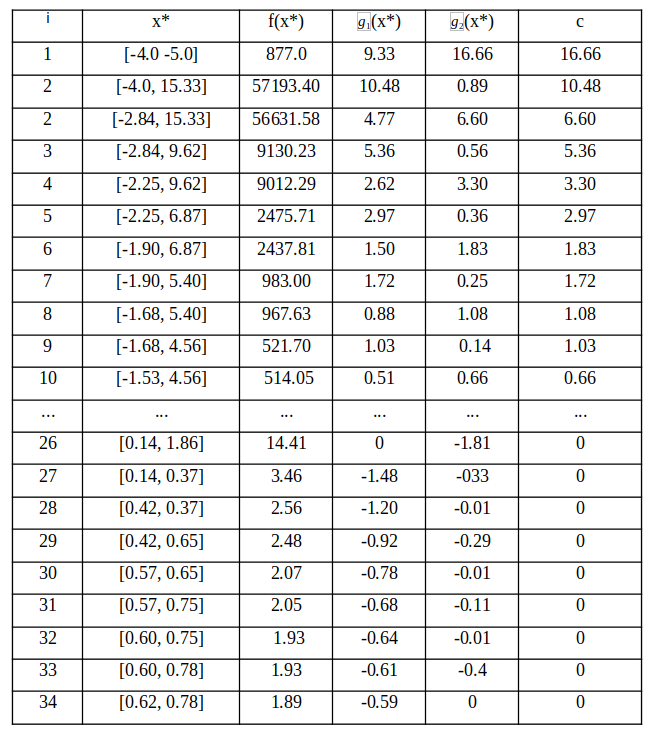
m2 = 10

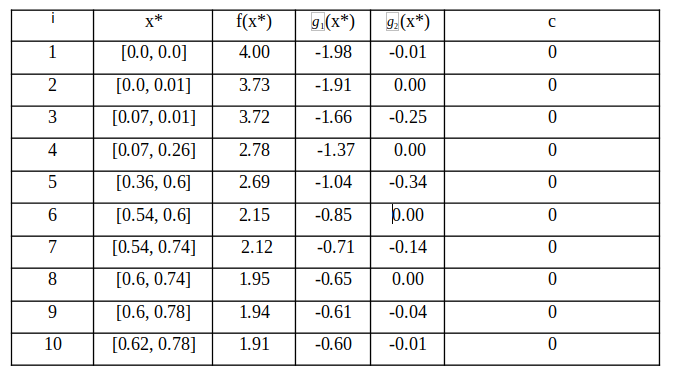
* 1. Obserwacje wyników



* 1. Badanie wpływu położenia punktu startowego wewnątrz oraz poza ograniczeniem

Punkt startowy poza ograniczeniami:



Punkt startowy wewnątrz ograniczeń:

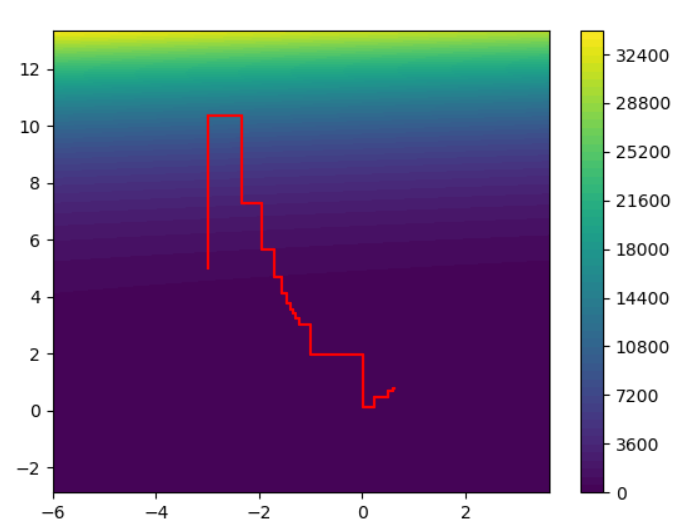
W pierwszym przypadku algorytm miał większy problem ze znalezieniem wyniku lecz w obu przypadkach algorytm poprawnie obliczył wartość minimum funkcji celu a wartości funkcji kary są do siebie zbliżone

