Wojciech Myszka

Metody numeryczne MCM037202 wer. 50 z drobnymi modyfikacjami!

2023-01-05 16:31:09 +0100

Spis treści

1. Wstęp

Na tych stronach znajdują się podstawowe informacje dotyczące kursu MCM037202 „Metody numeryczne” dla studentów kierunku Mechatronika. Kurs projektowany był dosyć już dawno, a w roku akademickim 2015/2016 odbyła się jego premiera. Pisząc oficjalną kartę katalogową kursu nie uwzględniłem, że będzie odbywał się on na siódmym semestrze i, że zajęcia odbywać się będą w trybie 5×3, a nie w $7 \times 2 + 1$. Stąd układ zajęć będzie się nieco różnił.

Kurs odbywa się wyłącznie w formie projektu, zakłada więc, ze podstawy teoretyczne studenci będą musieli odświeżyć sobie (lub zdobyć) samodzielnie. Tu są jedynie najbardziej podstawowe wiadomości dotyczące głównie arytmetyki komputerowej.

Lista literatury pomocniczej znajduje się na samym końcu.

Podczas zajęć używać będziemy programów Mathematica i Matlab. Zakładam, że z tym drugim programem studenci kierunku Mechatronika mieli już kontakt. Pierwszy jest znacznie mniej popularny, ale z różnych względów wart poznania. Programy mają swoją dokumentację (dostępną również on-line). W większości praktycznych zastosowań zamiast programu Matlab można używać programu Scilab. Jego zaletą jest to, że jest to program darmowy i może być legalnie używany na komputerach domowych.

Interesującą alternatywą dla opisanych powyżej programów jest Python (zwłaszcza w połączeniu z notatnikiem Jupyter i możliwościami oferowanymi przez biblioteki SciPy, NumPy czy Pandas). Zachęcam do korzystania z tych możliwości. Aby start ułatwić przygotowałem kilka notatników związanych bezpośrednio z tematyką laboratorium.

Sympatyczną metodą pozyskania oprogramowania Mathematica jest zdobycie komputerka Raspberry Pi. Mathematica jest częścią Raspbiana instalowanego via NOOBS. Uważam, że każdy student Mechatroniki powinien jakiś projekt z wykorzystaniem Maliny zrealizować.

Warto też zapoznać się z zadaniami przygotowywanymi przed dr. inż. Macieja Panka.

Wszystkie te informacje w postaci jednego pliku.

1.1. Zasady zaliczania

Proponuję zatem następujące zasady:

1. Być może czasami rzucę jakieś pytanie teoretyczne i punktował będę poprawne odpowiedzi.
2. Na zakończenie każdych zajęć prosił będę o zaprezentowanie uzyskanych wyników.
3. Po każdych zajęciach należy przesłać niezbyt długie sprawozdanie. Do przesyłania sprawozdań używamy e-portalu.
4. Istnieje możliwość zaliczania zajęć indywidualnym projektem.
   Zajęcia są obowiązkowe.

1.2. Oprogramowanie

Podczas tworzenia zajęć przyjąłęm następujące zasady:

1. Trzeba będzie programować.
2. Założyem, że studenci znają
   — język programowania C (drugi semestr studów),
   — być może język programowania C++;
   przyjąłem również, że:
   — mieli szansę zapoznać się w trakcie zajęć z Matlabem.
3. Przyjąłem, że w ich curriculum powinny pojawić się co najmniej elementarna wiedza o nastepujących zagadnieniach:
   — język programowania Python (i śrdowisko Jupyter),
   — język programowania Wolfram Language i środowisko Mathematica,
   — „sztuczna inteligencja”, *machine learning*, i zagadnienia pokrewne...
4. Znakomita większość potrzebnego oprogramowania zainstalowana będzie w laboratorium komputerowym (sala 604/B1).
   Pandemia skutecznie zniszczyłą te założenia. W szczególności

— utrudniony jest dostęp do sali 604/B1
— oprogramowanie Mathematica i Matlab to dosyć drogie oprogramowanie komercyjne.

1.3. Oprogramowanie zastępcze

1.3.1. Czym zastąpić Matlaba?

Prosta odpowiedź jest taka: nie da się!

Jak bliżej przyjrzeć się problemowi to można znaleźć dwa **darmowe** programy mogące go zastąpić:

1. GNU Octave,
2. Scilab.

GNU Octave

Fachowcy oceniają, że Octave jest w 99% zgodne z Matlabem. W większości przypadków, znakomita większość **darmowych** dodatków[1] jest zgodna na poziomie kodu źródłowego i może być wykonywana przez Octave.

Poza podstawowym zestawem poleceń języka Matlab (tu właśnie jest duża zgodność obu systemów oprogramowania) Matlab daje dostęp do sporej biblioteki (płatnych!) pakietów (*toolboxes*). Octave posiada również własną bibliotekę pakietów zwaną Octave Forge.

Scilab

Wedle podobnych ocen[2], Scilab jest zgodny w 98% z Matlabem. Jego wydajność jest mniejsza niż Matlaba czy Octave. Posiada również mniej pakietów dodatkowych.

Matlab

Istnieje możliwość skorzystania z próbnej, darmowej wersji systemu Matlab[3]. Pliki instalacyjne programu są spore ($>$ 6 GB). Wersja jest tylko na 30 dni, ale biorąc pod uwagę termin zaliczeń (grudzień) i tryb prowadzenia zajęć (15 godzin) może to być jakieś rozwiązanie.

1.3.2. Mathematica

W żadnym wypadku nie chce tweirdzić że Wolfram Language (używane przez system Mathematica) jest najprostszy czy najlepszy.

Niestety, nie ma darmowego zastępcy dla Mathematici.

Mathematica jest oprogramowaniem stosunkowo drogim. Studencka licencja kosztuje około 100 Funtów brytyjskich. Nieco tańsza jest wersja on-line (ok. 70 Funtów).

Mathematica jest oprogramowaniem bardzo dobrym i rozwijanym już od wielu lat. Pierwsza wersja pojawiła się w 1988 roku.

Jedną z metod pozyskania oprogramowania jest zakup komputerka Raspberry Pi. W standardowym zestawie oprogramowania jest darmowy pakiet Mathematici. Oczywiście możliwości samego komputera są ograniczone rozmiarem pamięci operacyjnej, ale najnowszy model daje do wyboru trzy warianty: 1 G (ok 170 zł), 2 G

[1] Największa biblioteka darmowych pakietów Matlaba dostępna jest pod adresem https://www.mathworks.com/matlabcentral/fileexchange/.

[2] https://www.yourthesisadvisor.com/matlab-vs-octave-vs-scilab-vs-freemat/

[3] https://www.mathworks.com/campaigns/products/trials.html

(220 zł) i 4 G (ok 260 zł). Nie są to ceny zawrotnie duże, choć potrzebny będzie jeszcze zasilacz, karta pamięci lub zewnętrzny dysk.

Kolejną możliwością skorzystania z oprogramowania Wolfram Mathematica jest ogłoszna inicjatywa *Free Wolfram Engine for Developers*.

Pozwala ona na skorzystanie z oprogramowania — w celach niekomercyjnych — do przygotowania oprogramowania:

— można projektować narzędzia dla siebie (lub dla firmy);
— można tworzyć osobiste projekty w domy czy w szkole/na uczelni,
— można zapoznać się z językiem.

Darmowa licencja nie pozwala opracowanych produktów udostępniać innym lub sprzedawać ich. Wymaga to wykupienia osobnej licencji. Można wykorzystać w badaniach, ale nie można publikacji „podeprzeć" opracowanym oprogramowaniem.

Warto o tym wiedzieć.

Warto też wiedzieć, że powstały ostatnio biblioteki dające dostęp do wszystkich funkcji (??)[4] języka Wolfram z poziomu języka Python. (Dla zainteresowanych — w tej chwili biblioteka **nie jest** dostępna jest w laboratorium[5].) Ale (oczywiście) trzeba mieć dostęp do samej Mathematici.

Darmowa wersja testowa oprogramowania

Istnieje również piętnastodniowa, darmowa wersja próbna[6] oprogramowania.

[4] To zapewne trzeba sprawdzić.
[5] W razie potrzeby mogę doinstalować.
[6] https://www.wolfram.com/mathematica/trial/

Część I

Teoria

2. Podstawy teoretyczne

2.1. Typy danych

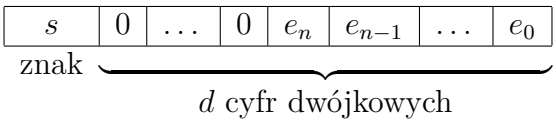
2.1.1. Dane typu całkowitego (`int`, `integer`)

Typ danych dostępny praktycznie w każdym języku programowania. Współcześnie podstawowa dana typu int zapisywana jest na 32 bitach (4 bajty) w kodzie uzupełnieniowym do dwóch. Z trzydziestu dwu bitów jeden (najbardziej znaczący[1]) wykorzystywany jest do zapamiętania znaku (zawsze 1 oznacza liczbę ujemną, a 0 — dodatnią), pozostałe służą do zapisu liczby. Dowolną liczbę całkowitą l można przedstawić w postaci rozwinięcia dwójkowego:

$$l = s \sum_{i=0}^{n} e_i 2^i \qquad (e_n \neq 0 \text{ dla } l \neq 0) \tag{2.1}$$

gdzie s jest znakiem liczby ($s = +1$ lub $s = -1$), a $e_i = 0$ lub 1 są cyframi rozwinięcia dwójkowego.

Ogólnie, gdy do zapisu liczby wykorzystujemy $d + 1$ bitów, to gdy $n < d$ — liczba l może być reprezentowana w wybranej arytmetyce. Może być zapisana jako:



W ten sposób mogą być reprezentowane liczby z zakresu $[-2^d, 2^d - 1]$. We współczesnych maszynach cyfrowych d przyjmuje wartości: 7, 15, 31, 63[2]. Zatem liczby całkowite w zależności od d mogą przyjmować wartości z różnych zakresów (tab. 2.1).

Plik nagłówkowy limits.h zawiera definicje kilku stałych, które pozwalają zawsze sprawdzić, jakie graniczne wartości mogą przyjmować zmienne określonego

[1] To ten pierwszy z lewej.

[2] Raz jeszcze przypominam, że zapis liczby jest na $d + 1$ bitach; ten dodatkowy bit, to **bit znaku**.

Tabela 2.1. Zakresy wartości w zależności od liczby bitów

d	typ (C)	zakres
7	char	$[-128, 127]$
15	short int	$[-32768, +32767]$
31	int	$[-2147483648, +2147483647]$
63	long int	$[-9223372036854775808, +9223372036854775807]$

typu. Definiują one stałe nazywajace się: SCHAR_MIN i SCHAR_MAX (minimalna i maksymalna wartość zmiennej signed char) SHRT_MIN i SHRT_MAX, INT_MIN i INT_MAX, LONG_MIN i LONG_MAX czy LLONG_MIN i LLONG_MAX. Trochę kłopotów może sprawiać typ long int który może mieć różne zakresy w zależności od tego czy komputer jest 32-bitowy czy 64-bitowy. Podczas programowania, korzystając ze stałych takich typów, nalezy pamiętać o odpowiednich przyrostkach: L — long, LL — long long czy U — unsigned, UL — unsigned long, itd.

W zasadzie wszystkie obliczenia na liczbach całkowitych dokonywane są dokładnie. Wyjątki od tej reguły są dwa:

1. Operacja dzielenia zazwyczaj nie wyprowadza poza typ całkowity. Jest to dzielenie całkowitoliczbowe („z resztą").
2. Wynik działania wykracza poza dopuszczalny zakres; podawany jest wówczas modulo 2^d.

2.1.2. Dane typu niecałkowitego

Dowolną liczbę rzeczywistą $x \neq 0$ można przedstawić w postaci

$$x = s \cdot 2^c m \tag{2.2}$$

gdzie s ($+1$ lub -1) to znak liczby, c — liczba całkowita zwana *cechą*, a m to liczba rzeczywista z przedziału $[\frac{1}{2}, 1)$ nazywana *mantysą*. Gdy $x \neq 0$ przedstawienie jest jednoznaczne.

