Laboratorium programistyczne: metody numeryczne (interpolacja, różniczkowanie i całkowanie)

Projekt „Matematyka dla Ciekawych Świata”,
Robert Ryszard Paciorek
<rrp@opcode.eu.org>

2018-05-23

1 Wielomiany

Najczęściej spotykanym zapisem wielomianu jest postać sumy jednomainów:

$$w(x) = \sum_{i=0}^{n} a_i x^i = a_0 + a_1 x + ... + a_n x^n$$

Żeby wiedzieć, o jaki wielomian chodzi, wystarczy że okreslimy listę jego współczynników $a_0 \dots a_n$.

1.1 Obliczanie wartości wielomianu

Niech a będzie listą kolejnych współczynników wielomianu $a_0 \dots a_n$. Do obliczenia wartości wielomianu w zadanym punkcie x naturalne wydaje się zastosowanie powyższego wzoru w następujący sposób:

```
def oblicz(a, x):
    w = 0
    for i in range(len(a)):
        w += a[i] * (x**i)
    return w
```

Niestety podejście takie jest bardzo nieefektywne, gdyż w każdym kolejnym kroku sumy musimy wykonać coraz więcej mnożeń. W ogólności w kroku n-tym musimy wykonać n mnożeń, co przekłada się na czasową złożoność obliczeniową rzędu $\mathcal{O}(n^2)$, co oznacza że czas wykonania rośnie z kwadratem ilości współczynników wielomianu (czyli, ogólnie rzecz biorąc, ze stopniem wielomianu). Innymi słowy, jeśli zwiększymy 10 razy stopień wielomianu, to czas liczenia zwiększy się nam $100 = 10^2$ razy. Ponieważ będziemy czasem potrzebowali wielomianów wysokiego stopnia — o czym za chwilę — nie jest to dla nas najlepsza sytuacja.

Jeżeli zauważymy, że potrzebujemy policzyć kolejno wszystkie potęgi wartości x, aż do n to algorytm ten możemy znacznie przyspieszyć implementując liczenie potęgi w tej samej pętli co sumowanie:

```
def oblicz(a, x):
    p = 1
    w = a[0]
    for i in range(1, len(a)):
        p = p * x
        w += a[i] * p
    return w
```

Zauważ, że dodatkowo wyrzuciliśmy a_0 poza pętlę. W ten sposób potrzebujemy już tylko n sumowań i $2n$ mnożeń (pomijając narzut związany z obsługą pętli itp).

Jeżeli chcemy jeszcze szybciej obliczyć wartość wielomianu w punkcie x możemy skorzystać z *schematu Hornera*, co pozwoli nam zredukować o połowę liczbę mnożeń. Wymaga on przekształcenia postaci w której zapisujemy wielomian poprzez powyciąganie x-ów przed nawias:

$$a_0 + x(a_1 + x(a_2 + ... + x(a_{n-1} + xa_n)))$$

i rozpoczęcia obliczeń od a_n.

```
def oblicz(a, x):
    w = 0
    for i in range(len(a)-1, 0, -1):
        w = x * (a[i] + w)
    w += a[0]
    return w
```

1.2 Postać Newtona

Wielomian możemy przedstawić także w innych postaciach. Jedną z takich postaci jest *postać Newtona*:

$$w(x) = b_0 + b_1(x - x_0) + b_2(x - x_1)(x - x_0) + ... + b_n(x - x_{n-1}) ... (x - x_0)$$

w której wielomian stopnia n określony jest przez $n + 1$ współczynników b_i oraz n punktów x_i.

Zadanie 1.2.1

Napisz funkcję `oblicz_wart(xx, x, b)` obliczającą wartość wielomianu w postaci Newtona. Funkcja powinna otrzymywać argumenty: `xx` - punkt w którym ma zostać obliczona wartość, `x` - lista współczynników x_i, `b` - lista współczynników b_i.

Wskazówka: do testowania funkcji możesz użyć wielomianu $w(x)$ zdefiniowanego przez `x = [-1, 1]` i `b = [2, 1, -3]`, który przyjmuje następujące wartości: $w(-2) = -8$, $w(-1) = 2$, $w(0) = 6$, $w(1) = 4$, $w(2) = -4$.

