Metody numeryczne - N10

DAMIAN PORADYŁO

N10 *Zadanie numeryczne*

Znaleźć wszystkie rozwiązania równania $\det(A-\lambda\mathbb{1})$ trzema metodami (poszukiwania miejsc zerowych) z dokładnością 10^{-8}. Które metody działają najszybciej?

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

1 OPIS ZADANIA

W zadaniu mieliśmy znaleźć wszystkie rozwiązania równania: $\det(A - \lambda I)$. Jest to nic innego jak równanie charakterystyczne macierzy. Wyliczając pierwiastki tego równania otrzymamy wartości własne naszej macierzy A.

Aby wyznaczyć nasze równanie charakterystyczne:

$$P(\lambda) = \det(A - \lambda I) = \left| \begin{bmatrix} 4 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 4 \end{bmatrix} - \lambda \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \right| \quad (1)$$

$$P(\lambda) = \left| \begin{bmatrix} 4 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 4 \end{bmatrix} - \begin{bmatrix} \lambda & 0 & 0 \\ 0 & \lambda & 0 \\ 0 & 0 & \lambda \end{bmatrix} \right| \quad (2)$$

$$P(\lambda) = \begin{vmatrix} 4-\lambda & -1 & 0 \\ -1 & 4-\lambda & -1 \\ 0 & -1 & 4-\lambda \end{vmatrix} \quad (3)$$

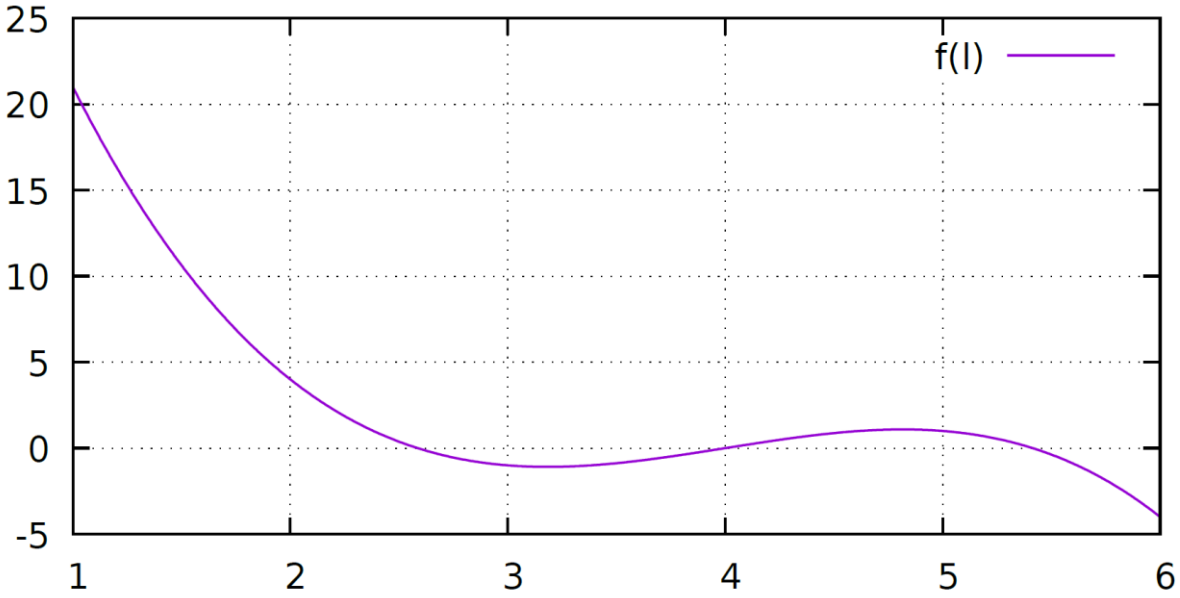
$$P(\lambda) = (4-\lambda) \begin{vmatrix} 4-\lambda & -1 \\ -1 & 4-\lambda \end{vmatrix} + 1 \begin{vmatrix} -1 & -1 \\ 0 & 4-\lambda \end{vmatrix} - 0 \begin{vmatrix} -1 & 4-\lambda \\ 0 & -1 \end{vmatrix} \quad (4)$$

$$P(\lambda) = (4-\lambda)((4-\lambda)^2 - 1) + (-4-\lambda) + 0 \quad (5)$$

Co ostatecznie daje nam:

$$P(\lambda) = -\lambda^3 + 12\lambda^2 - 46\lambda + 56 \quad (6)$$

Posłużyłem się programem ***gnuplot*** w celu zwizualizowania sobie, wyglądu naszej funkcji, a także zakresu poszukiwania rozwiązań. Otrzymałem następujący wykres:



Jak widzimy, nasze równanie posiada 3 rozwiązania w dziedzinie liczb rzeczywistych w zakresie $(2; 6)$

1.1 METODA NEWTONA

Metoda Newtona podczas poszukiwania pierwiastka równania f(x) = 0 wymaga od nas znajomości pierwszej pochodnej tejże funkcji. Jeżeli założymy, że punkt x_0 leży blisko pierwiastka to możemy rozwinąć funkcję w szereg Taylora w punkcje x_0

$$f(x) = f(x_0) + (x - x_0)f'(x_0) + \frac{(x - x_0)^2}{2!}f''(x_0) + \ldots$$

Jeśli uwzględnimy fakt, że **x** musi być pierwiastkiem równania f(x) przyrównanego do zera i z rozwinięcia Taylora weźmiemy tylko dwa pierwsze czynniki to:

$$0 = f(x_0) + (x - x_0)f'(x_0)$$

Równanie wówczas ze względu na x będzie postaci:

$$x - x_0 = \delta = -\frac{f(x_0)}{f'(x_0)}$$

Teraz x reprezentuje poprawione oszacowanie pierwiastka równania.

Ogólna formuła we wzorze Newtona może wyglądać:

$$x_i - x_{i-1} = \delta_i = -\frac{f(x_{i-1})}{f'(x_{i-1})}, \qquad i = 1,2,\ldots,n$$

1.2 METODA SIECZNYCH

Metoda siecznych jest ulepszeniem metody regula falsi. W metodzie regula falsi żądamy, aby funkcja f(x) zawsze przyjmowała różne znaki na krańcach przedziału poszukiwań pierwiastka.

Różne znaki gwarantują nam istnienie pierwiastka w tym przedziale. Otóż w metodzie siecznych taki wymóg nie jest konieczny. Do wyznaczenia kolejnego przybliżenia pierwiastka funkcji wykorzystujemy dwa poprzednio wyznaczone punkty

Po wyborze przybliżeń początkowych x_0 oraz x_1 tworzony jest rekurencyjny ciąg x_2, x_3,…. Na podstawie wzoru:

$$\begin{cases} x_0 = a, \\ x_1 = b, \\ x_i = x_{i-1} - f(x_{i-1})\dfrac{x_{i-1} - x_{x-2}}{f(x_{i-1}) - f(x_{i-2})}, i = 2, 3, \ldots, n. \end{cases}$$

W ostateczności kolejne przybliżenia mają postać:

$$x_2 = b - f(b)\frac{b-a}{f(b)-f(a)}$$
$$x_3 = x_2 - f(x_2)\frac{x_2 - b}{f(x_2) - f(b)}$$
$$\vdots$$
$$x_i = x_{i-1} - f(x_{i-1})\frac{x_{i-1} - x_{x-2}}{f(x_{i-1}) - f(x_{i-2})},$$

1.3 METODA REGULA FALSI

Metoda *Regula falsi* jako wartość początkową przyjmuje:

$$x_0 = \frac{af(b) - bf(a)}{f(b) - f(a)}$$

Musimy założyć, że w naszym przedziale $< a; b >$ znajduje się dokładnie jeden pierwiastek, a także, że na końcach przedziałów nasza funkcja ma różne znaki.