* 1. Badanie wpływu zmiany parametrów

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | x\* | f(x\*) | (x\*) | (x\*) | c | Iteracje | ɛ |
| (-3,5) | -3.0, 5.0 | 809.0 |  |  | 7.642082108204413 |  |  |
| 1: f(-3.0, 10.358) = 12188.10393278025 |  |  |  | 6.013655949166404 |  |
| 2: f(-2.344, 10.358) = 12037.76847493273 |  |  |  | 3.7256088423949993 |  |
| 3: f(-2.344, 7.266) = 3058.413585196332 |  |  |  | 3.305133071394838 |  |
| 4: f(-1.96, 7.266) = 3013.045292919997 |  |  |  | 2.0518826916277506 |  |
| 5: f(-1.96, 5.634) = 1151.794884872223 |  |  |  | 1.916257578166622 |  |
| 6: f(-1.718, 5.634) = 1133.6591796327505 |  |  |  | 1.2050777389003349 |  |
| 7: f(-1.718, 4.699) = 580.0111235267449 |  |  |  | 1.1403348181650785 |  |
| 8: f(-1.558, 4.699) = 571.2664556343611 |  |  |  | 0.7278108270954124 |  |
| 9: f(-1.558, 4.128) = 358.6946461954214 |  |  |  | 0.6740095721098704 |  |
| 10: f(-1.454, 4.128) = 354.1138828855382 |  |  |  | 0.44634517781588556 |  |
| … | … | … | … | …  0  0 | … |
| 24: f(0.015, 1.982) = 19.250603850258184 |  |  |  | 0  00 |  |
| 25: f(0.015, 0.124) = 3.94022807122395 |  |  |  | 0 |  |
| 26: f(0.239, 0.124) = 3.1510973994609253 |  |  |  | 0 |  |
| 27: f(0.239, 0.491) = 3.1010336386557333 |  |  |  | 0 |  |
| 28: f(0.489, 0.491) = 2.3432885931571468 |  |  |  | 0 |  |
| 29: f(0.489, 0.699) = 2.2816245197807006 |  |  |  | 0 |  |
| 30: f(0.585, 0.699) = 2.011139093906944 |  |  |  | 0 |  |
| 31: f(0.585, 0.763) = 2.001762817270135 |  |  |  | 0 |  |
| 32: f(0.612, 0.763) = 1.9284992985213942 |  |  |  | 0 |  |
| 33: f(0.612, 0.784) = 1.9276412339452098 |  |  |  | 0 |  |
| 34: f(0.62, 0.784) = 1.9050262912244913 |  |  |  | 0 |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | x\* | f(x\*) | (x\*) | (x\*) | c | Iteracje | ɛ |
| (-3,5) | -3.0, 5.0 | 809.0 |  |  | 7.642082108204413 |  |  |
| 1: f(-3.0, 10.358) = 12188.10393278025 |  |  |  | 6.013655949166404 |  |
| 2: f(-2.344, 10.358) = 12037.76847493273 |  |  |  | 3.7256088423949993 |  |
| 3: f(-2.344, 7.266) = 3058.413585196332 |  |  |  | 3.305133071394838 |  |
| 4: f(-1.96, 7.266) = 3013.045292919997 |  |  |  | 2.0518826916277506 |  |
| 5: f(-1.96, 5.634) = 1151.794884872223 |  |  |  | 1.916257578166622 |  |
| 6: f(-1.718, 5.634) = 1133.6591796327505 |  |  |  | 1.2050777389003349 |  |
| 7: f(-1.718, 4.699) = 580.0111235267449 |  |  |  | 1.1403348181650785 |  |
| 8: f(-1.558, 4.699) = 571.2664556343611 |  |  |  | 0.7278108270954124 |  |
| 9: f(-1.558, 4.128) = 358.6946461954214 |  |  |  | 0.6740095721098704 |  |
| 10: f(-1.454, 4.128) = 354.1138828855382 |  |  |  | 0.44634517781588556 |  |
| … | … | … | … | …  0  0 | … |
| 24: f(0.015, 1.982) = 19.250603850258184 |  |  |  | 0  00 |  |
| 25: f(0.015, 0.124) = 3.94022807122395 |  |  |  | 0 |  |
| 26: f(0.239, 0.124) = 3.1510973994609253 |  |  |  | 0 |  |
| 27: f(0.239, 0.491) = 3.1010336386557333 |  |  |  | 0 |  |
| 28: f(0.489, 0.491) = 2.3432885931571468 |  |  |  | 0 |  |
| 29: f(0.489, 0.699) = 2.2816245197807006 |  |  |  | 0 |  |
| 30: f(0.585, 0.699) = 2.011139093906944 |  |  |  | 0 |  |
| 31: f(0.585, 0.763) = 2.001762817270135 |  |  |  | 0 |  |
| 32: f(0.612, 0.763) = 1.9284992985213942 |  |  |  | 0 |  |
| 33: f(0.612, 0.784) = 1.9276412339452098 |  |  |  | 0 |  |
| 34: f(0.62, 0.784) = 1.9050262912244913 |  |  |  | 0 |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | x\* | f(x\*) | (x\*) | (x\*) | c | Iteracje | ɛ |
| (-3,5) | -3.0, 5.0 | 809.0 |  |  | 7.642082108204413 |  |  |
| 1: f(-3.0, 10.358) = 12188.10393278025 |  |  |  | 6.013655949166404 |  |
| 2: f(-2.344, 10.358) = 12037.76847493273 |  |  |  | 3.7256088423949993 |  |
| 3: f(-2.344, 7.266) = 3058.413585196332 |  |  |  | 3.305133071394838 |  |
| 4: f(-1.96, 7.266) = 3013.045292919997 |  |  |  | 2.0518826916277506 |  |
| 5: f(-1.96, 5.634) = 1151.794884872223 |  |  |  | 1.916257578166622 |  |
| 6: f(-1.718, 5.634) = 1133.6591796327505 |  |  |  | 1.2050777389003349 |  |
| 7: f(-1.718, 4.699) = 580.0111235267449 |  |  |  | 1.1403348181650785 |  |
| 8: f(-1.558, 4.699) = 571.2664556343611 |  |  |  | 0.7278108270954124 |  |
| 9: f(-1.558, 4.128) = 358.6946461954214 |  |  |  | 0.6740095721098704 |  |
| 10: f(-1.454, 4.128) = 354.1138828855382 |  |  |  | 0.44634517781588556 |  |
| … | … | … | … | …  0  0 | … |
| 24: f(0.015, 1.982) = 19.250603850258184 |  |  |  | 0  00 |  |
| 25: f(0.015, 0.124) = 3.94022807122395 |  |  |  | 0 |  |
| 26: f(0.239, 0.124) = 3.1510973994609253 |  |  |  | 0 |  |
| 27: f(0.239, 0.491) = 3.1010336386557333 |  |  |  | 0 |  |
| 28: f(0.489, 0.491) = 2.3432885931571468 |  |  |  | 0 |  |
| 29: f(0.489, 0.699) = 2.2816245197807006 |  |  |  | 0 |  |
| 30: f(0.585, 0.699) = 2.011139093906944 |  |  |  | 0 |  |
| 31: f(0.585, 0.763) = 2.001762817270135 |  |  |  | 0 |  |
| 32: f(0.612, 0.763) = 1.9284992985213942 |  |  |  | 0 |  |
| 33: f(0.612, 0.784) = 1.9276412339452098 |  |  |  | 0 |  |
| 34: f(0.62, 0.784) = 1.9050262912244913 |  |  |  | 0 |  |