W realizacji komputerowej cecha liczby c zapisywana jest na $d - t$ bitach, pozostałe t bitów przeznaczonych jest na reprezentację mantysy m. Zatem zamiast (na ogół nieskonczonego) rozwinięcia mantysy:

$$m = \sum_{i=1}^{\infty} e_{-i} \cdot 2^{-i} \qquad (e_{-1} = 1;\ e_i = 0 \text{ lub } 1 \text{ dla } i > 1) \tag{2.3}$$

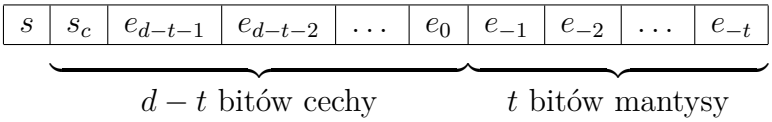
korzystamy (gdy mantysa została prawidłowo zaokrąglona do t cyfr):

$$m_t = \sum_{i=1}^{t} e_{-i} \cdot 2^{-i} \tag{2.4}$$

Wówczas

$$|m - m_t| \leq \frac{1}{2} \cdot 2^{-t} \tag{2.5}$$

Liczba x binarnie zapisywana jest jako:



Jezeli reprezentację zmiennoprzecinkową liczby x oznaczać będziemy $\mathrm{rd}(x)$ i $\mathrm{rd}(x) = s \cdot 2^c m_t$. Dla $x \neq 0$

$$\left| \frac{\mathrm{rd}(x) - x}{x} \right| \leq 2^{-t} \tag{2.6}$$

co można zapisać:

$$\mathrm{rd}(x) = x(1 + \varepsilon), \qquad \text{gdzie } |\varepsilon| \leq 2^{-t} \tag{2.7}$$

Liczby rzeczywiste reprezentowane są, na ogół, niedokładnie. Błąd względny ε jest nie większy od 2^{-t}. We współczesnych komputerach t przyjmuje wartości: 24 dla liczb typu float (32-bitowych) lub 53 (double; 64 bity).

Liczba cyfr mantysy decyduje o dokładności liczb rzeczywistych, a liczba cyfr cechy — o ich zakresie. Cecha $c \in [c_{\min}, c_{\max}]$, gdzie $c_{\min} = -c_{\max} - 1 = 2^{d-t-1}$.

Szczegóły zapisu liczb zmiennoprzecinkowych zawiera norma IEEE-754 [1].

Można zaryzykować twierdzenie że zakres liczb i ich precyzja są wystarczająco duże aby prowadzić obliczenia w miarę dokładnie, ale jak przyjrzeć się szczegółom — różnie to bywa. A wynik długotrwałych i skomplikowanych działań może być trudny do przewidzenia.

Programiści języka C mogą korzystać ze stałych zdefiniowanych w pliku nagłówkowym float .h: FLT_MIN i FLT_MAX, DBL_MIN i DBL_MAX, a nawet LDBL_MIN/LDBL_MAX dla liczb poczwórnej dokładności (long double) [2]. W przypadku liczb typu float maksymalna wartość to: 3.40282e+38.

Rozważmy prosty przykład:

Niech z będzie liczbą zespoloną $z = a + b\mathrm{i}$, $\mathrm{i} = \sqrt{-1}$. Z definicji, moduł liczby z równa się:

$$|z| = \sqrt{a \cdot a + b \cdot b}$$

W przypadu gdy wartości a i b są bardzo duże (lub bardzo małe) a^2 albo b^2 mogą nie zmieścić się w dopuszczalnym zakresie. Gdy używamy typu float , a wartość a jest większa od 1.84467341e19 — będziemy mieli kłopot. Ilustruje niższy program.

```
#include <stdio.h>
#include <math.h>
float
```

```
modul1 (float a, float b)
{
  return sqrtf (a * a + b * b);
}

float
modul3 (float a, float b)
{
  return fabs (a) * sqrtf (1 + powf (fabs (b) / fabs (a), 2));
}

float
modul2 (float a, float b)
{
  if (fabs (a) > fabs (b))
    return modul3 (a, b);
  else
    return modul3 (b, a);
}

int
main (int argc, char *argv[])
{
  float m;
  m = modul1 (2e-25f, 2e-25f);
  printf ("Bardzo male\n");
  printf ("modul1=%g\n", m);
  m = modul2 (2e-25f, 2e-25f);
  printf ("modul2=%g\n", m);
  printf ("Bardzo duze\n");
  m = modul1 (2e25f, 2e25f);
  printf ("modul1=%g\n", m);
  m = modul2 (2e25f, 2e25f);
  printf ("modul2=%g\n", m);
  return 0;
}
```

Modyfikacja polega na tym, że wzór zapisujmy w alternatywnej postaci. Niech $|a| > |b|$. Wówczas:

$$|z| = \sqrt{a^2 + b^2} = a\sqrt{1 + \left(\frac{b}{a}\right)^2}.$$

Druga metoda (funkcja modul2() daje poprawne wyniki gdy jej argumenty są bardzo małe jak i bardzo duże.

2.2. Uwarunkowanie zadania

Numeryczne uwarunkowanie zadania, to parametr pozwalający ocenić wpływ drobnych zmian parametrów wejściowych na wynik działania. Zainteresowani jesteśmy taką sytuacją, w której niewielkie zmiany (błędy) parametrów wejściowych będą powodowały niewielkie jedynie fluktuacje wyników obliczeń.

Okazuje się, że nawet w całkiem prostych przypadkach nie można liczyć na dobre uwarunkowanie problemu obliczeniowego. Co gorsza może ono zależeć od danych wejściowych.

Do oszacowania uwarunkowania obliczeniowego używa się metod bardzo podobnych do tych używanych przy określaniu stabilności układów. Po szczegóły odsyłam do [3].

Tu będą tylko przykłady.

Rozpatrzmy wielomian o postaci [3]:

$$\begin{aligned} w(x) = \prod_{i=1}^{20}(x-i) = \sum_{i=0}^{20} a_i x^i = x^{20} - 210x^{19} + 20615x^{18} - 1256850x^{17} \\ + 53327946x^{16} - 1672280820x^{15} + 40171771630x^{14} - 756111184500x^{13} \\ + 11310276995381x^{12} - 135585182899530x^{11} + 1307535010540395x^{10} \\ - 10142299865511450x^9 + 63030812099294896x^8 - 311333643161390640x^7 \\ + 1206647803780373360x^6 - 3599979517947607200x^5 \\ + 8037811822645051776x^4 - 12870931245150988800x^3 + 13803759753640704000x^2 \\ - 8752948036761600000x + 2432902008176640000 \end{aligned}$$

Pierwiastkami tego wielomianu są liczby $1, 2, \ldots, 20$.

Zaburzmy jeden ze współczynników (patrz również dodatek A). Normalnie wartość współczynnika a_{19} wynosi -210. Zastąpimy go współczynnikiem o wartości $-(210 + 2^{-23})$. Można to symbolicznie zapisać tak: $a_{19}(\varepsilon) = a_{19}(1 - \varepsilon)$, gdzie $\varepsilon = (2^{23} * 210)^{-1} < 2^{-30}$. Tak niewielka zmiana jednego tylko parametru powoduje, że wielomian ma zespolone miejsc zerowe.

Gdy niewielkie względne zmiany danych zadania powodują duże względne zmiany jego rozwiązania, to zadanie takie nazywamy źle uwarunkowanym [3].

Kolejnym problemem, który może być źle uwarunkowany jest zadanie liczenia iloczynu skalarnego dwu wektorów [3]:

$$S = \sum_{i=1}^{n} a_i b_i \neq 0.$$

Aby oszacować uwarunkowanie zadania zastąpimy dokładne wartości parametrów $\boldsymbol{a} = (a_1, a_2, \ldots, a_n)$ i $\boldsymbol{b} = (b_1, b_2, \ldots, b_n)$ przez wartości przemnożone przez $1+\alpha_i$ i $1+\beta_i$ odpowiednio. Założymy przy tym, że iloczyny $\alpha_i\beta_i$ są zaniedbywalnie małe. Aby oszacować zmienną zmianę wyniku liczymy iloraz

$$\left|\frac{\sum_{i=1}^{n} a_i(1+\alpha_i)b_i(1+\beta_i) - \sum_{i=1}^{n} a_i b_i}{\sum_{i=1}^{n} a_i b_i}\right| \approx \left|\frac{\sum_{i=1}^{n} a_i b_i(\alpha_i+\beta_i)}{\sum_{i=1}^{n} a_i b_i}\right|$$

$$\leq \max_i |\alpha_i + \beta_i| \frac{\sum_{i=1}^{n} |a_i b_i|}{\left|\sum_{i=1}^{n} a_i b_i\right|}$$

Jeżeli określimy $\mathrm{cond}(\boldsymbol{a}, \boldsymbol{b}) = \sum_{i=1}^{n} |a_i b_i| / |\sum_{i=1}^{n} a_i b_i|$ wskaźnikiem uwarunkowania (największy błąd przenosi się na wynik z takim mnożnikiem) to gdy wszystkie współczynniki mają taki sam znak — niewielki błąd ma bardzo niewielki wpływ na wynik. Gdy $\sum_{i=1}^{n} a_i b_i$ jest bliskie zeru — sytuacja jest znacznie gorsza.

2.3. Arytmetyka zmiennopozycyjna

Problem z operacjami arytmetycznymi realizowanymi przez komputery polega na tym, że „po cichu" zakładamy, że argumenty równe są swoim reprezentacjom zmiennoprzecinkowym, oraz, że podczas obliczeń, nie wystąpi ani nadmiar[3] ani niedomiar[4].

Niech $a = \mathrm{rd}(a) = 2^{c_a} m_a$ i $b = \mathrm{rd}(b) = 2^{c_b} m_b$ będą argumentami. Przyjmijmy, że $a \geq b > 0$. Ich (dokładna) suma wynosi:

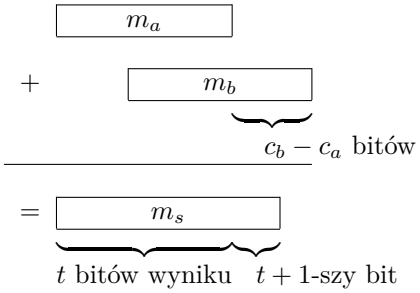
$$a + b = 2^{c_a}(m a_a + m_b 2^{-(c_a - c_b)}) = 2^{c_a} m_s \tag{2.8}$$

Mantysa wyniku wyliczana jest przez dodanie do mantysy liczby a odpowiednio przekształconej (to znaczy tak, żeby cechy liczb a i b były jednakowe) mantysy liczby b.

Wszystko to wygląda jakoś tak:

[3] Nadmiar to sytuacja taka, gdy wynik operacji jest większy niż może być zapamiętany w formie komputerowej (całkowitej lub zmiennoprzecinkowej).

[4] Niedomiar występuje wtedy gdy wyniki jest mniejszy niż najmniejsza liczba jaką można w formie zmiennoprzecinkowej zapisać.



Jakość wyniku zależy od możliwości poprawnego zaokrąglenia wyniku. A zatem od liczby bitów, na których zapisywany jest wynik. Gdy jest ich tylko t — nie jest dobrze. Historycznie, liczba **dodatkowych** bitów **wewnętrznego** rejestru arytmometru zmieniała się od zera aż do t (wynikowy rejestr podwójnej długości). Dopiero norma IEEE-754 [1] wprowadziła „obowiązek" korzystania z dodatkowego bitu.

Jeżeli symbolem fl oznaczać będziemy realizację działania w arytmetyce zmiennoprzecinkowej, to:

$$\mathrm{fl}(a+b) = \mathrm{rd}(a+b)$$

Ogólnie, dla dowolnej operacji $\diamond$

$$\mathrm{fl}(a \diamond b) = \mathrm{rd}(a \diamond b) = (a \diamond b)(1+\varepsilon), \quad \varepsilon = \varepsilon(a,b,\diamond), \quad |\varepsilon| \leq 2^{-t} \tag{2.9}$$

uzyskany (komputerowo) wynik, nieznacznie, różni się od wyniku „prawdziwego".

