Laboratorium **Metod Obliczeniowych** Wydział Elektrotechniki Automatyki i Informatyki Politechnika Świętokrzyska	
Studia: **Stacjonarne I stopnia**	Kierunek: **Informatyka**
Data wykonania: **30.10.2017**	Grupa: **3ID13B**
Imię i nazwisko: **Bartłomiej Osak**	
Numer ćwiczenia: **3**	Temat ćwiczenia: **Całkowanie numeryczne**

1. Program w środowisku MATLAB – metoda trapezów.

Polecenie:
Wyznacz metodą trapezów wartość:

$$s = \int_0^{\frac{\pi}{2}} f(x)dx \; dla \; f(x) = \sqrt{1 + 0.6 * \sin(x)}$$

Kod źródłowy:

```
function[y] = trapez_calka(a,b,n,f)
format long g
    if (n<2)
        disp('Liczba podprzedzialow musi byc >=2')
        return
    end
    h=(b-a)/n;
    y=0;
    for i = 1 : n-1
            y=y+feval(f,a+i*h);
    end
    y=h*(y+(feval(f,a)+feval(f,b))/2);
end
```

Wywołanie:

1. Przypisanie do zmiennej f funkcji inline danej w zadaniu.
   ```
   f=inline('sqrt(1+0.6*sin(x))')
   ```
   Zwrócono:
   ```
   f = Inline function:
        f(x) = sqrt(1+0.6*sin(x))
   ```
2. Wywołanie funkcji trapez_calka:
   ```
   trapez_calka(0,pi/2,10000,f)
   ```
3. Zwrócony wynik:
   ```
   ans = 1.84227758210136
   ```

Opis funkcji:

Funkcja trapez_calka przyjmuje cztery argumenty wywołania. Są to kolejno:

- a – dolna granica całkowania,
- b – górna granica całkowania,
- n – ilość podprzedziałów,
- f – nazwa funkcji, która zostaje wywoływana w trakcie pracy funkcji trapez_calka.

Zgodnie ze wzorem $h = (b - a)/n$ obliczana jest wartość h, czyli krok całkowania. Następnie w pętli for, czyli sposobem iteracyjnym dodawane są do siebie wartość wyniku poprzedniej iteracji z wywołaniem funkcji f dla argumentu $a + i * h$. Działanie dotyczy tylko wartości $y_1, \dots, y_{n-1}$. Po zakończeniu iteracji do zmiennej y, która przechowuje wynik dopisywany jest wynik z operacji mnożenia przez h (krok całkowania) sumy wywołania funkcji f dla dolnej i górnej granicy całkowania podzielonej przez 2 zgodnie ze wzorem:

$$h\left[\frac{y_0 + y_n}{2} + \sum_{i=1}^{n-1} y_i]\right]$$

Po obliczeniu zwracany jest końcowy wynik. Ponadto funkcja jest zabezpieczona przed podaniem błędnej liczby podprzedziałów. W razie wystąpienia błędnej wartości n wypisywany jest dla użytkownika stosowny komunikat.

2. Program w środowisku MATLAB – metoda Simpsona.

Polecenie:
Wyznacz metodą Simpsona wartość:

$$s = \int_0^{\frac{\pi}{2}} f(x)dx \; dla \; f(x) = \sqrt{1 + 0.6 * \sin(x)}$$

Kod źródłowy:

```
function[y] = simpson_calka(a,b,n,f)
format long g
    if (n<2) || mod(n,2)
        disp('Liczba podprzedzialow musi byc parzysta oraz >=2')
        return
    end
    h=(b-a)/n;
    y = 0;
    for i = 1 : n-1
        if mod(i,2)
            y=y+4*feval(f,a+i*h);
        else
            y=y+2*feval(f,a+i*h);
        end
    end
    y=(h*(y+feval(f,a) +  feval(f,b)))/3;
end
```

Wywołanie:

1. Przypisanie do zmiennej f funkcji inline danej w zadaniu.
   ```
   f=inline('sqrt(1+0.6*sin(x))')
   ```
   Zwrócono:
   ```
   f = Inline function:
        f(x) = sqrt(1+0.6*sin(x))
   ```
2. Wywołanie funkcji simpson_calka:
   ```
   simpson_calka(0,pi/2,10000,f)
   ```
3. Zwrócony wynik:
   ```
   ans = 1.84227758271821
   ```

Opis funkcji:
Funkcja simpson_calka przyjmuje cztery argumenty wywołania. Są to kolejno:

- a – dolna granica całkowania,
- b – górna granica całkowania,
- n – ilość podprzedziałów,
- f – nazwa funkcji, która zostaje wywoływana w trakcie pracy funkcji simpson_calka.

Zgodnie ze wzorem $h = (b - a)/n$ obliczana jest wartość h, czyli krok całkowania. Następnie w pętli for, czyli sposobem iteracyjnym dodawane są do siebie wartość wyniku poprzedniej iteracji z wywołaniem funkcji f dla argumentu $a + i * h$. W odróżnieniu od metody trapezów w metodzie Simpsona rozróżniamy iterację parzystą oraz nieparzystą. Dla iteracji parzystej wartość wywołania funkcji f dla argumentu $a + i * h$ jest mnożone razy 4. Dla nieparzystych występuje mnożenie razy 2. Działanie dotyczy tylko wartości $y_1, \dots, y_{n-1}$. Po zakończeniu iteracji do zmiennej y, która przechowuje wynik dopisywany jest wynik z operacji dzielenia przez 3 mnożenia przez h (krok całkowania) sumy wywołania funkcji f dla dolnej i górnej granicy całkowania zgodnie ze wzorem:

$$\frac{h}{3}[y_0 + y_4 + 4 * y_1 + 2 * y_2 + 4 * y_3 + \dots + 2 * y_{n-2} + 4 * y_{n-1}]$$

Po obliczeniu zwracany jest końcowy wynik. Ponadto funkcja jest zabezpieczona przed podaniem błędnej liczby podprzedziałów. W razie wystąpienia błędnej wartości n wypisywany jest dla użytkownika stosowny komunikat.

3. Analiza porównawcza metody trapezów i metody Simpsona w całkowaniu numerycznym.

Dana jest całka oznaczona:

$$s = \int_0^{\frac{\pi}{2}} f(x)dx \; dla \; f(x) = \sqrt{1 + 0.6 * \sin(x)}$$

Stąd wynika, iż:

- dolny przedział całkowania wynosi 0,
- górny przedział całkowania wynosi $\frac{\pi}{2}$

Zgodnie z metodami numerycznymi obliczania całek, w tym przypadku zgodnie z metodą trapezów oraz metodą Simpsona należy określić ilość podprzedziałów, czyli dosłownie w przypadku moich obu funkcji zaimplementowanych w środowisku MATLAB ilość iteracji pętli. Poniżej zostanie przedstawiona analiza wyników powyższej całki dla stopniowo rosnących wartości ilości podprzedziałów.

$$Wynik \; metodą \; analityczną: \int_0^{\frac{\pi}{2}} \left(\sqrt{1 + 0.6 * \sin(x)}\right) dx = \mathbf{1.8422775827182}$$

Ilość podprzedziałów [n]	**Wynik dla: metoda trapezów**	**$\lvert A - a \rvert$ dla metody trapezów**	**Wynik dla: metoda Simpsona**	**$\lvert A - a \rvert$ dla metody Simpsona**
10	1.84166054739111	0.061703532709001	1.84227832089241	0.000073817
100	1.84227141419693	0.000616852127001	1.84227758279227	0.0000000073817
1000	1.84227752103317	0.0000061685	**1.84227758271821**	0.0000000000009992
10000	1.84227758210136	0.000000061684	1.84227758271821	0.0000000000009992
100000	1.84227758271202	0.00000000061802	**1.8422775827182**	0
1000000	**1.84227758271812**	0.000000000008015	1.8422775827182	0

Wnioski z analizy:

Obie z metod umożliwiają dość poprawne obliczenie całki. Metoda trapezów dla 10 podprzedziałów zwraca błędny wynik. Przy 100 podprzedziałach zwraca zbliżony wynik, lecz pomimo 1 000 000 podprzedziałów nie zwraca wyniku idealnego, czyli równego wynikowi obliczonemu metodą analityczną. Metoda Simpsona jest o wiele bardziej efektywna, ponieważ już dla 10 podprzedziałów zwraca wynik zbliżony do ideału. Dla 1000 wynik różni się tylko cyfrą na 13 miejscu po przecinku. Dla 100000 podprzedziałów metoda Simpsona zwróciła wynik równy ideałowi, czyli 1.8422775827182. Ponadto na podstawie analizy błędów bezwzględnych można stwierdzić, iż metoda Simpsona jest wydajniejsza i szybsza w dochodzeniu do wyniku. Spowodowane jest to tym, iż w przypadku metody trapezów przybliżenie funkcji w przedziale było określane funkcją liniową, a dla metody Simpsona przybliżenie funkcji opiera się na paraboli uzyskiwanej na podstawie interpolacji funkcji wielomianem drugiego stopnia, najczęściej wielomianem Lagrange'a.

4. Program w języku Java – metoda trapezów i metoda Simpsona.

Polecenie:
Wyznacz metodą trapezów i metodą Simpsona wartość:

$$s = \int_0^{\frac{\pi}{2}} f(x)dx \; dla \; f(x) = \sqrt{1 + 0.6 * \sin(x)}$$

Kod źródłowy:

```
import java.util.Scanner;
public class CalkaTrapezSimpson {
    private static Double a;
    private static Double b;
    private static Integer n;
    private static Double wynikTrapez = 0D;
    private static Double wynikSimpson = 0D;
    private static Scanner s = new Scanner(System.in);
    public static Double getA() {
        return a;
    }
    public static void setA(Double a) {
        CalkaTrapezSimpson.a = a;
    }
    public static Double getB() {
        return b;
    }
    public static void setB(Double b) {
        CalkaTrapezSimpson.b = b;
    }
    public static Integer getN() {
        return n;
    }
    public static void setN(Integer n) {
        CalkaTrapezSimpson.n = n;
    }
    public static Double getWynikTrapez() {
        return wynikTrapez;
    }
    public static void setWynikTrapez(Double wynikTrapez) {
        CalkaTrapezSimpson.wynikTrapez = wynikTrapez;
    }
    public static Double getWynikSimpson() {
        return wynikSimpson;
    }
    public static void setWynikSimpson(Double wynikSimpson) {
        CalkaTrapezSimpson.wynikSimpson = wynikSimpson;
    }

    public static void wprowadzDane() {
        Boolean koniec = false;
        String tmpA;
        String tmpB;
        String tmpN;
```

```
    while (!koniec) {
        try {
          System.out.println("Podaj wartosc granice dolna, granice gorna calkowania oraz
          ilosc podprzedzialow.");
            System.out.print(">> Dolna granica (a): ");
            tmpA = s.next();
            System.out.print(">> Gorna granica (b): ");
            tmpB = s.next();
            System.out.print(">> Ilosc podprzedzialow (n): ");
            tmpN = s.next();
            setA(Double.parseDouble(tmpA));
            if (tmpB.equals("pi/2")) {
                setB(Math.PI / 2);
            } else {
                setB(Double.parseDouble(tmpB));
            }
            setN(Integer.parseInt(tmpN));
            if (getA() > getB() || getN() < 2) {
                throw new Exception();
            } else {
                koniec = true;
            }
        } catch (Exception e) {
           System.out.println("\n[Wyjatek]: Blad! Sprobuj ponownie! Blad wpisania lub blad
          matematyczny: a<b oraz n>2!\n");
        }
    }
}
public static Double funkcja(Double x) {
    Double wynik = Math.sqrt(1 + 0.6 * Math.sin(x));
    return wynik;
}
public static void trapezCalka() {
    Double h = (getB() - getA()) / getN();
    for (int i = 1; i < getN(); i++) {
        setWynikTrapez(getWynikTrapez() + funkcja(a + i * h));
    }
    setWynikTrapez(h * (getWynikTrapez() + (funkcja(a) + funkcja(b)) / 2));
    System.out.println("\n[Metoda trapezow - wynik]: " + getWynikTrapez());
}
public static void simpsonCalka() {
    Double h = (getB() - getA()) / getN();
    try {
        if ((getN() & 1) != 0) {
            throw new Exception();
        } else {
            setWynikSimpson(funkcja(a) + funkcja(b));
            for (int i = 1; i < getN(); i++) {
                if ((i & 1) == 0) {
                    setWynikSimpson(getWynikSimpson() + 2 * funkcja(a + i * h));
                } else {
                    setWynikSimpson(getWynikSimpson() + 4 * funkcja(a + i * h));
                }
            } }
```

```
        } catch (Exception e) {
              System.out.println("\n[Metoda Simpsona - wyjatek!]: Ilosc podprzedzialow dla metody
              Simpsona musi byc liczba parzysta!");
            System.exit(-1);
        }
        setWynikSimpson((h * getWynikSimpson()) / 3);
        System.out.println("\n[Metoda Simpsona - wynik]: " + getWynikSimpson());
    }
    public static void main(String[] args) {
        wprowadzDane();
        trapezCalka();
        simpsonCalka();
    }
}
```

