W tym zadaniu miałam za zadanie znaleźć **wszystkie** rozwiązania równania $det(A - \lambda I) = 0$ trzema metodami z dokładnością do 10^{-8}. Wybrałam trzy najbardziej znane i naprostsze metody rozwiązywania równań nieliniowych - metodę bisekcji, regula falsi i siecznych.

$$A = \begin{pmatrix} 4 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 4 \end{pmatrix}$$

Funkcja jest ciągła. Szukamy dwóch punktów, w których znak jest przeciwny - czyli $f(x_1) \cdot f(x_2)$ <0. Wówczas wiemy, że w podanym przedziale jest gdzieś miejsce zerowe. Tę informację wykorzystujemy do dwóch pierwszych metod - metody bisekcji i regula falsi.

Na początku, aby ułatwić sobie nieco rozwiązanie zadania, sprawdzam innymi metodami w jakich przedziałach mam poszukiwać rozwiązania i wyznaczyłam równanie charakterystyczne macierzy :

$$det(A - \lambda I) = -\lambda^3 + 12\lambda^2 - 46\lambda + 56$$

Przedziały które wybrałam do wszystkich zadań to: [2, 3.8] [3.5, 4.6] [5.1, 8.0]

Metoda bisekcji

Jeżeli znajdziemy już te dwa punkty, w których znak jest przeciwny, możemy zastosować przybliżenie miejsca zerowego, takie że: bierzemy środkowy punkt przedziału $[x_1, x_2]$, $x_3 = \frac{(x_1+x_2)}{2}$.Następnie ustalamy w którym z tych przedziałów ($[x_1, x_3]$czy$[x_3, x_2]$) funkcja zmienia znak - ponieważ w którymś z nich na pewno musi (chyba że dokładnie x_3 jest miejscem zerowym). Powtarzamy tę procedurę aż znajdziemy takie x_n dla którego $|f(x_n)| \leq \epsilon|$ (czyli będzie naszym miejscem zerowym z zadaną dokładnością pozukiwania rozwiązania równań).

Algorytm programu dla jednego miejsca zerowego wygląda następująco:

1. Sprawdzam czy miejsce zerowe znajduje się w $x_1$lub x_2 lub w punkcie po środku i jesli tak to je zwracam

2. Jesli nie to zamieniam punkt środkowy z jednym z punktów początkowych, tak aby wartość w obu pozostałych punktach różniła się znakiem.

3. Powtarzam krok 2 aż do momentu w którym znajdę się odpowiednio blisko miejsca zerowego i zwracam wartość tego punktu.

Ostatecznie uzyskałam nastepujące **wyniki**:

λ_1 =2.58578644

λ_2 =4.00000000

λ_3 =5.41421356

w odpowiednio 28, 26 i 29 iteracjach. Oczywiście ilość iteracji zależy głównie od wybranych przedziałów, np dla miejsca zerowego w x= 4 można uzyskać wynik juz w 1 iteracji gdy się odpowiednio dobierze przedział.

Analiza błędu i złożoność

Błąd mojej metody to 10^{-8}. Jeśli chodzi o złożoność, to analizując szybkość działania mojego programu, poprzez zmniejszenie epsilon z 10^{-8} do 10^{-16} (poprawiając dokładność o 8 rzędów)zauważyłam że długość wykonywania kodu prawie nie zwiększyła się (z 3074 ms do 3183 ms), a ilość iteracji wzrosła nieznacznie - teraz wykonało się to w 48,46 i 43 iteracjach. Wydaje się więc że złożoność czasowa wynosi O(1) i myślę, że pamięciowa to może być O(log N).

Kod programu:

Aby go uruchomić należy użyć komendy: **make all**, a następnie: **./bisekcja.x**

```c
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <unistd.h>
#include <time.h>

double funkcja(double x){
return -x*(x*(x-12)+46)+56; // -x^3+12x^2-46x+56
}



double bisekcja(double a,double b,double epsilon){
int counter=0;
sleep(1);
double poczatek =a;
double koniec=b;
if(fabs(funkcja(a)-0.0)<epsilon) return a;
if(fabs(funkcja(b)-0.0)<epsilon) return b;
double srodek=0;

while(fabs(a-b)>epsilon){
counter++;
srodek=(a+b)/2;
if(fabs(funkcja(srodek))<epsilon){
printf("Licznik iteracji dla przedzialu [%f,%f]:
%d\n",poczatek,koniec,counter);
return srodek;
}
if(funkcja(a)*funkcja(srodek)<0.0) b=srodek;
else a=srodek;
}
return -1.0;
}

int main(){
clock_t start=clock();
//double e=0.0000000000000001;
double e=0.00000001;
int k=0;
double tabX1[]={2.0,3.5,5.1};
double tabX2[]={3.8,4.6,8.0};
double rozwiazania[]={0.0,0.0,0.0};
for(k=0;k<3;++k){
rozwiazania[k]=bisekcja(tabX1[k],tabX2[k],e);
}

for(k=0;k<3;++k){
printf("%d. miejsce zerowe to : %.8f\n",k+1,rozwiazania[k]);
}
clock_t stop=clock();
```

```
stop-=start;
stop/=CLOCKS_PER_SEC/1000;

printf("Czas trwania programu: %ld ms\n",stop);

return 0;
}
```

