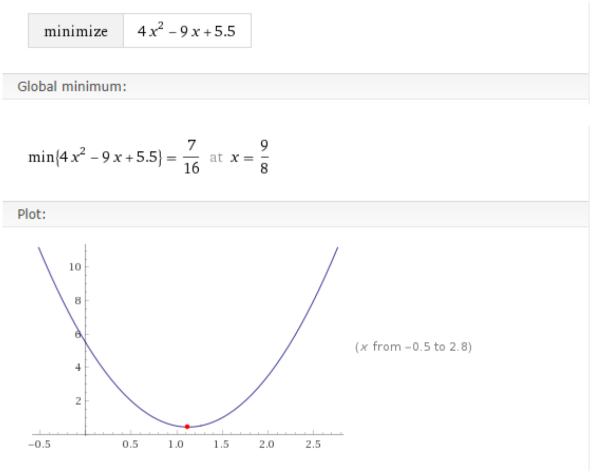
<table>
<tr><td colspan="2">Laboratorium Metod Obliczeniowych
Wydział Elektrotechniki Automatyki i Informatyki
Politechnika Świętokrzyska</td></tr>
<tr><td>Studia: Stacjonarne I stopnia</td><td>Kierunek: Informatyka</td></tr>
<tr><td>Data wykonania: 18.12.2017</td><td>Grupa: 3ID13B</td></tr>
<tr><td colspan="2">Imię i nazwisko: Bartłomiej Osak</td></tr>
<tr><td>Numer ćwiczenia:
10</td><td>Temat ćwiczenia:
Optymalizacja jednowymiarowa</td></tr>
</table>

1. Ustalenie minimum lokalnego funkcji dla zadanego przedziału.



Zgodnie z wykresem funkcji – dla zadanego przedziału minimum lokalne wynosi 1.125.

2. Optymalizacja jednowymiarowa – wyszukiwanie minimum lokalnego funkcji za pomocą *algorytmu złotego podziału* – program w środowisku MATLAB.

Polecenie: Za pomocą metody złotego podziału wyznacz minimum lokalne funkcji:

$$f(x) = 4x^2 - 9x + 5.5$$

w przedziale $x \in [0.5; 2]$.

Kod źródłowy:

```
function[c] = metoda_zlotego_podzialu(a,b,e)
    f = inline(input('Podaj rownanie funkcji f(x): ','s'));
    k = (sqrt(5)-1)/2;
    xl = b-k*(b-a);
    xp = a+k*(b-a);
    while 1
        if f(xl) < f(xp)
            b = xp;
            xp = xl;
            xl = b-k*(b-a);
        end
        if f(xl) > f(xp)
            a = xl;
            xl = xp;
            xp = a+k*(b-a);
        end
        if abs(a-b) < e
            c = (a+b)/2;
            break;
        end
    end
end
```

Wywołanie:

- **Uruchomienie funkcji metoda_zlotego_podzialu z parametrami: dolna i górna granica przedziału argumentu x oraz wartość epsilon (dokładność):**

  ```
  metoda_zlotego_podzialu(0.5,2,0.001)
  ```

- **Wynik zadziałania:**

  ```
  Podaj rownanie funkcji f(x): 4*x^2-9*x+5.5
  ans = 1.125
  ```

Zrzut ekranu z działania programu:

```
Command Window
>> metoda_zlotego_podzialu(0.5,2,0.001)
Podaj rownanie funkcji f(x): 4*x^2-9*x+5.5

ans =

    1.1251
```

Opis programu:

Funkcja metoda_zlotego_podzialu przyjmuje 3 operandy wywołania. Są to:

- a – dolna granica przedziału dla argumentów x
- b – górna granica przedziału dla argumentów x
- e – wartość epsilon, która określa dokładność dla algorytmu