Uruchomienie programu:

1. Wprowadzenie przez użytkownika wartości argumentów: a – granica dolna całkowania, b – granica górna całkowania, n – ilość podprzedziałów:

```
Podaj wartosc granice dolna, granice gorna calkowania oraz ilosc podprzedzialow.

>> Dolna granica (a): 0

>> Gorna granica (b): pi/2

>> Ilosc podprzedzialow (n): 10000
```

2. Wypisanie wyniku:

```
[Metoda trapezow - wynik]: 1.8422775821013613

[Metoda Simpsona - wynik]: 1.842277582718207
```

Opis programu:

Program składa się z jednej klasy publicznej, która posiada 6 pól statycznych. Są to:

- a – pole statyczne przechowujące wartość dolnej granicy całkowania,
- b – pole statyczne przechowujące wartość górnej granicy całkowania,
- n – pole statyczne przechowujące ilość podprzedziałów,
- wynikTrapez – pole statyczne zainicjowane wartością 0 przechowujące wynik całki obliczonej metodą trapezu,
- wynikSimpson – pole statyczne zainicjowane wartością 0 przechowujące wynik całki obliczonej metodą Simpsona.

Ponadto w programie występują 4 metody statyczne nie wliczając getterów, setterów oraz metody inicjującej main. Są to:

- funkcja – metoda przechowująca w zmiennej lokalnej wygląd funkcji. Przyjmuje jeden argument x, który używany jest do obliczania wartości funkcji. Metoda zwraca wynik funkcji w danym punkcie x.
- wprowadzDane – metoda odpowiadająca za wprowadzenie danych przez użytkownika. Występuje tu obsłużenie wyjątków w sytuacji, gdy użytkownik wpisze błędną liczbę podprzedziałów lub w przypadku, gdy dolna granica całkowania będzie większa od górnej granicy. Ponadto obsłużono również wyjątek wpisywania przez użytkownika liter tam, gdzie wymagane są wartości liczbowe.
- trapezCalka – metoda obliczająca całkę z funkcji metodą trapezów zgodnie z algorytmem opisanym w punkcie 1 sprawozdania, w opisie programu w środowisku MATLAB.
- simpsonCalka - metoda obliczająca całkę z funkcji metodą Simpsona zgodnie z algorytmem opisanym w punkcie 2 sprawozdania, w opisie programu w środowisku MATLAB. Metoda nie wypisze wyniku w przypadku, gdy ilość podprzedziałów zapisana w polu n będzie liczbą nieparzystą.

Program po wywołaniu wszystkich metod i wpisaniu danych przez użytkownika wypisuje oczekiwany wynik.