Metoda Regula Falsi

W tej metodzie również rozpoczynamy od znalezienia odpowiednich przedziałów w których następuje zmiana znaku. Ja użyję tych samych przedziałów co poprzednio, aby zbadać różnice w ich szybkości.

Tym razem jednak, jako kolejne przybliżenia miejsca zerowego nie bierzemy już środka przedziału, ale coś co może więcej powiedzieć nam o samej funkcji - przecięcie siecznej przechodzącej przez wartości funkcji w tych punktach i osi OX. Zapisuje sie je wzorem:

$$x_3 = \frac{f(x_1)x_2 - f(x_2)x_1}{f(x_1) - f(x_2)}$$

Tak jak poprzednio, całą procedurę kończę gdy $|f(x_n)| \leq \epsilon|$. Dopóki jednak warunek nie zachodzi, wybieram ten przedział ($[x_1, x_3]$czy$[x_3, x_2]$), w którym funkcja zmienia znak.

Algorytm programu dla jednego miejsca zerowego:

1. Sprawdzam czy miejsce zerowe znajduje się w $x_1$lub x_2 lub w punkcie przecięcia siecznej i osi OX, jeśli tak to je zwracam.

2. Jesli nie to zamieniam nowy punkt z jednym z poprzednich punktów, tak aby wartość w obu pozostałych punktach różniła się znakiem.

3. Powtarzam krok 2 aż do momentu w którym znajdę się odpowiednio blisko miejsca zerowego i zwracam wartość tego punktu.

Ostatecznie uzyskałam nastepujące **wyniki**:

λ_1 =2.58578644

λ_2 =4.00000000

λ_3 =5.41421356

odpowiednio w 26, 4 i aż 93 iteracjach w czasie 3005ms. Widać że akurat w tej metodzie przedział o rozpiętości 3.1 znacząco wpłynął na ilośc iteracji. Dla poprzedniej metody nie miało to aż takiego znaczenia - możliwe więc, że w którymś z miejsc funkcja np gwałtownie malała i oddaliła nas od miejsca zerowego. Zatem czy ta metoda zbiegnie szybciej od poprzedniej czy nie, zależy w dużym stopniu od wartości początkowych.

Analiza błędów i złożoność

Błąd tej metody to 10^{-8}. Gdy błąd epsilon zmniejszyłam do 10^{-14}, to wynik uzyskałam po 42,5 i 159 iteracjach w czasie 3005 ms. Wnioskuję z tego, że złożoność programu pamięciowa będzie rzędu O(log N) a czasowa O(1). Co ciekawe, nie byłam w stanie policzyć ostatniego pierwiastka wielomianu z dokładnością 10^{-16}, możliwe że jakiś czynnik wyzerowałam lub przestał się zmieniać w tak niewielkiej odległości, np gdybym wyzerowala mianownik siecznej to już nie dojdę do rozwiązania.

Kod programu:

Aby go uruchomić należy użyć komendy: **make all**, a następnie: **./regula.x**

```c
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <unistd.h>
#include <time.h>

double funkcja(double x){
return -x*(x*(x-12)+46)+56; // -x^3+12x^2-46x+56
}

double falsi(double a,double b,double epsilon){
int counter=0;
sleep(1);
double poczatek =a;
double koniec=b;
if(fabs(funkcja(a)-0.0)<epsilon) return a;
if(fabs(funkcja(b)-0.0)<epsilon) return b;
double przeciecie=0;

while(fabs(a-b)>epsilon){
counter++;
przeciecie=(funkcja(a)*b-funkcja(b)*a)/(funkcja(a)-funkcja(b));
if(fabs(funkcja(przeciecie))<epsilon) {
printf("Licznik iteracji dla przedzialu [%f,%f]:
%d\n",poczatek,koniec,counter);
return przeciecie;
}
if(funkcja(a)*funkcja(przeciecie)<0.0) b=przeciecie;
else a=przeciecie;
}
return -1;
}

int main(){
clock_t start=clock();
//double e=0.0000000000001;
double e=0.00000001;
int k=0;
double tabX1[]={2.0,3.5,5.1};
double tabX2[]={3.0,4.6,8.0};
double rozwiazania[]={0.0,0.0,0.0};
for(k=0;k<3;++k){
rozwiazania[k]=falsi(tabX1[k],tabX2[k],e);

}
for(k=0;k<3;++k){
printf("%d. miejsce zerowe to : %.8f\n",k+1,rozwiazania[k]);
}
clock_t stop=clock();
stop-=start;
stop/=CLOCKS_PER_SEC/1000;
```

```
printf("Czas trwania programu: %ld ms\n",stop);

return 0;
}
```

Metoda siecznych

Metoda ta również korzysta z przecięcia siecznej przechodzącego przez zadane dwa punkty x_1i x_2. Tym razem, punkty te są dowolne i nie ma znaczenia czy są przeciwnych znaków. Jedyne kryterium to takie, żeby były różne od siebie.