2 Interpolacja

Interpolacja to zgadywanie wartości nieznanych na podstawie znanych. Dokładniej, jest to wyznaczanie przybliżonych wartości nieznanej funkcji, w oparciu o znane wartości w tzw. punktach węzłowych, które też znamy. Realizujemy to wyznaczając tzw. funkcję interpolacyjną, która w punktach węzłowych przyjmuje ustalone wartości. Innymi słowy, wybieramy sobie pewną postać funkcji (np. decydujemy, że nasza szukana funkcja interpolacyjna będzie wielomianem pewnego stopnia) i dopasowujemy ją tak, by przechodziła przez znane wartości w znanych punktach.

2.1 Interpolacja wielomianowa

Bardzo często jako funkcje interpolacyjne stosowane są funkcje wielomianowe. W tym wypadku zadanie interpolacji to po prostu zadanie znalezienia wielomianu przechodzącego przez dane punkty.

2.1.1 Obliczanie współczynników interpolacyjnych

Możliwe jest obliczanie współczynników a_i interpolowanego wielomianu zapisanego w postaci potęgowej, jednak ze względów obliczeniowych bardziej praktyczne jest posłużenie się postacią Newtona. Realizowane jest to przy pomocy następującego algorytmu:

```
j = 0
while j <= n:
        b[j] = f[j]
        j += 1

j = 1
while j <= n:
        k = n
        while k >= j:
                b[k] = (b[k] - b[k-1]) / (x[k] - x[k-j])
                k -= 1
        j += 1
```

gdzie `n` jest stopniem wielomianu interpolacyjnego (obliczanego dla $n+1$ punktów węzłowych), `x[i]` jest wartością x dla i-tego punktu, a `f[i]` wartością funkcji w i-tym punkcie węzłowym.

Zadanie 2.1.1

Napisz funkcję `oblicz_wsp(x, y)` obliczającą współczynniki wielomianu interpolacyjnego dla danego zbioru punktów węzłowych (określonych przez listy `x`, `y`). Wykorzystaj ją do narysowania wykresu wielomianu interpolującego następującą funkcję:

x	$f(x)$
1	8
3	-5
6	3
7	9

2.2 Interpolacja trygonometryczna

Interpolacja wielomianowa, ze względu na swoją naturę, nie za bardzo sprawdza się w przypadku funkcji okresowych. Stosowana jest wtedy często interpolacja trygonometryczna, w której funkcja interpolująca ma postać:

$$f(x) = a_0 + \sum_{k=1}^{m} a_k \cos(kx) + \sum_{k=1}^{m} b_k \sin(kx)$$

W szczególnym przypadku nieparzystej liczby n równoodległych[1] punktów $x_i = i \cdot \frac{2\pi}{n}$ ta funkcja jest określona przez następujące wzory:

- $m = \frac{n-1}{2}$
- $a_0 = \frac{1}{n} \sum_{k=0}^{n-1} f(x_k)$
- $a_j = \frac{1}{n} \sum_{k=0}^{n-1} f(x_k) \cdot \cos(j \cdot x_k)$
- $b_j = \frac{1}{n} \sum_{k=0}^{n-1} f(x_k) \cdot \sin(j \cdot x_k)$

3 Całkowanie

Najprostszą podejściem do numerycznego obliczania całki oznaczonej z danej funkcji byłoby liczenie sumy pól prostokątów pod wykresem funkcji dla odpowiednio małego boku dx takiego prostokąta w współrzędnej X. Za wysokość prostokąta możemy przyjmować wartość funkcji w początku, środku lub końcu danego przedziału. Prostym usprawnieniem tej metody, nie wymagającym znacznie większej ilości obliczeń ani nie

1. W przypadku punktów równoodległych rozwiązaniem jest dyskretna transformata Fouriera

powodującej znacznego wzrostu skomplikowania programu, jest zastosowanie trapezów zamiast prostokątów.

Zadanie 3.0.1

Napisz funkcję `calka(f, a, b, dx = 0.3)` obliczającą wartość całki określonej na odcinku od `a` do `b` z funkcji `f`, korzystającą z metody trapezów.
Wskazówka, zauważ że dodając do `a` kolejne wartości `dx` możemy przekroczyć `b`.

4 Różniczkowanie

Najprostszą metodą obliczania (przybliżania) wartości pochodnej funkcji w danym punkcie jest obliczenie jej jako: $f'_x = \frac{f(x+dx)-f(x)}{dx}$, gdzie dx jest odpowiednio małą odległością pomiędzy dwoma punktami.