Na początku do zmiennej f przypisywana jest funkcja, która wprowadza użytkownik. Następnie do zmiennej stałej k przypisywana jest stała wartość wynosząca:

$$k = \frac{\sqrt{5} - 1}{2}$$

Wartość ta wynika ze stosunku złotego podziału. Następnie do zmiennych xl oraz xk przypisywane są wartości zgodne z zapisem:

$$xl = b - k * (b - a)$$
$$xp = a + k * (b - a)$$

Następnie rozpoczynamy iterowanie do momentu aż przedział $[a, b]$ nie osiągnie dostatecznie małego rozmiaru. W pętli tej wykonujemy:

$$Jeżeli: f(xl) < f(xp)\ to:$$
$$b = xp$$
$$xp = xl$$
$$xl = b - k * (b - a)$$

$$Jeżeli: f(xl) > f(xp)\ to:$$
$$a = xl$$
$$xl = xp$$
$$xp = a + k * (b - a)$$

Powyższe zapisy oznaczają złoty podział odcinka na dwa takie odcinki, których stosunek długości dłuższego z nich do długości krótszego jest równy stosunkowi długości dzielonego odcinka do długości dłuższego odcinka. Iteracja zakończy się, jeżeli:

$$|a - b| < e$$

czyli, gdy wartość bezwzględna różnicy dolnej i górnej granicy przedziału argumentów x będzie mniejsza od wartości epsilon. Jeżeli warunek jest spełniony to algorytm kończy działanie i jest zwracany wynik operacji:

$$x^* = c = \frac{a + b}{2}$$

Jest to nasz ostateczny wynik dla zadanych danych wejściowych.

Analiza uzyskanego wyniku:

Uzyskane wyniki dla różnych wartości epsilon umieszczono w tabeli. Wartość dokładna to: 1.125.

ε	Uzyskany wynik:	δ (błąd względny)
0.1	1.1307	16.17%
0.01	1.1261	0.097%
0.001	1.1251	0.008%
0.0001	1.1249	0.00008%
...	...	...
0.0000001	**1.125**	**0%**

Dopiero dla bardzo małych wartości epsilon uzyskujemy dokładny wynik.

3. **Optymalizacja jednowymiarowa – wyszukiwanie minimum lokalnego funkcji za pomocą *algorytmu złotego podziału* – program w języku JAVA.**

Polecenie: Za pomocą metody złotego podziału wyznacz minimum lokalne funkcji:

$$f(x) = 4x^2 - 9x + 5.5$$

w przedziale $x \in [0.5; 2]$.

Kod źródłowy:

```
import java.util.Scanner;

public class OptimizationMidPoint {

    private static double e;
    private static final double K;

    static {
        K = (Math.sqrt(5) - 1) / 2;
    }

    private static double a;
    private static double b;

    public static double f(double x) {
        return 4 * Math.pow(x, 2) - 9 * x + 5.5;
    }

    public static void enterData() {
        boolean end = false;
        while (!end) {
            try {
                System.out.println("\n### Program minimum lokalne funkcji ###");
                System.out.println("### Dla: f(x)=4x^2-9x+5.5 ###");
                System.out.println("\n");
                System.out.print(">> Wprowadź dolną granicę przedziału zbioru  argumentów x
                ([x0,xn] -> x0):");
                a = new Scanner(System.in).nextDouble();
                System.out.print(">> Wprowadź górną granicę przedziału zbioru argumentów x
                ([x0,xn] -> xn):");
                b = new Scanner(System.in).nextDouble();
                System.out.print(">> Wprowadź warunek dokładności - epsilon (e):");
                e = new Scanner(System.in).nextDouble();
                end = true;


            } catch (Exception e) {
                System.out.println("\n[ERROR] Wystąpił błąd wpisywania danych!");
                for (int i = 0; i < 500; i++) {
                    System.out.println();
                }
            }
        }
    }

    public static void midPointAlgorithm() {

        double xl = b - K * (b - a);
        double xp = a + K * (b - a);

        for (; ; ) {
            if (f(xl) < f(xp)) {
                b = xp;
```

```
                xp = xl;
                xl = b - K * (b - a);
            }
            if (f(xl) > f(xp)) {
                a = xl;
                xl = xp;
                xp = a + K * (b - a);
            }
            if (Math.abs(a - b) < e) {
                double result = (a + b) / 2;
                System.out.println("[WYNIK - MIN LOKALNE f(x)]: " + result);
                break;
            }
        }
    }

    public static void main(String[] args) {
        OptimizationMidPoint.enterData();
        OptimizationMidPoint.midPointAlgorithm();
    }
}
```

Zrzut ekranu z działania programu:

```
### Program minimum lokalne funkcji ###
### Dla: f(x)=4x^2-9x+5.5 ###


>> Wprowadź dolną granicę przedziału zbioru argumentów x ([x0,xn] -> x0):0,5
>> Wprowadź górną granicę przedziału zbioru argumentów x ([x0,xn] -> xn):2
>> Wprowadź warunek dokładności - epsilon (e):0,001
[WYNIK - MIN LOKALNE f(x)]: 1.12508142809223
```