Wydaje się, że niewielkie ($2^{-t} \in [10^{-15}, 10^{-6}]$) względne błędy nie powinny mieć wielkiego wpływu na wyniki. Okazuje się, że jest inaczej, a sposób realizacji obliczeń może mieć ogromny wpływ na wyniki.

Rozpatrzmy dwa różne algorytmy wyliczania wartości różnicy kwadratów dwu liczb:

— $A1(a^2 - b^2) = a^2 - b^2$

— $A2(a^2 - b^2) = (a-b)(a+b)$

$$\begin{aligned}\mathrm{fl}(a^2 - b^2) &= (a \times a(1+\varepsilon_1) - b \times b(1+\varepsilon_2))(1+\varepsilon_3) \\ &= (a^2 - b^2)\left(1 + \frac{a^2\varepsilon 1 - b^2\varepsilon_2}{a^2 - b^2}\right)(1+\varepsilon 3) \\ &= (a^2 - b^2)(1+\delta)\end{aligned}$$

W przypadku gdy a^2 jest bliskie b^2, a ε_1 i ε_2 mają przeciwne znaki — błąd względny δ moze być dowolnie duży.

W przypadku drugiego algorytmu:

$$\begin{aligned}\mathrm{fl}((a-b)(a+b)) &= ((a-b)(1+\varepsilon_1)(a+b)(1+\varepsilon_2))(1+\varepsilon_3) \\ &= (a^2 - b^2)(1+\delta)\end{aligned}$$

gdzie $\delta \leq |\varepsilon_1| + |\varepsilon_2| + |\varepsilon_3|$, zatem δ jest zawsze mniejszy od $3 \cdot 2^{-t}$.

W szczególności, ze względu na algorytm dodawania (2.8) gdy a i b są takie, że $|b| \leq \frac{1}{2}2^{-t}|a|$, zachodzi zależność:

$$\mathrm{fl}(a+b) \equiv a.$$

(Wspominałam o tym na zajęciach z Technologii Informacyjnych.)

Konsekwencje tego mogą być bardzo poważne w przypadku numerycznego aproksymowania ilorazem różnicowym, pochodnej funkcji w punkcie. Zamiast otrzymywania coraz to lepszego przybliżenia pochodnej (gdy różnica wartości pomiędzy dwiema wartościami zmiennej niezależnej) dostaniemy wartości nie mające nic wspólnego z pochodną.

2.4. Algorytmy numerycznie poprawne

Korzystając z komputera do wszelkich obliczeń najistotniejszą sprawą staje się kwestia jak zachowywać się będą algorytmy rozwiązywania problemów: dane które dostarczamy nigdy nie są dokładne — są reprezentacją zmiennoprzecinkową danych. Również rozwiązanie — nawet jeżeli dokładne — będzie tylko reprezentacją zmiennoprzecinkową rozwiązania.

Z powyższych względów *za numerycznie najwyższej jakości uznamy takie algorytmy, dla których obliczone rozwiązanie jest nieco zaburzonym rozwiązaniem (dokładnym) zadania o nieco zaburzonych danych.* Algorytmy spełniające powyższy postulat nazywamy *numerycznie poprawnymi.*

Dokładniej problemu nie będę analizował odsyłając do literatury [3].

2.5. Algorytmy numerycznie stabilne

Kolejnym ważnym problemem jest badanie stabilności algorytmów obliczeniowych. Algorytm numerycznie stabilny to taki, który gwarantuje rozwiązania (dla dowolnego zadania rozważnej klasy) z błędem tego samego rzędu co optymalny poziom błędu rozwiązania danego zadania. Oznacza to tyle, że błąd w algorytmie numerycznie stabilnym jest na poziomie „nieuniknionego" błędu wynikającego z przybliżonej reprezentacji danych i wyniku [3].

2.6. Arytmetyka „wyższej" precyzji

Pewnym rozwiązaniem problemów związanych z ograniczeniami wynikającymi ze skończonej liczby bitów przeznaczonych na zapis liczb może być arytmetyka wielokrotnej precyzji (ang. *Multiple Precision Arithmetic*). GNU Multiple Precision

Arithmetic Library[5] (lub GMP) jest jedną z bardziej popularnych implementacji tej idei.

Biblioteka pozwala prowadzić obliczenia na liczbach całkowitych, zmiennoprzecinkowych oraz — standardowy język C na to nie pozwala — wymiernych [4].

Język programowania Python nie wprowadza, praktycznie, żadnych ograniczeń na liczby całkowite, pozwalając uniknąć ograniczenia zakresu związane ze standardowymi sposobami zapamiętywania liczb. Standardowo, działania na liczbach zmiennoprzecinkowych prowadzone są w arytmetyce 64-bitowej. Istnieje też implementacja biblioteki GMP dla Pythona (gmpy). Pakiet mpmath korzysta z gmpy.

Mathematica pozwala deklarować precyzję prowadzonych obliczeń (patrz rozdział 10).

Matlab podobne możliwości ma w pakiecie Symbolic Math Toolbox (bardzo ograniczona liczba licencji) lub pakiecie Multiple Precision Toolbox for MATLAB.

[5] https://gmplib.org/

3. Generatcja danych

Główną wadą prowadzenia obliczeń numerycznych jest to, że nie zawsze wiadomo jaki powinien być wynik. W związku z tym stosuje się bardzo prostą metodę: Jako danych wejściowych do algorytmu używa się danych „syntetycznych". Wygenerowanych w sposób algorytmiczny i takich, dla których można przewidzieć wynik. Bardzo często dane są zaburzane. Najczęściej zaburzenie jest generowane (pseudo)losowo i znamy jego rozkład, średnią (ta, bardzo często, równa jest zero) i wariancję.

Bardzo często będziemy potrzebowali danych w formie:

$$y_i = f(x_i) + \zeta_i, \quad i = 1, 2, \ldots, N; \quad \zeta_i \text{ — losowe} \tag{3.1}$$

Zazwyczaj (albo najprościej) $x_i = x_0 + i\Delta$, gdzie $\Delta = (x_N - x_0)/N$ (czasami dopuszczamy, żeby i zaczynało się od zera).

Aby wygenerować wartość losową — należy skorzystać z generatora liczb pseudolosowych dostarczanych przez używany język programowania. W przypadku C może to być GSL — GNU Scientific Library[1]. Zarówno matlab jak i Mathematica mają swoje własne biblioteki.

3.1. Mathematica

1. Generacja $x_i = x_0 + i\Delta$. Można w tym celu użyć polecenia **Table**.:

   $$\text{data1} = \text{Table}[x, \{x, x_0, x_N, \Delta\}]$$

   na przykład:

   $$\text{data1} = \text{Table}[i, \{i, 0, 10, 0.1\}]$$

   Zmienna (złożona) data1 zawierać będzie 101 wartości z zakresu od 0 do 10.
2. Generacja $f(x)$.
   — najpierw definiujemy funkcję; W języku Mathematici nie jest to specjalnie skomplikowane:

   $$f(\text{x\_}) := x + \sin(x)$$

   (zwracam uwagę na zapis $x\_$; oznacza on, że x jest zmienną niezależną funkcji oraz symbol := oznaczajacy definicję, a nie podstawienie). Cechą

[1] http://www.gnu.org/software/gsl/.

charakterystyczną funkcji jest to, że mogą działać ona na zmiennych (stałych) prostych:
f[x_]:=Sin[x] + x
f[4]
4 + Sin[4]
(mathematica, tak długo jak się da liczy symbolicznie)
N[f[4]]
3.2432
jak i złożonych:
N[f[{3, 4}]]
{3.14112, 3.2432}

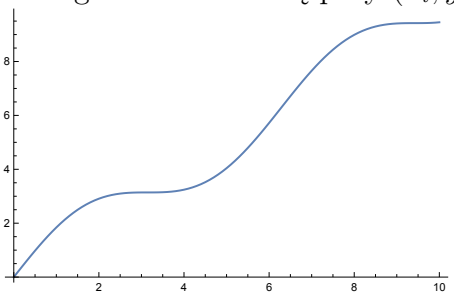
— następnie nakładamy funkcję na nasze dane:

$$\text{data2} = f(\text{data1})$$

— możemy (przy okazji) zobaczyć wykres:

$$\text{ListLinePlot}\left[\{\text{data1}, \text{data2}\}^{\mathsf{T}}\right]$$

(symbol $^{\mathsf{T}}$ oznacza transpozycję; transpozycja jest potrzebna ze względu formę danych dla polecenie ListLinePlot; macierz utworzona poleceniem {data1, data2} to dwie kolumny x-ów i y-ków. ListLinePlot wymaga wektora, którego elementami są pary (x_i, y_i); transpozycja to załatwia.)



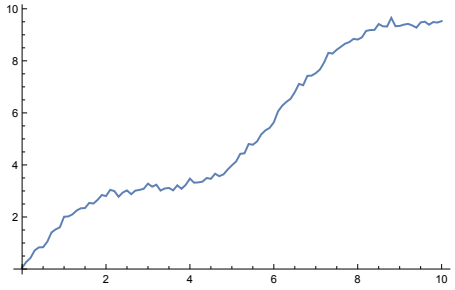
3. Wreszcie możemy nałożyć zaburzenia. Zazwyczaj o zaburzeniach będziemy zakładać że mają rozkład normalny[2] o średniej zero i niezbyt wielkim odchyleniu standardowym. Funkcja RandomVariate pozwala wygenerować zadaną liczbę danych o wskazanym rozkładzie. Pierwszym argumentem funkcji jest rodzaj rozkładu, drugim — ilość liczb.

$$\text{data3} = \text{RandomVariate}[\text{NormalDistribution}[], 101]$$

(brak parametrów oznacza średną zero i wariancję 1, rozkładu normalnego). Łatwo zgadnąć, że data2 + data3 to będzie wektor zaburzonych wartości naszej

[2] Jest to typowe zaburzenie z jakim możemy mieć do czynienia, na przykład, podczas pomiaru.

funkcji, czyli $f(x) + \zeta$. Poniżej wykres, przy czym dane zostały zaburzone zmienną o rozkłądzie $N(0, 0.1)$:



3.2. Matlab

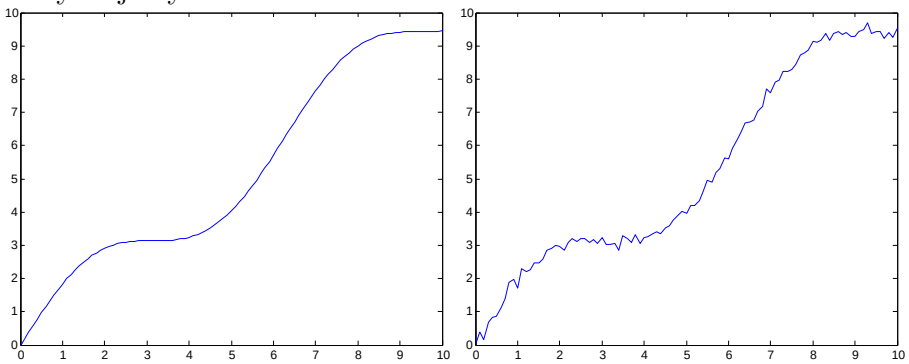
W matlabie wszystko praktycznie tak samo, tylko (chyba) nieco prościej:

1. Generacja $x_i = x_0 + i\Delta$. Można w tym celu użyć operatora „:” w formie Start:przyrost:Koniec:
   `data1=0:0.1:10;`
   (gdy nie dodamy średnika, matlab wypisze wszystkie wartości.
2. Teraz funkcja. Ze zdefiniowaniem funkcji jest pewien kłopot. Funkcje powinny znajdować się w plikach zewnętrznych, ale można spróbować czegoś takiego[3]:
   `f=@(x) sin(x)+x;`
   Aby sprawdzić czy działa piszemy `f(data1)` albo `data2=f(data1)`.
3. Generacja liczb losowych odbywa się za pomocą funkcji random. Wywołanie jej jest proste:
   `data3 = random(name,A,B,...)`
   gdzie (name) to nazwa rozkładu (dla rozkładu normalnego będzie to **norm** albo **Normal**), A, B,... to parametry rozkładu. Parametr może być jeden, może nie być go wcale, w przypadku rozkładu normalnego parametry powinny być dwa: średnia rozkładu i wariancja. Uwaga: nazwę rozkładu podajemy jako tekst.
   `random('Normal', 0.0, 1.0)`

   wygeneruje jedną wartość. Na końcu wszystkich parametrów może pojawić się dodatkowy parametr mówiący ile wartości generować (czy raczej jaką tablicę wartościami wypełnić).
   `data3 = random('Normal', 0.0, 0.1, [1,101])`
   Żeby zobaczyć dane oryginalne i zaburzone (data2+data3) używamy polecenia plot:
   `plot(data1, data2)`
   `plot(data1, data2+data3)`

[3] Taka konstrukcja nazywa się *funkcją anonimową*.

i uzyskujemy:



3.3. AI: Odrobina teorii

AI to oczywiście *Artificial Intelligence* czyli Sztuczna Inteligencja.