Jeżeli, chodzi o kolejne przybliżenia naszych pierwiastków:

$$x_{i+1} = \begin{cases} \dfrac{x_i f(a) - af(x_i)}{f(a) - f(x_i)}, & gdy\ f(a)f(x_i) \geq 0 \\ \dfrac{x_i f(b) - bf(x_i)}{f(b) - f(x_i)}, & gdy\ f(b)f(x_i) < 0 \end{cases}$$

2 WYNIKI

```
Regula Falsi
2.5857864376
4.0000000000
5.4142135624
Liczba iteracji: 6

Metoda Newtona
2.5857864376
4.0000000000
5.4142135624
Liczba iteracji: 18

Metoda siecznych:
2.5857864376
4.0000000000
5.4142135624
Liczba iteracji: 10
```

Liczba iteracji (nie licząc kolejnych zagęszczeń przedziałów) jest najmniejsza w metodzie regula falsi, największa ilość iteracji jest wymagana podczas metody Newtona.

Wynik, generalnie zgadzają się z wynikami uzyskanymi poprzez wyliczenie wartości własnych naszej macierzy A:

```
1   import numpy as np
2   from scipy import linalg
3
4   a = np.empty(2)
5   b = np.empty(3)
6
7   a.fill(-1.0)
8   b.fill(4.0)
9
10
11  u, v = linalg.eigh_tridiagonal(b, a)
12
13  print(u)
14

n10 ×
C:\Users\Damiano\AppData\Local\Programs\Python\
[2.58578644 4.         5.41421356]
```

3 KOD PROGRAMU

```cpp
#include <iostream>
#include <iomanip>
#include <cmath>

#define E 1e-10
int i1 = 0;
int i2 = 0;
int i3 = 0;

double f(double l) {
    return -(l*l*l) + (12 * (l * l)) - (46 * l) + 56;
}

double df(double l) {
    return -3*(l*l) + (24*l) - 46;
}

double secantMethod(double (*f)(double), double a, double b) {
    double x0 = 0.0;

    while(true) {
        double f_a = f(a);
        double f_b = f(b);
        if(std::fabs(f_b - f_a) < E) break;
        x0 = b - f_b * (b - a) / (f_b - f_a);
        a = b;
        b = x0;
        i1++;
    }
    return x0;
}

double methodNewton(double (*f)(double), double start) {
    double x0 = start;
    double f_x0 = f(x0);

    while(std::fabs(f_x0) > E) {
        double d_f = df(x0);
        x0 -= f_x0 / d_f;
        f_x0 = f(x0);
        i2++;
    }
    return x0;
}

double regulaFalsi(double (*f)(double), double a, double b) {
    double fa = f(a), fb = f(b), x1 = a, x0 = b, f0;
    while(fabs(x1 - x0) > E) {
        x1 = x0;
        x0 = a - fa * (b - a) / (fb - fa);
        f0 = f(x0);
        if(fabs(f0) < E) break;
        if(fa * f0 < 0){
            b = x0;
            fb = f0;
        }
        else {
            a = x0;
            fa = f0;
        }
        i3++;
    }
    return x0;
}
int main(int argc, char const *argv[]) {

    double x0, x1;

    printf("Regula Falsi\n");
    for(double i=2; i<6; i+=0.01) {
        x0 = i;
        x1 = 0.01 + x0;
        if(f(x0) * f(x1) <= 0) {
            std::cout << std::fixed << std::setprecision(10) << regulaFalsi(f, x0, x1) << std::endl;
        }
    }
    std::cout << "Liczba iteracji: " << i3 << std::endl  << std::endl;

    printf("Metoda Newtona\n");
    for(double i=2; i<6; i+=0.01) {
        x0 = i;
        x1 = x0 + 0.01;
        if(f(x0) * f(x1) <= 0) {
            if(methodNewton(f, x0) >= x0 && methodNewton(f, x0) <= x1) {
                std::cout << std::fixed << std::setprecision(10) << methodNewton(f, x0) << std::endl;
            }
        }
    }
    std::cout << "Liczba iteracji: " << i2;

    std::cout << "\n\n";
    printf("Metoda siecznych:\n");
    for (double i=2; i<6; i+=0.01) {
        x0 = i;
        x1 = x0 + 0.01;
        if(f(x0) * f(x1) <= 0 ) {
            std::cout << std::fixed << std::setprecision(10) << secantMethod(f, x0, x1) << std::endl;
        }
    }

    std::cout << "Liczba iteracji: " << i1 << std::endl;
}
```

4 URUCHOMIENIE

>> make run

W skład zestawu wchodzą:

- opracowanie
- makefile
- kod programu