Opis programu:

Program składa się z jednej klasy publicznej o nazwie OptimizationMidPoint. Klasa ta posiada 4 pola statyczne:

- double e – pole przechowujące wartość epsilon wprowadzaną przez użytkownika,
- final double K – pole finalne przechowują wartość stałą K wynikającą ze złotego podziału odcinka. Wynosi ona wartość:

$$k = \frac{\sqrt{5} - 1}{2}$$

- double a – pole przechowujące wartość dolnej granicy przedziału dla argumentów x wprowadzaną przez użytkownika,
- double b – pole przechowujące wartość górnej granicy przedziału dla argumentów x wprowadzaną przez użytkownika.

Ponadto klasa zawiera 3 metody statyczne nie wliczając metody inicjalizacyjnej main. Są to:

- double f (double x) – metoda przechowująca funkcję, dla której obliczamy wartość minimum w zadanym przedziale,
- void enterData() – metoda odpowiedzialna za wprowadzanie wartości a,b oraz e przez użytkownika. W przypadku błędu wpisania wypisywany jest stosowny komunikat i konsola jest czyszczona.
- void midPointAlgorithm() – metoda odpowiedzialna za obliczanie wartości minimum zgodnie z algorytmem. Na początku do zmiennej xl oraz xk przypisywane są wartości:

$$xl = b - k * (b - a) \; oraz \; xp = a + k * (b - a)$$

Następnie rozpoczynamy iterowanie do momentu aż przedział $[a, b]$ nie osiągnie dostatecznie małego rozmiaru. W pętli tej wykonujemy:

$$Jeżeli\text{: } f(xl) < f(xp) \; to\text{:}$$
$$b = xp$$

$$xp = xl$$
$$xl = b - k * (b - a)$$

$$Jeżeli: f(xl) > f(xp)\ to:$$
$$a = xl$$
$$xl = xp$$
$$xp = a + k * (b - a)$$

Powyższe zapisy oznaczają złoty podział odcinka na dwa takie odcinki, których stosunek długości dłuższego z nich do długości krótszego jest równy stosunkowi długości dzielonego odcinka do długości dłuższego odcinka. Iteracja zakończy się, jeżeli:

$$|a - b| < e$$

czyli, gdy wartość bezwzględna różnicy dolnej i górnej granicy przedziału argumentów x będzie mniejsza od wartości epsilon. Jeżeli warunek jest spełniony to algorytm kończy działanie i jest zwracany wynik operacji:

$$x^* = c = \frac{a + b}{2}$$

Jest to nasz ostateczny wynik dla zadanych danych wejściowych.

4. Optymalizacja jednowymiarowa – wyszukiwanie minimum lokalnego funkcji za pomocą *algorytmu opartym na interpolacji Lagrange'a* – program w języku JAVA.

Polecenie: Za pomocą metody złotego podziału wyznacz minimum lokalne funkcji:

$$f(x) = 4x^2 - 9x + 5.5$$

w przedziale $x \in [0.5; 2]$.