Natomiast następujących zapisów nie należy w żadnym wypadku traktować jako jakiejkolwiek uporządkowanej informacji na temat sztucznej inteligencji. Umieszczam tu kilka bardzo ogólnych i zupełnie przypadkowych uwag.

Tematy sztucznej inteligencji czy uczenia maszynowego czy *big data* (wszystkie jakoś ze sobą powiązane) odgrywają dziś coraz większą rolę. Stąd nieśmiały pomysł, aby praktycznie (oraz szybko i łatwo) pokazać proste zadania z tym związane. Natomiast tematyka jest bardzo szeroka i warta podjęcia szerszych studiów (na przykład na II stopniu?).

Pod względem matematycznym, na podstawy sztucznej inteligencji składają się:

- algebra liniowa:
  - przestrzenie wektorowe,
  - tensory,
  - przekształcenia liniowe,
  - podprzestrzenie
  - teoria macierzy,
  - wartości i wektory własne,
- analiza matematyczna:
  - granice, ciągłość, nieciągłość,...
  - pochodne,
  - całki,
  - podstawowe twierdzenia rachunku całkowego,
  - ...
- rachunek prawdopodobieństwa:
  - prawdopodobieństwa warunkowe,
  - „prawo wielkich liczb”,
  - statystyka bayesowska,

  - błądzenie losowe (np. łańcuchy Markowa)
  - ...
- optymalizacja:
  - programowanie liniowe,
  - programowanie kwadratowe,
  - zagadnienia kombinatoryczne,
  - ...
- heurystyka:
  - algorytmy genetyczne,
  - ewolucja różniczkowa (*Differential evolution*),
- metody iteracyjne,
  - metoda Newtona,
  - metody gradientowe,
  - metody gradientów sprzężonych
  - interpolacja
- ...

3.3.1. Klasyfikacja

Jednym z najważniejszych zadań sztucznej inteligencji jest klasyfikacja, czyli taki problem decyzyjny, który odpowiada do jakiej kategorii należy badany obiekt. Jednym z przykładów jest rozpoznanie rodzaju schorzenia na podstawie objawów. Dodać trzeba, że bardzo często algorytmy sztucznej inteligencji sprawdzają się tu bardzo dobrze.

Podstawowe metody stosowane przez algorytmy klasyfikacji to:

- „naiwny Bayes" — metoda oparta na twierdzeniu Bayesa o prawdopodobieństwie warunkowym,
- budowa hiperpłaszczyzn (hiperpowierzchni) rozgraniczających w przestrzeni cech,
- k-Najbliższych Sąsiadów — metoda polegająca na zmierzeniu odległości wektora cech klasyfikowanego obiektu od sklasyfikowanych i wyboru tej kategorii, w której znajduje się k najbliższych obiektów.
- drzewa decyzyjne — rozpatruje się sekwencyjnie kolejne cechy klasyfikowanego obiektu i na tej podstawie dokonuje ostatecznej klasyfikacji,
- regresja logistyczna,
- ...

Zazwyczaj klasyfikacja wymaga ciągu uczącego, czyli zestawu obiektów, które są wstępnie sklasyfikowane. Na tej podstawie algorytm „dostraja się taki sposób, żeby jego decyzje były zgodne z zadeklarowanymi klasami obiektów. Bardzo często uczenie się ma kolejny etap, czyli sprawdzian: działanie algorytmu testowane jest

na kolejnych preklasyfikowanych obiektach. Jeżeli wyniki nie są zadowalające — dostarcza się kolejny ciąg uczący.

Stąd przyjęła się nazwa „Uczenie maszynowe" lub *Machine Learning*.

Pamiętać trzeba, że nigdy decyzje klasyfikatora nie są w 100% poprawne.

W przypadku klasyfikacji binarnej można stworzyć „tablicę pomyłek" (*confusion matrix*). Niech nasz algorytm służy do decydowania czy pacjent jest chory czy zdrowy. Możemy mieć do czynienia z czterema sytuacjami:

1. Pacjent jest zdrowy i tak jest klasyfikowany przez algorytm; decyzja jest **prawdziwie pozytywna** czyli TP[4].
2. Pacjent jest zdrowy, ale klasyfikowany jako chory — **fałszywie negatywnie** czyli FN.
3. Pacjent jest chory, ale klasyfikowany jako zdrowy —**fałszywie pozytywnie**, FP,
4. Chory pacjent jest klasyfikowany jako chory — **prawdziwie negatywnie**, TN.

Przypadki klasyfikacji fałszywie pozytywnej określane są również mianem błędu pierwszego rodzaju, a przypadki fałszywie negatywne — błędy drugiego rodzaju.

Czasami używa się dodtkowych parametrów: czułość (TPR — *True Positive Rate*), swoistość (TNR — *True Negative Rate*), precyzja (PPV *Positive Predictive Value*) i dokładność (ACC *Accuracy*). Określone są one wzorami:

— TPR:

$$TPR = TP/(TP + FN)$$

— TNR:

$$TNR = TN/(FP + TN)$$

— PPV:

$$PPV = TP/(TP + FP)$$

— ACC:

$$ACC = (TP + TN)/(P + N)$$

gdzie: $P = TP + FN$, a $N = TN + FP$; $P + N$ to wielkość testowanej populacji.

3.3.2. Przykład

Poniższy przykład został wybrany z dokumentacji.

Do klasy $A = \{1, 2\}$ należą dwa elementy, do klasy $B = \{3, 4\}$ kolejne dwa. zbiory są rozłączne i klasyfikacja wydaje się prosta: wszystko co mniejsze od 2,5 to A, a pozostałe — B.

$$\mathbf{TrainingSet = \{1 \rightarrow A, 2 \rightarrow A, 3 \rightarrow B, 4 \rightarrow B\};}$$

[4] True Positive

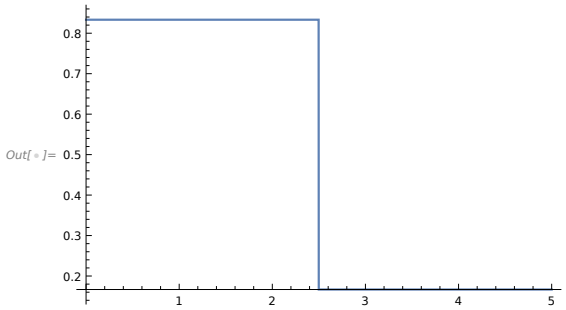
c = Classify[TrainingSet]

ClassifierFunction [□]

c[2.5, "Probabilities"]

A → 0.166667, B → 0.833333

Plot[c[x, "Probability" → A], {x, 0, 5}]



Gdy sytuację skomplikujemy i $A = \{1, 2, 3{,}1\}$, a $B = \{3, 3{,}15, 4, 5\}$ to zbiory znowu, są rozłączne, ale przedziały liczbowe „zachodzą" na siebie. Klasyfikacja nie jest już taka prosta.

TrainingSet = {1 → A, 2 → A, 3.1 → A, 3 → B, 3.15 → B, 4 → B, 5 → B};

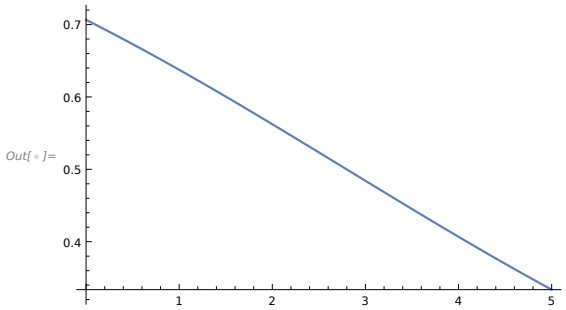
c = Classify[TrainingSet]

ClassifierFunction [□]

c[2.5, "Probabilities"]

A → 0.523397, B → 0.476603

Plot[c[x, "Probability" → A], {x, 0, 5}]



***c*[3]**

B

Co ciekawe, w pierwszym przypadku użyty klasyfikator to **Nearest Neighbors** (najbliżsi sąsiedzi), a w drugim **regresja logistyczna**.

3.3.3. „Odwrotna klasyfikacja”

Pod pojęciem „odwrotnej klasyfikacji” rozumiem zadanie stworzenia obiektu, który będzie miał określone cechy. Takie zadanie już bardziej nadaje się do określenia jako „sztuczna inteligencja” bo algorytm tworzy (na przykład obraz obiektu) mający pożądane cechy. Technologia nazywa się GAN (*Generative Adversarial Network*) i ostatnio nie schodzi z łam internetowych dzienników. Pojawiły się zestawy sztucznie generowanych twarzy, sztucznie generowanych kotków, a nawet obrazki ptaków (i kwiatów) generowane na podstawie słownego opisu („ptak z białym brzuszkiem i zółtym ogonem”) [5].

3.3.4. Lektury

Wydaje się, że zacząć można od Tadeusiewicza [6] i [7]. Autor jest niewątpliwie autorytetem w tym zakresie. Po polsku dostępnych jest kilka prac: [8], [9], [10]. Tematyka jest tak gorąca, że można napotkać sporo obszernych źródeł (udostępnianych, na przykład, przed publikacją) lub jako materiały pomocnicze do kursów: [11], [12], [13]. Ciekawe jest też obszerne zetsawienie zasobów [14].

3.3.5. Uwagi

Sugeruję dokładne zapoznanie się z dokumentacją polecenia **Classify** dostępną bądź na stronach producenta lub — lepiej — w czasie zajęć, po otwarciu notatnika Mathematici. Dokumentacja jest interaktywna i można polecenia w helpach modyfikować i patrzeć na efekty lub kopiować do notatnika. Warto zapoznać się z całym rozdziałem dotyczącym uczenia maszynowego [15].

Polecenie Classify automatycznie tworzy procedurę (funkcję), która podany element klasyfikuje do kategorii.

Dostępne metody klasyfikacji to:
— "DecisionTree"
— “GradientBoostedTrees”
— “LogisticRegression” ’
— “Markov”
— “NaiveBayes”
— “NearestNeighbors”
— “NeuralNetwork”
— “RandomForest”
— “SupportVectorMachine”

Polecenie Classify może wybierać metodę klasyfikacji tak, aby optymalizować jedno z kryteriów:
— użycie pamięci,
— jakość klasyfikacji,
— szybkość klasyfikacji
— szybkość nauki
— może też wybrać kryterum automatycznie, lub użyć wszystkich danych jako danych uczących.

Dodatkowo, oprogramowanie ma już kilka wcześniej przygotowanych funkcji klasyfikujących, pozwalających na rozpoznawanie:
— flag,
— wieku na podstawie twarzy,
— płci na podstawie twarzy,
— języku w jakim napisany jest tekst (lub program)
— ...

