

Wojciech Myszka

Technologie Informacyjne
Mechatronika 2010/2011
Trochę programowania
wersja: 4 z drobnymi modyfikacjami!

2011-01-07 20:16:34 +0100

Spis treści

1. Wstęp	2
1.1. Cel laboratorium	2
1.2. Wymagania	2
1.3. Materiały	2
1.3.1. Funkcje	2
1.3.2. Rekurencja	3
2. Struktura laboratorium	3
3. Zadania do wykonania	4
3.1. Rekurencje wyliczanie wartości Największego Wspólnego Dzielnika	4
Zadania	4
3.2. Ciąg Fibonacciego	5
3.3. Algorytm E	5
Schemat blokowy	5
Zadania	6
3.4. Algorytm B	6
Zadania	6
3.5. Metoda Newtona-Raphsona: pierwiastek dowolnego stopnia	6
3.5.1. Realizacja programowa	8
3.5.2. Zadania treningowe	8
4. Materiały pomocnicze	8
4.1. Potrzebne instrukcje języka Python	8
5. Wersja PDF dokumentu	8
Literatura	8

1. Wstęp

Tak nieszczęśliwie się złożyło, że niektóre grupy będą miały siedem laboratoriów inne zaś osiem (co zależy od parzystości tygodnia).

Zatem ostatnie laboratorium (ostatnie dwa laboratoria) poświęcone będą programowaniu w języku Python.

Tematyka zajęć w kolejnym semestrze będzie obejmowała programowanie w języku C — nie powinien to być specjalny problem.

Studenci na zajęciach siódmych rozpoczynają realizację zadań od „wprawek” (rozdział 2), a następnie kontynuują zadania do wykonania (rozdział 3) tyle ile dadzą rady. Im kto więcej zrobi tym lepsza ocena.

Zatem dla tych, którzy istotnie mają zajęcia ósme:

- powinny być one potraktowane w pierwszej kolejności jako zajęcia „odróbkowe”
- można też kontynuować zadania z laboratorium siódmego.

1.1. Cel laboratorium

Celem laboratorium jest zmierzenie się z (nowym?) językiem programowania i zaprogramowanie bardzo prostych problemów. Przy czym chciałbym aby samo programowanie poprzedzone było narysowaniem prostego schematu blokowego. Jako podstawowy zestaw bloków stosowanych na schematach blokowych można przyjąć ten z Wikipedii.

1.2. Wymagania

Zapoznanie się z elementarną dokumentacją programu Python (instrukcja laboratoryjna nr 4 [1] i/lub bardziej zaawansowaną dokumentacją po polsku dostępną on-line [2]).

1.3. Materiały

1.3.1. Funkcje

Bardzo często programując w jakimś języku programowania musimy skorzystać z jakiejś funkcji. Python dostarcza bardzo wiele funkcji, a na przykład najbardziej podstawowe funkcje matematyczne dostępne są w module `math`. Na początku programu piszemy:

```
import math
```

a później możemy z funkcji korzystać swobodnie:

```
print math.sin(30.*math.pi/180.)
```

(Sprawdź jaki będzie wynik!)

Mozemy również zdefiniować własną funkcję. Będzie to funkcja $f(x) = 3x^2 - 5x + 2$. (Zwracam uwagę na wcięcie!)

```
def f(x):  
    a = 3.
```

```
b = -5.  
c = 2.  
return a * x*x + b * x + c
```

Po jej zdefiniowaniu możemy już funkcji używać:

```
print f(1)
```

albo

```
z = 5 + f(10)
```

albo

```
for x in range (-10, 11):  
    print x, ":", f(x)
```

W powyższym przykładzie x jest zmienną niezależną (tak jak w funkcji $\sin(x)$), a polecenie **return** powoduje wyliczenie wartości i „podstawienie jej pod $f(x)$ ”. Funkcję można zdefiniować również tak:

```
def f(x):  
    a = 3.  
    b = -5.  
    c = 2.  
    y = a * x*x + b * x + c  
    return y
```

Teraz polecenie **return** zwraca (wyliczoną wcześniej) wartość y jako wartość funkcji $f(x)$.