Kod źródłowy:

```
import java.util.Scanner;

public class OptimizationLagrange {


    private static double e;
    private static double g;
    private static int n;
    private static double[] a;
    private static double[] b;
    private static double[] c;
    private static double[] d;


    public static double f(double x) {
        return 4 * Math.pow(x, 2) - 9 * x + 5.5;
    }

    public static void enterData() {
        boolean end = false;
        while (!end) {
            try {
                System.out.println("\n### Program minimum lokalne funkcji ###");
                System.out.println("### Dla: f(x)=4x^2-9x+5.5 ###");
                System.out.println("\n");
                System.out.print(">> Wprowadź ilość iteracji (Nmax):");
                n = new Scanner(System.in).nextInt();
                a = new double[n];
                b = new double[n];
                c = new double[n];
```

```
            d = new double[n];
            System.out.print(">> Wprowadź dolną granicę przedziału zbioru argumentów x
            ([x0,xn] -> x0):");
            a[0] = new Scanner(System.in).nextDouble();
            System.out.print(">> Wprowadź górną granicę przedziału zbioru argumentów x
            ([x0,xn] -> xn):");
            b[0] = new Scanner(System.in).nextDouble();
            c[0] = (a[0] + b[0]) / 2;
            System.out.print(">> Wprowadź warunek dokładności - epsilon (e):");
            e = new Scanner(System.in).nextDouble();
            g = 0.001 * e;
            end = true;
        } catch (Exception e) {
            System.out.println("\n[ERROR] Wystapił błąd wpisywania danych!");
            for (int i = 0; i < 500; i++) {
                System.out.println();
            }
        }
    }
}

public static void optimizationMethodLagrange() throws Exception {

    for (int i = 0; i < n; i++) {

        d[i] = 0.5d * ((f(a[i]) * (Math.pow(c[i], 2) - Math.pow(b[i], 2)) + f(c[i]) *
        (Math.pow(b[i], 2) - Math.pow(a[i], 2)) + f(b[i]) * (Math.pow(a[i], 2) -
        Math.pow(c[i], 2)))
        / (f(a[i]) * (c[i] - b[i]) + f(c[i]) * (b[i] - a[i]) + f(b[i]) * (a[i] - c[i])));
        if (b[i] - a[i] < e) {
            System.out.println("[WYNIK - MIN LOKALNE f(x)]: " + d[i]);
            System.exit(0);
        }

        if (i > 0) {
            if (b[i] - a[i] < e || Math.abs(d[i] - d[i - 1]) <= g) {
                System.out.println("[WYNIK - MIN LOKALNE f(x)]: " + d[i]);
                System.exit(0);
            }
        }

        if (a[i] < d[i] && d[i] < c[i] && a[i] < c[i] && i < n - 1) {
            if (f(d[i]) < f(c[i])) {
                a[i + 1] = a[i];
                c[i + 1] = d[i];
                b[i + 1] = c[i];
            } else {
                a[i + 1] = d[i];
                c[i + 1] = c[i];
                b[i + 1] = b[i];
            }
        } else {
            if (c[i] < d[i] && d[i] < b[i] && c[i] < b[i] && i < n - 1) {
                if (f(d[i]) < f(c[i])) {
                    a[i + 1] = c[i];
                    c[i + 1] = d[i];
                    b[i + 1] = b[i];
                } else {
                    a[i + 1] = a[i];
                    c[i + 1] = c[i];
                    b[i + 1] = d[i];
                }
            } else {
                System.out.println("[ERROR] Algorytm nie jest zbieżny!");
```

```
                System.exit(0);
            }
        }
    }}
    public static void main(String[] args) {
        OptimizationLagrange.enterData();
        try {
            OptimizationLagrange.optimizationMethodLagrange();
        } catch (Exception e1) {
            System.out.println("[ERROR] Wystąpił błąd obliczeniowy zadania!");
            OptimizationLagrange.enterData();
        }
    }
}
```

Zrzut ekranu z działania programu:

```
### Program minimum lokalne funkcji ###
### Dla: f(x)=4x^2-9x+5.5 ###

>> Wprowadź ilość iteracji (Nmax):10
>> Wprowadź dolną granicę przedziału zbioru argumentów x ([x0,xn] -> x0):0,5
>> Wprowadź górną granicę przedziału zbioru argumentów x ([x0,xn] -> xn):2
>> Wprowadź warunek dokładności - epsilon (e):0,001
[WYNIK - MIN LOKALNE f(x)]: 1.125
```

Opis programu:

Program składa się z jednej klasy publicznej o nazwie OptimizationLagrange. Zawiera ona 7 pól statycznych:

- double e – pole przechowujące wartość dokładności (epsilon), która jest wprowadzana przez użytkownika
- double g – pole przechowujące wartość współczynnika, który jest znacząco mniejszy od wartości epsilon. Potrzebny do przeprowadzenia testu stacjonarności.
- int n – pole przechowujące ilość iteracji algorytmu – wartość wprowadzana przez użytkownika.
- double[ ] a – tablica przechowująca początkowo dolną granicę przedziału, a następnie, w następnych komórkach przechowuje kolejne wartości dolnych granic nowych, zawężonych przedziałów.
- double[ ] b – tablica przechowująca początkowo górną granicę przedziału, a następnie, w następnych komórkach przechowuje kolejne wartości górnych granic nowych, zawężonych przedziałów.
- double[ ] c – tablica przechowująca początkowo punkt wewnętrzny przedziału (taki, że: $a < c < b$), a następnie, w następnych komórkach przechowuje kolejne wartości punktów wewnętrznych w nowych, zawężonych przedziałach.
- double[ ] d – tablica przechowująca wartości wierzchołka paraboli określonej wielomianem Lagrange.