3.3.6. Narzędzia

Nie ma **najlepszych** narzędzi do rozwiązywania żadnego problemu. Natomiast obliczenia na potrzeby sztucznej inteligencji prowadzone są/mogą być w różnych językach. Do najpopularniejszych należą:
— Python,
— R,
— Matlab,
— Wolfram Language,
— Julia,
— Prolog
— Lisp
— Java

— . . .

Część II

Dane

4. Zdobywanie danych

Rozdział ten poświęcony będzie „niekonwencjonalnym" metodom pozyskiwania danych. Myśląc o „niekonwencjonalnych metodach" nie mam na myśli kradzieży.

Konwencjonalna metoda pozyskiwania danych to uzyskanie ich w jakimś pliku, gdzie będą one wpisane w takim lub innym (standardowym, lub nie) formacie.

4.1. CSV

CSV czyli *Comma separated values* (dane oddzielone przecinkiem).

Najbardziej „standardowy" format danych tekstowych.

Liczby prezentowane są jako wartości dziesiętne, do oddzielenia części ułamkowej używana jest **kropka dziesiętna**. Generalnie jest kłopot z krajami (regionami), w których część ułamkowa od całkowitej oddzielona jest przecinkiem. Wówczas format musi być uogólniony: Zamiast przecinka oddzielającego kolejne wartości używany jest średnik. Liczby zapisywane są z **przecinkiem dziesiętnym**.

Stąd większość **dobrych** programów rozumiejących ten format pozwala zdefiniować znak rozdzielajacy poszczególne pola.

Kolejny problem pojawia się, gdy pole tekstowe zawiera znak, który powinien być traktowany jako jako znak rozdzielający pola. Wówczas całe pole musi być ujęte w cudzysłowy.

Osobnym problemem są programy typu arkusz kalkulacyjny (Excel, LibreOffice Calc,...). Są one na ogół „zlokalizowane"[1]. Stąd normalny, „polski" Excel nie rozpoznaje wartości typu 1.3 (jeden i trzy dziesiąte). Rozpoznaje za to 1,3... I tak będzie wartości eksportował do pliku CSV (oddzielając pola znakami średnik).

Warto wiedzieć (i pamiętać), że w pliku xlsx (podobnie jak i ods)[2] wszystkie wartości pamiętane są jako liczby dziesiętne zapisane **tekstowo** z kropką dziesiętną... Zawartość komórki `=1/3` (formuła) pamiętana jest w dwu formatach: jako tekst („zapis" formuły) i jako wartość (0.333333333333333). Gdy skopiujemy

[1] Tym okropnym terminem będącym kalką z języka angielskiego (od *locale*) określa się proces tworzenia przystosowanej do danego języka i lokalnych zwyczajów wersji programu, czyli sposób zapisu liczb niecałkowitych, słownik używany do określania poprawności językowej, znak waluty,...

[2] Oba formaty (Excel i Calc) to skompresowane katalogi zawierające tekstowe pliki XML.

zawartość komórki w trybie specjalnym (tylko liczby) to w pliku pamiętane będą dwie formy:

— wyświetlana (0,333333333333333),
— wartość (0.333333333333333).

Może to mieć jakiś wpływ na dokładność obliczeń...

Dobra wiadomość jest taka, że zachowanie tych programów można zmienić (prosząc by wyświetlał wartości dziesiętne z kropką).

4.2. Matlab

Matlab ma możliwość eksportu danych do postaci tekstowej, ale również do postaci binarnej. Ta ostatnie nie jest „przenośna" (inne programy będą miały kłopoty z jej odczytaniem).

Matlab nie akceptuje wszystkich formatów danych wynikających z lokalizacji, w większości wypadków traktuje dane przyjmuje jako *liczby dziesiętne z kropką*; oczekuje tekstu w formacie ASCII (to znaczy bez znaków diakrytycznych).

Nie widzę możliwości zmiany sposobu traktowania przecinka. Dokumentacja mówi, że w systemach unixowych Matlab **zawsze** oczekuje, że separatorem dziesiętnym jest kropka. W przypadku systemów windowsowych informacja jest enigmatyczna: *Matlab nie obsługuje każdej lokalizacji; w takim wypadku użyta zostanie lokalizacja C.*

4.3. Mathematica

4.4. Import danych z tablicy w pliku PDF

Często dostępne dane udostępnione są w pliku PDF. W zasadzie stosunkowo łatwo jest skopiować zawartość takiego pliku, ale czasami pliki PDF są dosyć nieprzyjazne. Czasmi ciężko skopiować wiersz, a skopiowanie kolumny z pliku PDF jest właściwie niemożliwe.

Aplikacja tabula pomaga rozwiązać te problemy.

Oprogramowanie (aplikacja napisana w Java) po uruchomieniu tworzy serwer z bardzo wygodnym interfejsem dostępnym przez dowolną przeglądarkę WWW. Pozwala załadować plik PDF, a następnie wskazać tabele (fragmenty tabel) które chcemy z pliku wyciągnąć i skonwertować do formatu CSV.

Program jest bardzo dobrze udokumentowany i bardzo łatwy w obsłudze.

4.5. Import danych z wykresu

Jeszcze gorszą sytuacją jest, gdy posiadamy jedynie dostęp do wykresu danych. W takiej sytuacji wydzwać się może, że sytuacja jest beznadziejna...

Bardzo wygodnym narzędziem może być napisana w Javie aplikacja WebPlotDigitizer. Pozwala ona na wczytanie pliku graficznego zawierającego wykres, a następnie w sposób automatyczny na zdigitalizowanie wykresu. Łatwo można zebrane punkty uzupełnić w sposób ręczny.

Program posiada dosyć bogatą dokumentację oraz przydatny tutorial na YouTube.

Dane zapisywane są w pliku CSV.

Część III

Instrukcje laboratoryjne

5. Uszczegółowienie instrukcji laboratoryjnych ~~na czas zarazy~~

Uwaga: Informacje w tym rozdziale będą na bieżąco uzupełniane!

5.1. Wprowadzenie

Po pierwsze, **nie da się** literalnie zrealizować wszystkich instrukcji laboratoryjnych znajdujących się na stronach zajęć.

Po drugie, instrukcji laboratoryjnych jest 7, a zajęć mamy tylko 5. Trzeba coś wybrać.

Każde zajęcia składać się będą z trzech części:

1. Prezentacja wybranych problemów (ok 45 minut).
2. Dyskusja problemów.
3. Praca własna i przygotowanie sprawozdania.

Przez cały czas zajęć jestem dostępny (e-mail, w miarę potrzeby Zoom). W razie jakichkolwiek wątpliwości można się kontaktować w dowolnym terminie (ale wówczas nie można liczyć na bardzo szybką reakcję).

Umawiamy się, że sprawozdania dostarczone powinny być (via e-mail) do następnych zajęć ~~(nie dotyczy to pierwszych zajęć w tygodniu parzystym, które odbyły się 9 października),~~ nie później jednak niż do końca jedenastego tygodnia semestru (zaliczenia, prace dyplomowe, etc)[1].

Cały czas podczas trwania zajęć pozostaję do dyspozycji (e-mail, zoom).

5.2. Laboratorium 1

~~Ze względu na brak oprogramowania~~[2] do zrealizowania jest wersja ~~minimalistyczne,~~ sprowadzająca się do tego:

[1] Niespełnienie tego warunku wiązać się może z problemami formalnymi.

[2] Nie mogę założyć, że macie dostęp do czegokolwiek.

1. wybieramy interesujący[3] nas zbiór danych;
2. prezentujemy dane z tego zbioru korzystając z **jakiegokolwiek** dostępnego nam narzędzia (może to być nawet program typu arkusz kalkulacyjny).

Sprawozdanie (niezbyt długie) powinno zawierać uzyskane wyniki i zapowiedź (albo jakieś rozważania dotyczące) używanego podczas reszty zajęć narzędzia/pakietu oprogramowania.

Za „interesujące dane'' można uznać, na przykład, porównanie przebiegu epidemii w jakimś kraju (Polsce) w tym samym okresie w roku 2020 i 2021. Dla Polski ciekawe daty, to od pierwszego[4] września do końca roku.

5.3. Laboratorium 2

Laboratorium dotyczy generalnie interpolacji danych. W trakcie zajęć zostaną przedstawione prezentacje pokazujące możliwość:

— Mathematici,
— Notatników Pythona (jupyter notebook).

Uwagi:

— Excel pozwala chyba na realizację interpolacji liniowej...
— Wydaje mi się, że cokolwiek bardziej skomplikowanego trzeba zaprogramować na podstawie wzorów (lub używając VB).

5.3.1. Zadania do wykonania

1. Wybrać jakieś źródło danych.
2. Wyselekcjonować kilkanaście punktów z danych.
3. Zaprezentować na tych punktach kilka rodzajów interpolacji (schodkowa, liniowa, kwadratowa,..., spline).
4. Zdecydować, która z nich najlepiej opisuje „rzeczywistość".
5. Napisać sprawozdanie (nieco dłuższe: dwie–trzy strony?). Powinno zaczynać się od wyjaśnienia (własnymi słowami) co to jest **interpolacja**.

5.4. Laboratorium 3

Będą demonstracje w Jupyterze i Mathematici z objaśnieniami.

[3] Instrukcja laboratoryjna **numer 1** odsyła do dwu zestawów danych „meteorologicznych". Można również poszukać jakiegoś innego zestawu danych (COVID, dane demograficzne, wyniki słuchalności stacji radiowych,...).

[4] Pierwszego września rozpoczął się rok szkolny.

Podstawą zajęć jest instrukcja laboratoryjna 2A. Zajęcia poświęcone są FFT[5] i są **obowiązkowe**[6].

Sprawozdanie, w szczególności powinno zawierać (punkt 4 Zadań do wykonania):

> Opisać od czego zależy rozdzielczość (zdolność do identyfikowania w przebiegu złożonym składowych o bardzo bliskich częstotliwościach) szybkiej transformaty Fouriera. Można posiłkować się notatnikiem Jupyter.

Dodatkowo powinny być opisane **Wszystkie** zależności między czasem próbkowania, liczbą próbek (częstością próbkowania) a rozdzielczością i zakresem rozpoznawanych przez FFT częstości (twierdzenie o próbkowaniu).

Problemy można zilustrować przykładami.

Laboratorium (teoretycznie[7]) można próbować zrealizować w Excelu. Potrzebny jest dodatkowy pakiet *Analysis ToolPak*.

5.4.1. Zadania do wykonania

1. Wybrać jakikolwiek zestaw danych, o którym można podejrzewać, że ma charakter „okresowy”[8].
2. Wyliczyć transformatę Fouriera danych.
3. Pokazać „piki” związane z podejrzewanym okresem.
4. Wyliczyć wartość okresu.
5. Dodatkowo należy opisać wszystko co wynika z twierdzenia Shannona-Kotielnikowa, oraz zależności między czasem próbkowania, okresem próbkowania a rozdzielczością FFT, zakresem częstotliwości i inne takie...

Hint:

— Można podejrzewać, że dane meteorologiczne (na przykład temperatura) zmieniają się w sposób okresowy (cieplej w południe, chłodniej w nocy).

— Bardzo często w czasie pełni księżyca jest wyż. Zbadanie tego wymagałoby znajomości zmian ciśnienia w ciągu roku. Nie wiem na ile trudno jest pozyskać takie dane. Ale patrz również tu:

5.5. Pozostałe laboratoria

Jeżeli chodzi o ostatnie dwa sprawozdania, sugeruję, że powinniście mieć Państwo możliwość wykazania się inwencją. W końcu powinienem jakoś wszystkich ocenić.

[5] FFT to właśnie interpolacja trygonometryczna.

[6] Ponieważ instrukcji laboratoryjnych jest więcej niż zajęć — studenci mają pewną dowolność w realizacji instrukcji laboratoryjnych. Natomiast zajęcia FFT są obowiązkowe.