1.3.2. Rekurencja

Poniżej rekurencyjna definicja funkcji silnia

```
def silnia(n):  
    if n == 0:  
        return 1  
    else:  
        return n*silnia(n-1)
```

Sprawdź czy funkcja działa.

2. Struktura laboratorium

Zaczynamy od wprawek.

1. Co robi ta instrukcja? Jaki będzie wyniki? Sprawdź! Zobacz co się będzie działo gdy zmienisz wartość a (na ujemną, na przykład). (Poniżej \square oznacza odstęp.)

```

a = 5
if a > 0:
    print "a_dodatnie!"
else:
    print "a_ujemne!"

```

Zmodyfikuj program tak, by rozpoznawał przypadek gdy a jest równe zero i informował o tym.

2. Co robi ten program? Jaki będzie wynik?

```

for i in range(7, 9)
    print i

```

Zmodyfikuj go tak aby drukował tablicę funkcji x^2 dla x zmieniającego się od -10 do 10 włącznie.

3. Co robi poniższy program?

```

a = 1536
while a%2 == 0:
    a = a/2
    print a

```

Przy okazji, jaki będzie wynik programu?

```

while 1:
    print "Dosc!"

```

A tego:

```

while 0:
    print "Dosc!"

```

Peksperymentuj i opisz działanie instrukcji while.

3. Zadania do wykonania

3.1. Rekurencyjne wyliczanie wartości Największego Wspólnego Dzielnika

Wariant rekurencyjny wyznaczania NWD wygląda jakoś tak: $gcd(k, n) = n$ gdy $k = 0$ natomiast $gcd(k, n) = gcd(n \bmod k, k)$ gdy $k > 0$.

Zadania

1. Zaprogramuj w Pythonie funkcję gcd.
2. Porównaj wyniki liczone każdą z trzech metod.

3.2. Ciąg Fibonacciego

Ciąg Fibonacciego jest jednym z wielu przykładów „operacji” zdefiniowanej rekurencyjnie.

Zadaniem jest zaprogramowanie (w Pythonie) rekurencyjnej funkcji wyliczającej zadany wyraz ciągu.

Dodatkowo program powinien zliczać liczbę wywołań funkcji. W tym celu należy jedną zmienną przeznaczyć na licznik i zaraz po wejściu do funkcji zwiększać ten licznik o jeden.

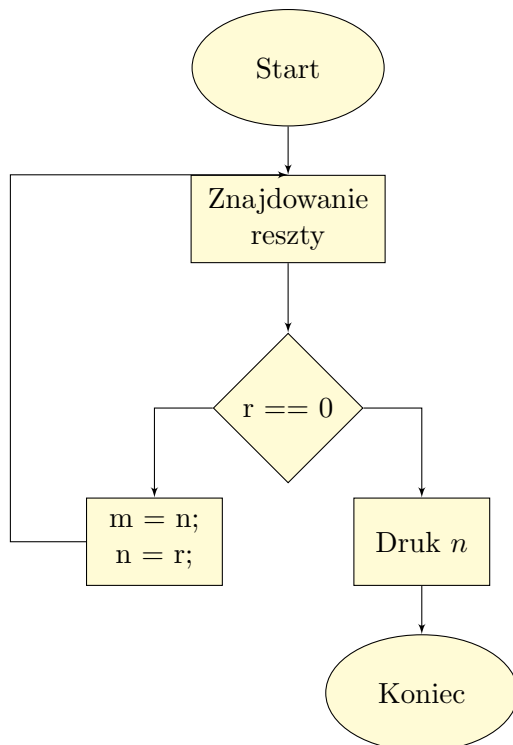
Przed zakończeniem obliczeń program powinien wyświetlać wyliczony wyraz ciągu oraz liczbę wywołań funkcji.

3.3. Algorytm E

Oto jedna z jego wersji algorytmu Euklidesa: *Dane są dwie dodatnie liczby całkowite m i n , należy znaleźć ich **największy wspólny dzielnik (NWD)** tj. największą dodatnią liczbę całkowitą, która dzieli całkowicie zarówno m jak i n .*

1. [Znajdowanie reszty] Podziel m przez n i niech r oznacza resztę z tego dzielenia. (Mamy $0 \leq r < n$.)
2. [Czy wyszło zero?] Jeśli $r = 0$ zakończ algorytm; odpowiedzią jest n .
3. [Upraszczenie] Wykonaj $m \leftarrow n$, $n \leftarrow r$ i wróć do kroku 1.