Ponadto klasa zawiera 3 metody statyczne nie wliczając metody inicjalizacyjnej main. Są to:

- double f (double x) – metoda przechowująca funkcję, dla której obliczamy wartość minimum w zadanym przedziale,
- void enterData() – metoda odpowiedzialna za wprowadzanie wartości n, a, b oraz e przez użytkownika. W przypadku błędu wpisania wypisywany jest stosowny komunikat i konsola jest czyszczona
- void optimizationMethodLagrange() – metoda odpowiedzialna za wyszukiwanie minimum za pomocą wielomianu Lagrange. Na początku obliczamy minimum paraboli przechodzącej przez punkty a,b,c, czyli wierzchołek paraboli zgodnie ze wzorem:

$$d = \frac{1}{2}\frac{f(a)(c^2 - b^2) + f(c)(b^2 - a^2) + f(b)(a^2 - c^2)}{f(a)(c - b) + f(c)(b - a) + f(b)(a - c)}$$

  Minimum istnieje tylko wtedy, gdy mianownik równania jest dodatni. Gdy obliczony punkt d znajduje się w zbiorze $(a, c) \cup (c, b)$ to w następnej iteracji przedział poszukiwań jest modyfikowany i cała procedura jest powtarzana aż do osiągnięcia zadanej dokładności. Modyfikacja przedziału poszukiwań prowadzona jest według następującej zasady:

$$\left.\begin{aligned} a^{(i+1)} &= a^{(i)} \\ c^{(i+1)} &= d^{(i)} \\ b^{(i+1)} &= c^{(i)} \end{aligned}\right\} jeżeli\ a^{(i)} < d^{(i)} < c^{(i)}\ oraz\ f\left(d^{(i)}\right) < f\left(c^{(i)}\right)$$

$$\left.\begin{aligned} a^{(i+1)} &= d^{(i)} \\ c^{(i+1)} &= c^{(i)} \\ b^{(i+1)} &= b^{(i)} \end{aligned}\right\} jeżeli\ a^{(i)} < d^{(i)} < c^{(i)}\ oraz\ f\left(d^{(i)}\right) \geq f\left(c^{(i)}\right)$$

$$\left.\begin{aligned} a^{(i+1)} &= c^{(i)} \\ c^{(i+1)} &= d^{(i)} \\ b^{(i+1)} &= b^{(i)} \end{aligned}\right\} jeżeli\ c^{(i)} < d^{(i)} < b^{(i)}\ oraz\ f\left(d^{(i)}\right) < f\left(c^{(i)}\right)$$

$$\left.\begin{aligned} a^{(i+1)} &= a^{(i)} \\ c^{(i+1)} &= c^{(i)} \\ b^{(i+1)} &= d^{(i)} \end{aligned}\right\} jeżeli\ c^{(i)} < d^{(i)} < b^{(i)}\ oraz\ f\left(d^{(i)}\right) \geq f\left(c^{(i)}\right)$$

Jeżeli długość przedziału $\left[a^{(i)}, b^{(i)}\right]$ maleje w kolejnych iteracjach to metoda jest zbieżna do minimum lokalnego funkcji w przedziale $\left[a^{(0)}, b^{(0)}\right]$. Jeżeli metoda nie jest zbieżna to pętla programu jest przerywana i jest wypisywany komunikat. Ponadto algorytm zostanie przerwany, gdy:

$$i > N(n)$$

Algorytm kończy działanie, gdy:

$$b^{(i)} - a^{(i)} < \varepsilon(e) \quad lub \quad \left|d^{(i)} - d^{(i-1)}\right| \leq \gamma(g)$$

Gdy algorytm zakończy działanie to uzyskanym wynikiem jest:

$$x^{*} = d^{(i)} \rightarrow przybliżone\ rozwiązanie$$

Dla podanego przykładu algorytm bezproblemowo obliczył minimum funkcji w zadanym przedziale, ponieważ funkcja f ma postać funkcji kwadratowej. W przypadku funkcji unimodalnej, której wykres nie jest zbliżony do funkcji kwadratowej algorytm zwróci błąd o niezbieżności metody. Z tego względu jest to algorytm, którego zastosowania są bardzo mocno ograniczone.