[7] Bo nigdy tego nie próbowałem robić...

[8] Cokolwiek to znaczy.

Pozostały wśród instrukcji laboratoryjnych następujące tematy:

1. Aproksymacja (aka regresja)
2. Błędy obliczeń
3. Optymalizacja
4. Sztuczna inteligencja

Błędy obliczeń i **Sztuczna inteligencja** mogą być trudne do realizacji w warunkach domowych (ale, oczywiście, są to tematy dosyć interesujące). Zamierzam te tematy zademonstrować na spotkaniach.

5.5.1. Aproksymacja

Instrukcja Laboratoryjna 3.

Tu zadanie może polegać na próbie aproksymacji dziennego przebiegu temperatury za pomocą jakiejś funkcji aproksymującej. Inaczej niż w przypadku interpolacji nie możemy tu liczyć na to, że funkcja aproksymacyjna przejdzie przez jakikolwiek z punktów pomiarowych. Chodzi jedynie o jak najlepsze „oddanie charakteru zmienności" przebiegu.

Stworzyłem notatnik jupytera pokazujący jak to może wyglądać.

Kolejny notatnik (generalnie poświęcony aproksymacji liniowej) pokazuje w jaki sposób można wykorzystać aproksymację liniową do wyznaczania parametrów funkcji aproksymującej o postaci

$$y = a_0 + \sum_{i=1}^{N} a_i f_i(x)$$

gdzie f_i to zestaw zadanych funkcji, przez odpowiednie komponowanie „macierzy pomiarów"

Zadanie aproksymacji ma największy sens wtedy, gdy znamy (z teorii) opis jakiegoś zjawiska z dokładnością do kilku (kilkunastu) parametrów. Zadanie wówczas sprowadza się do takiego dobrania parametrów aby jak najlepiej przybliżyć opisywane zjawisko.

Oczywiście wielomian prawie nigdy nie opisuje żadnego rzeczywistego zjawiska. Ale używając wielomianu stosunkowo łatwo uzyskać żądaną dokładność (zwiększając jego stopień). Zwracam uwagę (przypominam), że jeżeli mamy N punktów to wielomian stopnia $N-1$ będzie funkcją interpolacyjną (przechodzącą przez wszystkie punkty).

5.5.2. Optymalizacja

Instrukcja Laboratoryjna 5.

Tu, generalnie nie mam wiele do dodania. Przygotowałem dodatkowy tekst na temat optymalizacji. Zawiera on informacje ogólne oraz odsyłacze do szczegółowych

notatników jupytera pozwalających „potrenować“ minimalizowanie funkcji liniowych (z ograniczeniami) oraz oraz coś na temat zadań optymalizacji gdzie żąda się aby zmienne decyzyjne przyjmowały wartości całkowite.

Kolejny notatnik jupytera pokazuje różne problemy związane z optymalizacją lokalną i globalną.

Sprawozdanie, w szczególności, może obejmować tylko porównanie naiwnej metody Monte-Carlo z metodą złotego podziału (zaprogramowanymi w dowolnym języku programowania).

5.5.3. Błędy, precyzja obliczeń

Przewiduję prezentację kwestii związanych z kumulacją błędów podczas obliczeń w notatniku Mathematici.

Zadania:

1. *Nieśmiertelne* zaprojektować i przeprowadzić eksperyment pozwalający stwierdzić z jaką precyzją (dwójkową: liczba bitów) wykonywane są obliczenia w Excelu/LibreOffice Calc/Arkuszach Google i jaki jest zakres wartości dostępnych do obliczeń.
2. Poczytać o arytmetyce poczwórnej[9] precyzji w języku C/C++ i zaimplementować jakiś obliczenia w przykładowym programie.
3. Kolejnym nieśmiertelnym zadaniem jest problem

$$x_{k+1} = \alpha x_k(1 - x_k) \quad x_0 = 0.5 \quad \alpha = 3.7$$

który jest bardzo niestabilny numerycznie. Można spróbować użyć biblioteki `mpmath` w Pythonie i zasymulować te obliczenia z praktycznie dowolną precyzją dziesiętną, zebrać w tabelce i porównać wyniki dla rożnych α (3,7 jest złe, ale 3.2 jest OK) i różnych precyzji. Gotowy notatnik jupytera jest również dostępny.

5.5.4. Sztuczna inteligencja

Przewiduję prezentację funkcjonowania sztucznej inteligencji (uczenia maszynowego) w Matematici.

Zadania:

1. Wydaje się, że minimum to powtórzenie procedury opisanej w instrukcji dla innego zestawu autorów.

[9] Jak wszyscy(?) wiedzą liczby zmiennoprzecinkowe w języku C to: `float`, `double`, `long double`. Typ `float` to 32 bity, typ `doubel` to 64. Niestety typ `long double` to nie całkiem 128 bitów (tylko 80). Warto to wiedzieć i znaleźć jakieś informacje źródłowe na ten temat oraz zdobyć informację jak jest to realizowane w różnych kompilatorach.

2. Prawdę mówiąc nie mam pomysłu co jeszcze ciekawego można zrobić ~~w warunkach domowych~~ z zakresu uczenia maszynowego czy sztucznej inteligencji. Ale trzeba pamiętać, że jest bardzo wiele przykładów takich zadań w Pythonie (Jupyter). Warto pamiętać, że również MATLAB pozwala na zabawy z uczeniem maszynowycm. **Zaakceptuję wszystko.**

5.6. Kilka słów więcej na temat optymalizacji

5.6.1. Wprowadzenie

Szukanie minimum (maksimum) funkcji to zadanie praktycznie bardzo istotne. Sprowadza się ono najczęściej do problemu:

— szukania miejsc zerowych funkcji pochodnych;
— iteracyjnych metod poszukiwania minimum (maksimum) metodami, które bardzo często sprowadzają się do lokalnego wyznaczania kierunku spadku funkcji (czyli lokalnego wyznaczania pochodnej).

Ale oprócz opisanych zadań możemy mieć do czynienia z zadaniami bardziej skomplikowanymi w których poszukujemy minimum (maksimum) uwzględniając ograniczenia narzucone na zmienne:

— na przykład zakres,
— oczekiwanie, że spełniają inne wymogi (na przykład są liczbami całkowitymi).

5.6.2. Ograniczenia

Najprostszą metodą uwzględnienia ograniczeń jest dodanie **kary** za przekroczenie ograniczeń. Jeżeli funkcja kary jest dobrana odpowiednio — działa to całkiem nieźle...

5.6.3. Programowanie liniowe

Szczególnym przypadkiem jest zadania optymalizacji gdy funkcja celu jest funkcją liniową o postaci:

$$f(x) = \sum_{i=1}^{N} a_i x_i$$

Funkcja taka maleje (rośnie) bez ograniczeń. Stąd zazwyczaj są to zadania optymalizacji z ograniczeniami postaci:

$$\sum_{j=0}^{N} \alpha_j^k x_j \leq \beta^k; \quad k = 1, 2, \ldots M$$

Bardzo często wymaga się również aby x_j były dodatnie (lub nieujemne).

Szczególnym przypadkiem są zadania w którym pojawiają się znaki równości w ograniczeniach. Wymagają one specjalnych metod postępowania.

Łatwo pokazać, że optimum funkcji z takimi ograniczeniami realizuje się na krawędzi (w wierzchołku) obszaru rozwiązań dopuszczalnych.

Istnieją specjalne metody rozwiązywania tego typu zagadnień. Najpopularniejszą jest metoda simplex.

5.6.4. Programowanie liniowe całkowitoliczbowe

Bardzo szczególnym przypadkiem może być zadanie identyczne z powyższym w którym dodano kolejne ograniczenie: wszystkie (lub niektóre) zmienne decyzyjne x_i mogą przyjmować tylko wartości całkowite. Zadania takie zazwyczaj pojawiają się w ekonomi.

Łatwo można pokazać, że rozwiązanie problemu bez ograniczenia na całkowitoliczbowość, a następnie „zaokrąglenie" wybranych zmiennych do wartości całkowitych ie prowadzi do rozwiązań optymalnych.

Opracowano specjalne metody rozwiązywania tego typu zadań.

5.6.5. Optymalizacja „dyskretna"

Kolejnym, szczególnym przypadkiem zadani optymalizacji jest optymalizacja dyskretna (to znaczy taka, w której zmienne decyzyjne nie przyjmują wartości ciągłych).

Typowym zadaniem z tej klasy jest zagadnienie komiwojażera (codziennie rozwiązywane[10] przez kurierów rozwożących przesyłki).

W zadaniu tym trzeba wybrać optymalną (najtańszą, zajmującą najmniej czasu,...) trasę dla kuriera (uwzględniającą dodatkowo ograniczenia czasowe dla odbiorców, pojemność i nośność pojazdu,...)

W najprostszym przypadku złożoność obliczeniowa tego problemu jest rzędu $O(n!)$.

5.6.6. Przykłady

1. Przygotowany notatnik jupyter zawiera przykład optymalizacji funkcji jednej i dwu zmiennych w Pythonie
2. Na stronie Linear programming and discrete optimization with Python using PuLP są bardzo ładne przykłady opisujące programowanie liniowe i całkowitoliczbowe. Notatniki pobrać można z GitHuba. Wymagają one pakietu PuLP, który standardowo nie jest dostępny w anakondzie. Można o dodać wykonując następujące polecenie:

[10] Albo i nie.

```
conda install -c conda-forge pulp
```

3.

5.7. METAR — METeorological Aerodrome Report

Jednym ze źródeł informacji meteorologicznych mogą być raporty METAR generowane przez każde lotnisko co godzinę (lub częściej w przypadku lotnisk wojskowych). Każdy raport zwiera informacje o

— kierunku i sile wiatru,
— widzialności (w metrach)
— zjawiskach meteorologicznych
— wielkości zachmurzenia
— wysokości podstawy chmur.
— temperaturze
— temperaturze punktu rosy
— ciśnieniu atmosferycznym

Dane są identyfikowane przez kod lotniska (dla Wrocławia jest to EPWR).

Dane są gromadzone i stosunkowo łatwo zdobyć dane bieżące, znacznie gorzej ze zdobyciem danych historycznych (często trzeba zanie płącić). Możecie Państwo podjąć pewien wysiłek, żeby dane takie zdobyć. Natomiast udostępniam pliki zawierające takie informacje wyciągnięte z mojej domowej „stacji meteorologicznej", pochodzące z:

— bazy METAR (temperatura i ciśnienie)
— lokalnego barometru (tylko ciśnienie)

Dane nie są kompletne (to znaczy pojawiają się pojedyncze wartości NaN). Do rozważań należy wybierać plik AVERAGE86400.

6. Laboratorium 1: Wprowadzenie, Matlab, Mathematica, różniczkowanie i całkowanie numeryczne

6.1. Cel zajęć

1. Wprowadzenie.
2. Przypomnienie/zapoznanie się z programami Mathematica i Matlab.
3. Generowanie danych „syntetycznych" o zadanych parametrach.
4. Wykresy funkcji i danych.
5. Algorytmy różniczkowania numerycznego przebiegów czasowych.
6. Algorytmy całkowania numerycznego przebiegów czasowych.

6.2. Zadania do wykonania

1. Zapoznanie się z elementarną dokumentacją systemów Mathematica i Matlab — przed zajęciami.
2. Przećwiczenie podanych przykładów.
3. Zapoznanie się z dokumentacją on-line obu systemów (w przypadku Matlaba dostępne są również programy demonstracyjne).
4. Przypomnienie sobie wzorów na numeryczne różniczkowanie (hint: iloraz różnicowy) i całkowanie (hint: metoda prostokątów, metoda trapezów).
5. Wygenerowanie „syntetycznych" zestawów danych i sprawdzenie procedur całkowania/różniczkowania.
6. Wygenerowanie syntetycznych zestawów danych (zaburzonych szumem o rozkładzie normalnym o średniej zero i zadanej wariancji) oraz sprawdzenie procedur całkowania/różniczkowania i zbadanie wpływu wariancji na wyniki.