Schemat blokowy



Zadania

1. Spróbuj zaprogramować w Pythonie Algorytm E.

3.4. Algorytm B

1. Przyjmij $k \leftarrow 0$, a następnie powtarzaj operacje: $k \leftarrow k + 1$, $u \leftarrow u/2$, $v \leftarrow v/2$ zero lub więcej razy do chwili gdy przynajmniej jedna z liczb u i v przestanie być parzysta.
2. Jeśli u jest nieparzyste to przyjmij $t \leftarrow -v$ i przejdź do kroku 4. W przeciwnym razie przyjmij $t \leftarrow u$.
3. (W tym miejscu t jest parzyste i różne od zera). Przyjmij $t \leftarrow t/2$.
4. Jeśli t jest parzyste to przejdź do 3.
5. Jeśli $t > 0$, to przyjmij $u \leftarrow t$, w przeciwnym razie przyjmij $v \leftarrow -t$.
6. Przyjmij $t \leftarrow u - v$. Jeśli $t \neq 0$ to wróć do kroku 3. W przeciwnym razie algorytm zatrzymuje się z wynikiem $u \cdot 2^k$.

Zadania

1. Narysuj schemat blokowy algorytmu B
2. Spróbuj zaprogramować w Pythonie Algorytm B.

3.5. Metoda Newtona-Raphsona: pierwiastek dowolnego stopnia

Załóżmy, że mamy wyznaczyć pierwiastek stopnia n z liczby w , czyli znaleźć taką liczbę x , że:

$$x^n = w \tag{1}$$

lub inaczej:

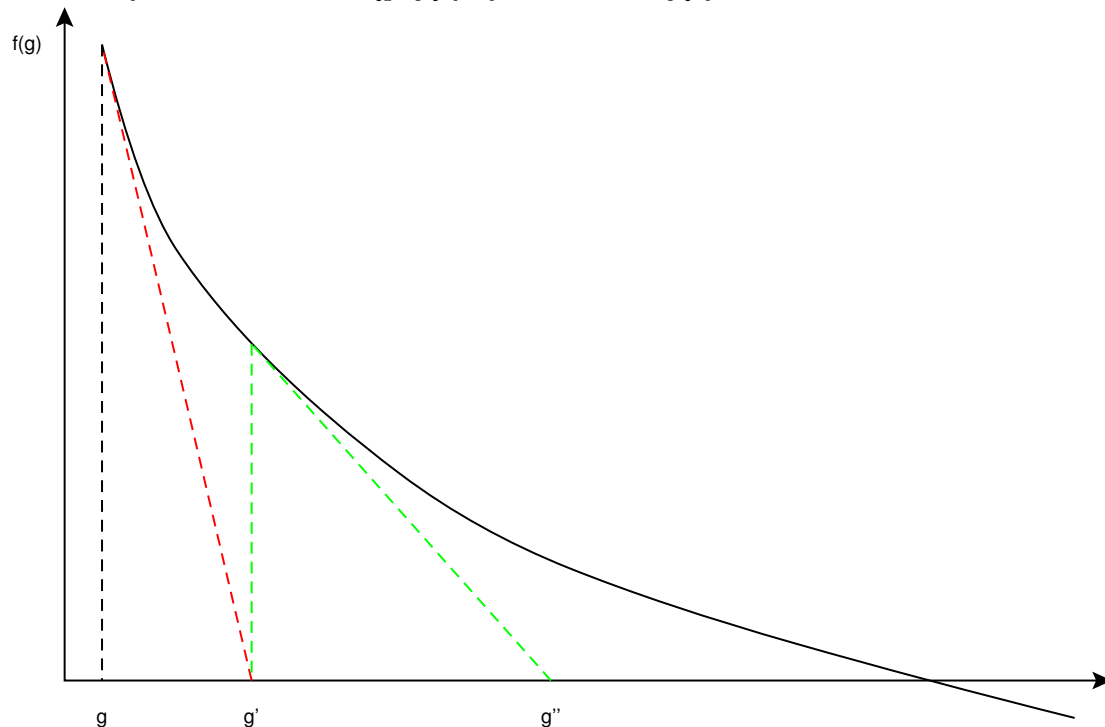
$$x^n - w = 0 \tag{2}$$

Jeżeli oznaczymy $f(x) = x^n - w$ to zadanie to można zapisać ogólniej: należy znaleźć takie x , że $f(x) = 0$.