Prowadzimy przez nie sieczną, tym samym wyznaczając x_3 jako miejsce zerowe tej siecznej. Różnica pomiędzy poprzednimi dwoma metodami jest taka, że tym razem w kolejnych krokach będziemy używać ostatnich dwóch punktów bez względu na znaki (czyli w kolejnej iteracji pozostanie x_2 i x_3).

Metoda siecznych może zbiegać szybciej od poprzednich metod, ale niestety nie zawsze prowadzi do rozwiązania - czasami nie jest w ogóle zbieżna do pierwiastka równania.

Algorytm programu dla jednego miejsca zerowego:

1. Sprawdzam czy miejsce zerowe znajduje się w $x_1$lub x_2 lub w punkcie przecięcia siecznej i osi OX, jeśli tak to je zwracam.

2. Jesli nie to zamieniam nowy punkt z pierwszym z podanch punktów.

3. Powtarzam krok 2 aż do momentu w którym znajdę się odpowiednio blisko miejsca zerowego i zwracam wartość tego punktu.

Ostatecznie uzyskałam **poniższe wyniki**. Niestety musiałam zmniejszyć pierwszy przedział z [2,3.8] do [2,3] ponieważ metoda nie zbiegała w ogóle do tego miejsca zerowego, a do bliższego krańcowi przedziału, czyli 4:

λ_1 =2.58578644

λ_2 =4.00000000

λ_3 =5.41421356

odpowiednio w 7, 1 i 7 iteracjach.

Analiza błędów i złożoność.

To rozwiązanie ma dokładność 10^{-8}.Tak jak w poprzednich przypadkach, gdy zwiększyłam dokładność z 10^{-8}do 10^{-16} to złożoność czasowa praktycznie nie zmieniła się (z 3003 do 3004 ms) a iteracje wzrosły do 9,1,13 iteracji (czyli złożoność czasowa O(1) i pamięciowa O(log N)). Warto jednak pamiętać, że metoda w ogóle nie zadziałała dla mojego przedziału, więc nie doszłaby do miejsca zerowego w każdej sytuacji.

W porównaniu jednak, gdy użyłam nowego przedziału dla regula falsi to zadziałał on odrobinę lepiej niż wcześniej (czyli 22,4 i 93 iteracje) co czyni go dla tego przypadku wolnejszym od metody siecznej.

Więc czasami metoda siecznych działa szybciej, o ile działa.

Kod programu:

Aby go uruchomić należy użyć komendy: **make all**, a następnie: **./siecznych.x**

```c
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <unistd.h>
#include <time.h>

double funkcja(double x){
return -x*(x*(x-12)+46)+56; // -x^3+12x^2-46x+56
}

double siecznej(double a,double b,double epsilon){
int counter=0;
sleep(1);
double poczatek =a;
double koniec=b;
if(fabs(funkcja(a)-0.0)<epsilon) return a;
if(fabs(funkcja(b)-0.0)<epsilon) return b;
double zerowe=0;

while(fabs(a-b)>epsilon){
counter++;
zerowe=(funkcja(a)*b-funkcja(b)*a)/(funkcja(a)-funkcja(b));
if(fabs(funkcja(zerowe))<epsilon) {
printf("Licznik iteracji dla przedzialu [%f,%f]: %d\n",poczatek,koniec,counter);
return zerowe;
}
a=b;
b=zerowe;
}
return -1;
}

int main(){
clock_t start=clock();
double e=0.00000001;
// double e=0.0000000000000001;
int k=0;
double tabX1[]={2.0,3.5,5.0};
double tabX2[]={3.0,4.5,6.0};
double rozwiazania[]={0.0,0.0,0.0};
for(k=0;k<3;++k){
rozwiazania[k]=siecznej(tabX1[k],tabX2[k],e);
}

for(k=0;k<3;++k){
printf("%d. miejsce zerowe to : %.8f\n",k+1,rozwiazania[k]);
}
clock_t stop=clock();
stop-=start;
stop/=CLOCKS_PER_SEC/1000;
```

```c
printf("Czas trwania programu: %ld ms\n",stop);

return 0;
}
```