* 1. Omówienie efektywności działania

Przetestowano działanie algorytmu dla różnych punktów startowych bez zmian parametrów dodatkowych, otrzymane wyniki były zbliżone.

* 1. Badanie wpływu zmian parametrów

Wpływ wielkości epsilona: Parametr o największym wpływie na ilość iteracji. To w głównej mierze od niego zależy ilość iteracji oraz dokładność w znalezieniu ekstremum. Im mniejsza jest jego wartość, tym dokładniej znalezione jest ekstremum. Jednak zbyt mała wartość sprawia, że algorytm niepotrzebnie wykonuje bardzo dużą ilość iteracji. Jeśli wystarczy nam jedynie jakieś przybliżenie ekstremum, to powinniśmy rozważyć ograniczenie tego parametru.

Wpływ ilości iteracji: Wiedząc jaka będzie liczba iteracji, na przykład poprzez wcześniejsze ustawienie jej jako bardzo dużą wartość, można za pomocą tego parametru „uciąć” pomiary o większej dokładności niż jest nam potrzebna. W przypadku tego testu ucięto 70 ostatnich iteracji, które miały już między sobą marginalne różnice.

Wpływ punktu początkowego: Jeśli chodzi o funkcję z jednym minimum, wybór punktu początkowego nie ma wpływu na odnalezienie minimum, ponieważ z każdego miejsca funkcja powinna zbiegać do minimum.

1. Obsługa programu

Interfejs programu został napisany w języku ReactJS:

Możliwa jest zmiana parametrów oraz wprowadzenie ograniczeń w formie gi(x) 0

dla n < 3 program wyświetla wykres



1. Opis problemów

Podczas tworzenia aplikacji, jak i interfejsu napotkaliśmy kilka problemów, które opisujemy poniżej:

* Problem z utworzeniem kierunku sprzężonego. Po utworzeniu projektu wiemy, że interface’ów w bibliotece numpy bardzo upraszcza działania matematyczne i tworzenie metody złotego podziału jest bardzo szybkie.
* Problem z utworzeniem funkcji kary. Po utworzeniu projektu wiemy, że przyjmuje ona bardzo duże wartości gdy punkt nie spełnia ograniczeń.
* Problem ze stworzeniem interface’u. Po utworzeniu projektu wiemy że implementacja aplikacji webowej przez endpoint jest bardzo wygodną metodą.

Literatura

*[1] dr inż. Ewa Szlachcic materiały z wykł. "Teoria i metody optymalizacji"*

*[2] D. Kincaid, W. Cheney Analiza numeryczna, Wydawnictwa Naukowo-Techniczne, Warszawa 2006*

*[3] Stachurski A.,Wierzbicki A., „Podstawy optymalizacji”, Oficyna Wydawnicza Politechniki Warszawskiej, Warszawa 2001*

*[4] K. Gawrylczyka Prof. P.S. materiały z wykł. "Podstawy analizy i projektowania komputerowego"*