6.3. Zadanie praktyczne

1. Wczytać dane przykładowe lub inne dane przykładowe.
2. Narysować wykres danych

3. Na wykresie inteligentnie zaznaczyć okresy wzrostu i spadku mierzonej wartości (użyć do tego pochodnej?)

Uwaga: wszystkie dane są skompresowane, w formacie xz. Po kliknięciu w plik o tym rozszerzeniu powinno automatycznie uruchomić się oprogramowanie pozwalające dostać się do zawartości pliku.

6.3.1. Format pierwszego zestawu danych danych

Dane pochodzą z pomiarów temperatury, ciśnienia atmosferycznego, wilgotności względnej oraz jasności światła z czujnika „wystawionego za okno". Okno jest od strony południowo-wschodniej więc, z oczywistych względów, dane nie mogą być traktowane jako wiarogodne. Pomiary realizowane są co pięć minut z czujnika TI CC2650 SensorTag.

Format danych (CSV)[1] jest następujący:

```
2015-09-27 17:15:00 CEST , 1443366900 , 1.6820185584e+01
```

Pierwsza kolumna to czas w postaci czytelnej, druga — tak zwany unix time stamp, a trzecia to zmierzona wartość. Gdy zamiast zmierzonej wartości pojawi się `NaN` oznacza to, że brakuje pomiaru. Taki punkt powinien „wypaść" z danych.

Dane pochodzą z systemu RRDtool, który dane zapisuje w specyficznych bazach danych wykorzystując format Round Robin Archives, który pozwala przechowywać najaktualniejsze dane z ostatniego okresu (na przykład doba) i skonsolidowane dane z okresów dłuższych — tydzień, miesiąc, rok. Stąd pojawiające się zestawach danych informacje typu:

```
--- AVERAGE 300
```

określają typ konsolidacji (może to być oprócz AVERAGE — MIN lub MAX) oraz okres konsolidacji (w sekundach: 300, 1800, 86400, 2678400). Dane nawet jeżeli pojawiają się częściej są uśredniane.

6.3.2. Format drugiego zestawu danych

W srugim przypadku dane pochodzą z osobistej stacji pogodowej netatmo. Pomiary robione są na zewnątrz i wewnątrz pomieszczeń. Mierzone jest ciśnienie atmosferyczne (tylko wewnątrz) temperatura, poziom szumu (tylko wewnątrz) i wilgotność.

Pliki zawierają dane mierzone „dosyć gęsto", ale okres próbkowania jest niejednostajny. Na przykład w przypadku szumu, oprócz znacznika czasu (TimeStamp) jest czas podany w formie tekstowej oraz wartość mierzonego parametru. Separatorem jest średnik.

[1] Comma Separated Values.

```
Timestamp;"Timezone : Europe/Warsaw";noise
1485903709;"2017/02/01 00:01:49";36
```

W przypadku innych parametrów będzie identycznie. Separatorem części ułamkowej (jeżeli występuje) jest kropka.

7. Laboratorium 2: Interpolacja

7.1. Wstęp

Rozpatrzmy takie zadanie: Mamy zestaw par punktów $\{x_i, y_i\}, i = 0, 1, 2, \ldots, N$. Mogą pochodzić one z pomiarów jakiegoś zjawiska (z eksperymentu) albo z próbkowania jakiejś nieznanej funkcji $f(x)$ ($y_i = f(x_i)$). Nie znamy zależności pomiędzy x a y i poszukujemy takiej funkcji $w(x)$, że: $w(x_i) = y_i$.

Co więcej bardzo często żądamy dodatkowo, żeby funkcja pozwalała na łatwe manipulacje (dodawanie, mnożenie funkcji, różniczkowanie), a wyliczanie jej wartości nie było zbyt kosztowne.

Czasami zadanie stawiane jest niej ambitnie — mamy zestaw $\{x_i, y_i\}, i = 0, 1, 2, \ldots, N$, a interesują nas wartości funkcji „między" tymi punktami.

W przypadku, gdy $A \leq x_1 \leq x_2 \leq \cdots \leq x_n \leq B$, a interesują nas wartości funkcji dla argumentów $x < A$ lub $x > B$ (czyli **spoza** zakresu pomiarów) zadanie będziemy nazywać **ekstrapolacją**.

7.2. Wielomiany

Bardzo często jako funkcję $w(x)$ wybierane są wielomiany. Łatwo je różniczkować, stosunkowo łatwo obliczać wartość a i zadanie znalezienia współczynników wielomianu interpolacyjnego jest stosunkowo proste.

$$w_n(x) = \sum_{i=0}^{N} a_i x^i$$

Do wyliczania wartości takiego wielomianu najlepiej stosować schemat Hornera.

7.3. Interpolacja Lagrange'a

Interpolacja Lagrange'a polega na znalezieniu dla danego zestawu danych $\{x_i, y_i\}, i = 0, 1, 2, \ldots, N$ wielomianu W_n stopnia nie wyższego niż n, którego wartości w $n + 1$ punktach x_i są takie same jak wartości interpolowanej funkcji, tzn.:

$$W_n(x_i) = y_i, \quad \text{dla } i = 0, 1, 2, \ldots, N$$

przy czym zakładamy, że gdy $i \neq j$ to $x_i \neq x_j$.

Można pokazać, że tak postawione zadanie interpolacyjne ma jednoznaczne rozwiązanie, które można przedstawić w postaci:

$$W_n(x) = \sum_{i=0}^{N} y_i l_i(x)$$

gdzie funkcja pomocnicza $l_i(x_j) = \delta_{ij} = \begin{cases} 1 & \text{dla } i = j \\ 0 & \text{dla } i \neq j \end{cases}$, w szczególności:

$$l_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{N} \frac{x - x_j}{x_i - x_j}$$

Można pokazać że jest to jednoznaczne rozwiązanie problemu interpolacji Lagrange'a.

Niestety, postać Lagrange'a wielomianu interpolacyjnego jest bardzo niewygodna do prowadzenia jakichkolwiek obliczeń. Zazwyczaj wylicza się współczynniki wielomianu klasycznego i używa ich w obliczeniach.

Aby wyliczyć współczynniki wielomianu należy rozwiązać układ $n+1$ równań liniowych:

$$a_0 + a_1 x_+ a_2 x_i^2 + \cdots + a_N x_i^N = y_i, \quad i = 0, 1, \ldots, N$$

Z względu na specyficzną postać współczynników $(1, x, x^2, \ldots, x^N)$ — są one zależne, co może być problemem w przypadku równoodległych węzłów) najlepiej stosować specjalne metody rozwiązywania tego układu równań.

Jedna z nich, oparta na wyliczaniu ilorazów różnicowych opisana zostanie pokrótce niżej. Wspominam o niej z „konikarskiego obowiązku" — to właśnie do wyliczania wartości wielomianów interpolacyjnych Babbage budował swój pierwszy „komputer:" Maszynę różnicową.

Wielomian interpolacyjny $W_n(x)$ zapisywany jest w alternatywnej postaci:

$$W_N(x) = \sum_{i=0}^{N} b_k p_k(x)$$

Wielomiany $p_k(x)$ opisane są wzorem:

$$\begin{aligned} p_0(x) &= 1 \\ p_k(x) &= (x - x_0)(x - x_1) \ldots (x - x_{k-1}), \quad k = 1, 2, \ldots, N \end{aligned}$$

Współczynniki b_k dane są wzorem:

$$b_k = \sum_{i=0}^{k} \frac{y_i}{\prod_{\substack{j=0 \\ j \neq i}}^{k} (x_i - x_j)}$$

Współczynniki b_k noszą nazwę *ilorazów różnicowych* funkcji f (pamiętamy, że $y_i = f(x_i)$) opartej na węzłach $x_0, x_1, \ldots, x_k$. Ilorazy różnicowe oznaczać będziemy tak: $f[x_l, x_{l+1}, \ldots, x_{l+k}]$

$$f[x_l, x_{l+1}, \ldots, x_{l+k}] = \sum_{i=l}^{l+k} \frac{y_i}{\prod\limits_{\substack{j=l \\ j \neq i}}^{l+k} (x_i - x_j)}$$

Można pokazać, że zachodzi następująca zależność rekurencyjna:

$$f[x_l, x_{l+1}, \ldots, x_{l+k}] = \frac{f[x_{l+1}, x_{l+2}, \ldots, x_{l+k}] - f[x_l, x_{l+1}, \ldots, x_{l+k-1}]}{x_{l+k} - x_l}$$

Wyliczenie współczynników b_k sprowadza się do rekurencyjnego wyliczania ilorazów różnicowych funkcji f. Budujemy tablicę zawierającą wartości x_i, wartości $f(x_i)$ oraz odpowiednie ilorazy różnicowe:

$$\begin{array}{llllll}
x_0 & f(x_0) & & & & \\
x_1 & f(x_1) & f[x_0, x_1] & & & \\
x_2 & f(x_2) & f[x_1, x_2] & f[x_0, x_1, x_2] & & \\
\ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\
x_n & f(x_n) & f[x_{n-1}, x_n] & \ldots & \ldots & f[x_0, \ldots, x_n]
\end{array}$$

Szukane współczynniki b_k równe są elementom przekątniowym powyższej tablicy.

Powyższe obliczenia można wykonać bardzo łatwo korzystając z następującego algorytmu:

```
for (k = 1; k <= n, k++)
  for (l = n; l >= k; l--)
    f[l] = (f[l] - f[l - 1]) / (x[l] - x[l - 1]);
```

Ponieważ najwygodniejszą (i najefektywniejszą) metodą liczenia wartości wielomianów jest metoda Hornera — trzeba ze korzystając ze współczynników b_k wyliczyć współczynniki a_i wielomianu w postaci naturalnej. Jest to również dosyć proste [3]. Realizuje je następujący algorytm:

```
a[n] = b[n];
for (i = n - 1; i~>= 0; i--)
  {
    xi = x[i];
    a[i] = b[i];
    for (k = i; k <= n - 1; k++)
      a[k] = a[k] - xi * a[k + 1];
  }
```

Naszkicowany powyżej algorytm jest praktycznie najefektywniejszym algorytmem interpolacji. Jedynie w przypadku interpolacji trygonometrycznej i użycia Szybkiej Transformaty Fouriera, dla dużych N będzie ona tańsza od algorytmu ilorazów różnicowych. Zainteresowanie Babbage'a tym algorytmem nie powinno zatem budzić zdziwienia.

Przydaność algorytmu interpolacji pokazuje przykładowy notatnik Mathematici, gdzie na podstawie siedemnastu wartości funkcji sinus, które każdy zna (powinien znać) budowany jest wielomian interpolacyjny. Dla dużych n (w tym wypadku 16) obliczenia chwilę trwają, ale wyniki są całkiem dokładne.

7.4. Funkcje sklejane

Interpolacja za pomocą wielomianów stopnia n, gdy n jest duże ma szereg efektów ubocznych — gwarantuje co prawda, że $w_n(x_i) = f(x_i)$, ale nie gwarantuje żadnego „przyzwoitego zachowania pomiędzy węzłami. Im więcej punktów — tym wyższy stopień wielomianu i tym „gorzej" się on zachowuje.

Funkcję rzeczywistą S nazywamy funkcją sklejaną stopnia m z węzłami $a = x_0 < x_1 < \ldots < x_N = b$ jeśli

1. w każdym przedziale (x_{i-1}, x_i) dla $i = 0, 1, \ldots, N+1$ $(x_{-1} = -\infty, x_{N+1} = \infty)$ S jest wielomianem stopnia nie wyższego niż m,
2. S i jej pochodne rzędu $1, 2, \ldots, m-1$ są ciągłe na całej osi rzeczywistej $S \in C^{m-1}$.

Gdy $m = 1$, funkcja sklejana jest po prostu łamaną. Otrzymana funkcja interpolacyjna będzie ciągła, ale pochodna będzie nieciągła, co znaczy, że funkcja nie jest gładka. Można (podwyższając m) doprowadzić do sytuacje, że funkcja jest ciągłą, ma pochodną, która również jest ciągła. Otrzymana funkcja będzie gładka. (Natomiast może okazać się, że druga pochodna już ciągłą nie jest — co ma swoje konsekwencje.) Zazwyczaj ciągłość pierwszej, a czasami drugiej pochodnej jest wystarczająca.