Jeżeli zadanie dodatkowo uprościmy zakładając:

- funkcja ma dokładnie jedno miejsce zerowe,
- jest różniczkowalna,
- jej pochodna w całym przedziale jest albo dodatnia albo ujemna;

to możemy naszkicować następujący rysunek ilustrujący nasze zadanie:



Zaczynamy w punkcie g ; wartość funkcji w tym punkcie wynosi $f(g)$. Musimy w jakiś sposób zdecydować w którym kierunku należy wykonać następny krok. Zauważmy, że możemy w tym celu wykorzystać pochodną (czerwona, przerywana linia na powyższym rysunku). Jeżeli przybliżymy funkcję za pomocą pochodnej (stycznej do funkcji, przechodzącej przez punkt $(g, f(g))$ to następnym przybliżeniem będzie punkt przecięcia stycznej z osią x .

Z równania prostej mamy:

$$\frac{f(g) - 0}{g - g'} = f'(g) \quad (3)$$

czyli

$$\frac{f(g)}{f'(g)} = g - g' \quad (4)$$

i dalej

$$g' = g - \frac{f(g)}{f'(g)} \quad (5)$$

Jeżeli zauważymy, że $f(x) = x^n - w$ oraz, że $f'(x) = nx^{n-1}$ to kolejne przybliżenie wyliczane będzie ze wzoru:

$$g' = g - \frac{g^n - w}{ng^{n-1}} \quad (6)$$

albo

$$g' = \frac{ng^n - g^n + w}{ng^{n-1}} = \frac{(n-1)g^n + w}{ng^{n-1}} = \frac{1}{n} \left((n-1)g + \frac{w}{g^{n-1}} \right) \quad (7)$$

Gdy $n = 2$, wówczas

$$g' = \frac{1}{2} \left(g + \frac{w}{g} \right). \quad (8)$$

3.5.1. Realizacja programowa

Program będzie się składał z trzech części:

1. `blisko(a, b)` – funkcja o wartościach logicznych sprawdzająca czy $|a - b| \leq \varepsilon$,
2. `lepszy(w, g)` – funkcja rzeczywista wyliczająca następne, lepsze przybliżenie pierwiastka,
3. `pierwiastek(n, w, g)` – funkcja (rzeczywista) wyliczająca pierwiastek stopnia n z w zaczynając od przybliżenia g .

3.5.2. Zadania treningowe

1. Narysować schematy blokowe poszczególnych funkcji.
2. Zaprogramować w języku Python program wyliczający dla zadanej liczby, z wykorzystaniem trzech powyższych funkcji, pierwiastek z zadanej (wczytanej) (**bez rekurencji!**).
3. Zaprogramować wersję „rekurencyjną” powyższego algorytmu.

4. Materiały pomocnicze

4.1. Potrzebne instrukcje języka Python

1. Instrukcja warunkowa
2. Instrukcja `while`
3. Definicje funkcji

5. Wersja PDF dokumentu

Wersja PDF dokumentu

Literatura

- [1] Wojciech Myszka. Technologie informacyjne mechatronika 2010/2011 błędy obliczeń. python. Instrukcja laboratoryjna dostępna: <http://www.immt.pwr.wroc.pl/~myszka/MCM1001/L04/104.html>, 2010.
- [2] Mark Pilgrim. *Dive Into Python*. Apress, Lipiec 2004. Dostępne jest polskie tłumaczenie on-line: http://pl.wikibooks.org/wiki/Zanurkuj_w_Pythonie zatytułowane *Znurkuj w Pythonie*.