Zadanie 4.0.1

Napisz funkcję obliczającą pochodną funkcji $sin(x)$, w oparciu o powyższą zależność. Funkcja powinna posiadać argumenty pozwalające na wskazanie punktu x oraz wartości dx używanej do obliczeń. Zobacz jak obliczane przybliżenie pochodnej zalezy od wartości dx.

Główny problem takiego podejścia polega na znalezieniu odpowiednio małego dx. Można go jednak ominąć traktując f'_x jako funkcję zależną od dx, obliczając jej wartości dla kilku dx i interpolując jej wartość w zerze.

Zadanie 4.0.2

Napisz funkcję `pochodna(f, x, d = 0.3)` obliczającą wartość pochodnej funkcji $f(x)$ (przekazanej w argumencie `f`) w zadanym punkcie `x`. Funkcja powinna obliczać pochodną interpolując wartości $f'_x(dx)$ obliczonej dla $dx = -2d$, $dx = -d$, $dx = d$ i $dx = 2d$.

5 Regulator PID

Regulator PID jest to algorytm regulacji parametru procesu działający w pętli sprzężenia zwrotnego posiadający człony: proporcjonalny (P), całkujący (I) i różniczkujący (D).

Wejściem algorytmu jest wartość mierzona (bądź od razu różnica wartości zadanej i mierzonej). Jeżeli kierunek zmiany sterowania jest zgodny ze zmianą wartości mierzonej (zwiększenie wartości sygnału sterującego powoduje zwiększenie wartości mierzonej) to należy odejmować wartość mierzoną od zadanej, w przeciwnym razie od zadanej mierzoną.

Wyjściem algorytmu jest wartość zmiany jakiegoś sygnału sterującego - może być wykorzystana bezpośrednio przy sterowaniu krokowym lub akumulowana celem uzyskania stałej wartości sygnału sterującego.

Typowa algorytm ten działa w nieskończonej pętli, często z dodatkowym krokiem czasowym (odstępem pomiędzy wykonaniami kolejnych obiegów pętli) - deklarowanym jawnie lub wynikłym z kostrukcji układu sterującego.

Przykładowa implementacja algorytmu:

```
class PID:
  # nastawa - wartość zadana
  setPoint = 0

  # wartość wyjścia dla sterowania krokowego
  # (gdy akumulacja w układzie realizującym)
  outStep = 0

  # wartość wyjścia dla sterowania sygnałem
  outValue = 0
```

```
# parametry regulatora PID
Kp, Ki, Kd = 0, 0, 0

# limity wartości sterowanej
outValueMin, outValueMax = 0, 0

# poprzednia różnica między wartością zadaną a otrzymaną
# (poprzedni błąd regulacji / uchyb)
lastDiff = 0

# poprzednia wartość otrzymana (zmienna procesu)
lastVal = 0

# część całkująca, akumulowana pomiędzy krokami
integral = 0

def doStep(self, inputVal, stepTime):
  # obliczmy aktualny błąd regulacji
  # (na podstawie odczytanej wartości wejściowej)
  diff = self.setPoint - inputVal

  # wyłączamy regulację gdy prowadziłby do przesterowania
  if (self.outValue > self.outValueMax and diff > 0):
    return 1
  if (self.outValue < self.outValueMin and diff < 0):
    return -1

  # całkowanie przybliżamy jako jako suma pól trapezów
  self.integral += (diff + self.lastDiff) / 2 * stepTime

  # różniczkowanie przybliżamy jako tangens nachylenia
  # prostej pomiędzy poprzednim krokiem a obecnym
  # celem złagodzenia odpowiedzi na zmiany wartości zadanej
  # różniczkujemy sygnał wejściowy a nie błąd regulacji
  derivative = -(inputVal - self.lastVal) / stepTime

  # obliczenie wartości sygnału sterującego na podstawie tego kroku
  self.outStep   = self.Kp*diff + self.Ki*self.integral + \
                   self.Kd*derivative

  # akumulacja sygnału sterującego
  self.outValue += self.outStep

  # zapamiętanie aktualnego błędu regulacji
  # jako poprzedni dla następnego kroku
  self.lastDiff = diff

  # zapamiętanie aktualnej wartości wejściowej
  # jako poprzedniej dla następnego kroku
  self.lastVal  = inputVal
```

```
    return 0
```

Implementacja ta wymaga ustawienia parametrów pracy algorytmu takich jak `setPoint`, `outValueMin`, `outValueMax`, współczynniki `Kp`, `Ki`, `Kd`. Przydatne może być też zainicjowanie `lastVal` na obecny stan wejścia. Następnie działanie odbywa się w pętli:

1. odczyt wejścia
2. obliczenie wartości sterującej z użyciem PID (wywołanie metody `doStep`)
3. wystawienie wartości sterującej
4. opcjonalne odczekanie jakiegoś czasu

Do testowania algorytmu posłuży nam prosty model zbiornika, w którym chcemy utrzymać zadany poziom wody, a z którego cały czas odpływa jakaś jej ilość:

```
poziom = 15
wyplyw = 2

def getInput():
        global poziom
        return poziom;

def setOutput(doplyw):
        global poziom
        poziom -= wyplyw

        if poziom < 0:
                poziom = 0

        if doplyw > 100:
                doplyw = 100
        if doplyw < 0:
                doplyw = 0

        poziom += 0.3 * doplyw
```

Zadanie 5.0.1

Napisz funkcję `testuj(steps, kp, ki, kd)` testującą działanie algorytmu PID. Funkcja powinna wykonać `steps` iteracji algorytmu dla zadanych parametrów `kp`, `ki`, `kd` i narysować wykres zmian wartości poziomu wody w zbiorniku (zwracanej przez `getInput()` w danym kroku) oraz wartości sterującej zaworem dopływowym (przekazywanej do `setOutput()` w danym kroku).

Zadanie 5.0.2

Korzystając z funkcji `testuj(steps, kp, ki, kd)` poeksperymentuj z doborem wartości współczynników algorytmu.

6 Zadania dodatkowe

Zadanie 6.0.1

Napisz funkcję `oblicz_wart_tryg (xx, a, b)` obliczającą wartość interpolacji trygonometrycznej w punkcie `xx`, dla podanych list współczynników `a` i `b`.

Zadanie 6.0.2

Napisz funkcję `oblicz_wsp_tryg (y)` obliczającą i zwracającą współczynniki `a` i `b` interpolacji trygonometrycznej dla podanego w skrypcie przypadku.
Wskazówka: zauważ że nie obliczany i nie wykorzystywany jest współczynnik `b[0]`

Zadanie 6.0.3

Napisz funkcje obliczającą całkę jako sumę pól prostokątów o wysokości określonej przez prawy koniec przedziału o długości dx.

Zadanie 6.0.4

Napisz program dokonujący interpolacji następującego zbioru punktów:

x	$f(x)$
-3	58
-2	39
-1	-4
1	6
2	-37
3	-56

Narysuj wykres funkcji interpolacyjnej w przedziale [-3.7 3.7] oraz podaj wzór wielomianu interpolacyjnego w postaci potęgowej.

Zadanie 6.0.5

Napisz program który będzie dokonywał interpolacji wielomianowej funkcji $cos(x)$ opierając się kolejno na: 3, 6, 13 i 16 wybranych punktach z przedziału $[0, 4\pi]$ Program powinien narysować wykresy kolejnych funkcji interpolujących oraz wykres $cos(x)$ na wspólnym wykresie.

Zadanie 6.0.6

Na zajęciach pisaliśmy funckję przybliżającą wartoć pochodnej w oparciu o zależność $f'_x = \frac{f(x+dx)-f(x)}{dx}$, innym podejściem jest obliczanie jej jako $f'_x = \frac{f(x+dx)-f(x-dx)}{2 \cdot dx}$. Napisz funkcją obliczającą przybliżenie pochodnej funkcji $cos(x)$ obliczane jako $f'_x = \frac{f(x+dx)-f(x-dx)}{2 \cdot dx}$. Która z tych metod wydaje się lepsza.

Zadanie 6.0.7

Napisz program która oblicza numerycznie wartość pochodnej danej funkcji (np. $cos(x)$) w danym przedziale (z jakimś ustalonym krokiem) i rysuje wykres funcji oraz tej pochodnej w zadanym przedziale. Do obliczania pochodnej możesz użyć dowolnej metody omawianej na zajęciach.

Zadanie 6.0.8

Narysuj wykres funkcji $sin(x)$ dla x z przedziału [1,2] oraz styczne w punkcie $x = 1.4$ obliczone jako krzywe o nacyleniu odpowiadającemu przybliżeniu pochodnej obliczonemu jako: $f'_x = \frac{f(x+dx)-f(x)}{dx}$ i jako $f'_x = \frac{f(x+dx)-f(x-dx)}{2 \cdot dx}$ dla dx=0.13.