Funkcje sklejane to specjalna kategoria funkcji interpolacyjnych. Są one skonstruowane z „kawałków" (funkcja może być inna w każdym przedziale interpolacji), ale mają szereg zalet. Nadają się zwłaszcza, gdy trzeba na podstawie siatki (na przykład MES) utworzyć gładką powierzchnię.

Szczególnym przypadkiem są krzywe Beziera (Bezier curve).

7.5. Mathematica

7.5.1. Interpolacja wielomianowa

Do wyliczania wielomianów interpolacyjnych służy funkcja **Interpolation**. Można jeż użyć albo tak:

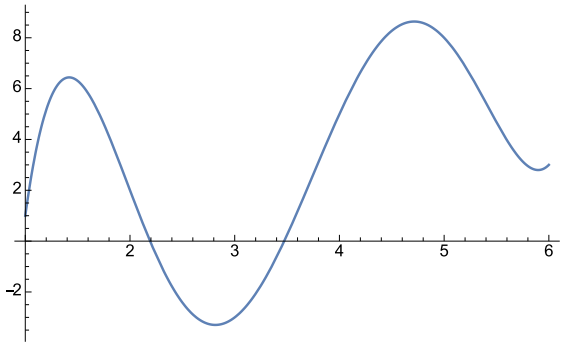
data = {1, 2, −3, 5, 8, 3}
f = Interpolation[data, InterpolationOrder → 4]

W tym przypadku zakłada się, że nasz zestaw danych to pary: $\{(1,1), (2,2), (3,-3), (4,5), (5,8), (6,3)\}$.

Rząd interpolacji (**InterpolationOrder**, standardowo 3) jest bardzo ważnym parametrem. Interpolacja odbywa się w sposób opisany wcześniej tylko wtedy, gdy rząd interpolacji równa się $n-1$ (gdzie n to liczność zbioru danych). Gdy rząd jest mniejszy niż $n-1$ — interpolacja dokonywana jest za pomocą jakiejś formy funkcji sklejanych. W szczególności można zażądać, żeby użyta była metoda funkcji sklejanych dodając parametr Method→"Spline".

Wynik interpolacji (w tym wypadku f) jest funkcją i może być używane tak jak każda funkcja: **p = Plot[f[x], {x, 1, 6}]** dając w efekcie piękny wykres, albo tak:

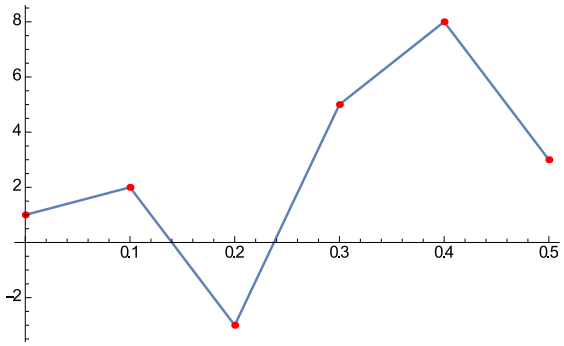
y = f[2.5]
−2.41406



Albo, gdy mamy zestaw danych **pp**, który jest tablicą dwuwymiarową zawierającą współrzędne węzłów w postaci (x, y):

pp $= \left\{\{0,1\}, \left\{\frac{1}{10}, 2\right\}, \left\{\frac{1}{5}, -3\right\}, \left\{\frac{3}{10}, 5\right\}, \left\{\frac{2}{5}, 8\right\}, \left\{\frac{1}{2}, 3\right\}\right\}$

ff = Interpolation[pp, InterpolationOrder → 1];
Show[Plot[ff[x], {x, 0, 0.5}], ListPlot[pp, PlotStyle → Red]]



Przykładowy notatnik Mathematici pokazujący zastosowanie interpolacji.

7.5.2. Interpolacja funkcjami sklejanymi

Do interpolacji funkcjami sklejanymi służyć mogą funkcje **Interpolate** lub **ListInterpolate**, gdy podamy jako metodę interpolacji "Spline". (W zasadzie nie widzę między nimi różnicy).

Przykładowy notatnik do porównania interpolacji wielomianowej i splajnów.

7.6. Matlab

Zakładam, że wszyscy Państwo znakomicie znacie Matlaba. Więc opisy są minimalne. Sugerują zabawy z Mathematicą.

7.6.1. Interpolacja wielominowa

Funkcja **interp1** może być użyta do prostych zadań interpolacji. Korzysta z metod: 'nearest', 'linear','spline','pchip', or 'cubic' i nie interpoluje w sposób opisany w części teoretycznej.

Użycie:

```
vq = interp1(x, v, xq, method)
```

gdzie x tablica współrzędnych x węzłów interpolacji, v — tablica współrzędnych y węzłów, xq tablica zawierająca wartości punktów, w których chcemy wyliczyć wartości interpolowanej funkcji. Zakłada się, że węzły interpolacji są zapisane w kolejności rosnącej.

Istnieje również możliwość interpolacji na podstawie danych, które nie są w żaden sposób uporządkowane: na przykład mamy zestaw punktów w przestrzeni trójwymiarowej i chcemy narysować poziomice. Używa się wówczas metod opisanych jako *interpolating scattered data*.

7.7. Zadania

1. Przygotować przykład pokazujący zachowanie zachowanie wielomianu interpolacyjnego pomiędzy węzłami interpolacji.
2. Użyć interpolacji wielomianowej i za pomocą funkcji sklejanych do przebiegu temperatury z czujnika podczerwonego (plik ma w nazwie `temperatura_ir`). W wyniku powinniśmy dostać dwa różne „przybliżenia" funkcji ale o tych samych wartościach w węzłach interpolacji. Należy policzyć pochodne obu funkcji interpolacyjnych. Wyniki porównać. Wyciągnąć wnioski. (Zamiast interpolacji wielomianowej — jeżeli trudno lub nie da się jej wyliczyć — można użyć jakiegoś innego sposobu interpolacji.)
   Hint: w przypadku Mathematici, aby uzyskać pochodną funkcji f wystarczy napisać f'.

7.8. Sprawozdanie

Standardowe.

7.9. Lektury uzupełniające

Nieco teorii, ale inaczej podanej znaleźć można na blogu Szymona Wąsowicza:

1. Tajniki interpolacji, część 1
2. Tajniki interpolacji, część 2
3. Tajniki interpolacji, część 3
4. Tajniki interpolacji, część 4
5. Tajniki interpolacji, część 5
6. Tajniki interpolacji, część 6

8. Laboratorium 2a: Interpolacja trygonometryczna

8.1. Interpolacja trygonometryczna

W przypadku gdy, funkcja (zjawisko), którą się zajmujemy jest okresowa czyli

$$g(y+t) = g(y)$$

dla wszystkich y do interpolacji najlepiej użyć funkcji okresowych. Dokonując zamiany zmiennych ($x = \frac{2\pi}{t}$) można rozpatrywać funkcje okresowe o okresie 2π.

Zadanie interpolacji trygonometrycznej polega na znalezieniu dla danej funkcji f:

$$t_n(x) = \sum_{j=0}^{n} c_j e^{ijx}$$

($i = \sqrt{-1}$). Oczekujemy, że wielomian ten przyjmie w $n+1$ punktach x_k z przedziału $[0, 2\pi]$ te same wartości co interpolowana funkcja, tzn.:

$$t_n(x_k) = f(x_k)$$

gdzie $x_k \neq x_l$, gdy $k \neq l$.

Zadanie te rozwiązuje się podobnie jak poprzednie rozwiązując układ $n+1$ równań liniowych z $n+1$ niewiadomymi $c_o, c_1, \ldots, c_n$

$$\sum_{j=0}^{n} c_j z_k^j = f(x_k)$$

gdzie $z_k = e^{ix_k}$.

Załóżmy, że węzły interpolacji są równoodległe czyli $x_k = \frac{2k\pi}{n+1}$ $(k = 0, 1, \ldots, n)$.

Ze względu na to, że funkcje e^{ijx} są ortogonalne (w sensie iloczynu skalarnego) zagadnienie można potraktować jako rzut dowolnego wektora w przestrzeni na osie. Można pokazać, że współczynniki c_j można wyznaczyć w sposób następujący:

$$c_j = \frac{(f, e^{ijx})}{n+1} = \frac{1}{n+1} \sum_{k=0}^{n} f(x_k) e^{-ijx_k}$$

(zapis $(\bullet, \bullet)$ oznacza iloczyn skalarny.

Jak wiadomo, wielomian $t_n(x) = \sum_{j=0}^{n} c_j e^{ijx}$ przedstawiony może być w postaci alternatywnej:

$$t_n(x) = \frac{1}{2}a_0 + \sum_{j=1}^{m}(a_j \cos jx + b_j \sin jx) + \delta\frac{1}{2}a_{m+1}\cos(m+1)x$$

gdzie $\delta = 0$, a $m = \frac{1}{2}n$, gdy n parzyste oraz $\delta = 1$, a $m = \frac{1}{2}(n-1)$, gdy n nieparzyste. Współczynniki a_j i b_j równe są odpowiednio:

$$a_j = \frac{2}{n+1}\sum_{k=0}^{n} f(x_k)\cos jx_k$$
$$b_j = \frac{2}{n+1}\sum_{k=0}^{n} f(x_k)\sin jx_k$$

Powyższy wzór nie jest wykorzystywany do obliczeń. Zazwyczaj stosuje się Szybką Transformatę Fouriera (FFT). Algorytm opracowany przez Cooleya i Tookeya w 1965 pominę. Koszt podejścia klasycznego wymaga $(n+1)^2$ operacji. Natomiast, w przypadku algorytmu FFT, gdy $n+1$ może być przedstawione (rozłożone) jako iloczyn $r_1 r_2 \dots r_p$ będzie rzędu $(n+1)(r_1 + r_2 + \dots + r_p)$. Algorytm FFT najczęściej stosowany jest, gdy $n+1 = 2^k$, wymaga on wówczas $n \log_2 n$ działań.

Pewnym problemem stosowania interpolacji trygonometrycznej jest to, że funkcja jest okresowa. To znaczy zakładamy, że $t_n(0) = t_n(x_{n+1})$. Dokonując pomiarów zazwyczaj tak nie jest. W dalszym ciągu „można" stosować FFT, funkcja będzie okresowa, ale... Aby uwolnić się od tego problemu, nakłada się zmierzone dane funkcję okna. Typowa funkcja okna dla $x = 0$ i $x = 2\pi$ przyjmuje wartość zero co powoduje zniekształcenie badanego przebiegu.

8.2. Mathematica

8.2.1. Interpolacja trygonometryczna

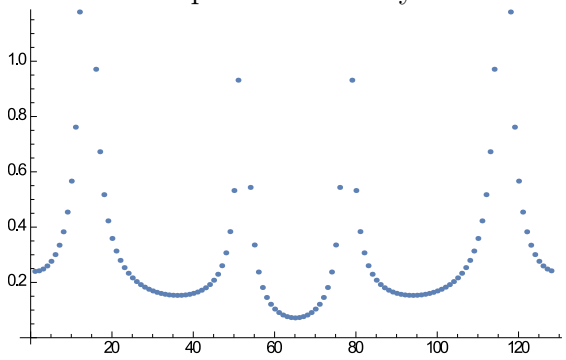
Zajmę się głównie szybką transformatą Fouriera. Funkcja Fourier służy do wyliczenia FFT z zadanego przebiegu danych. Najprostsze jeż użycie będzie takie:
a = Fourier[{1, 1, 2, 2, 1, 1, 0, 0}]
$\{2.82843 + 0.i, -0.5 + 1.20711i, 0. + 0.i, 0.5 - 0.207107i, 0. + 0.i, 0.5 + 0.207107i, 0. + 0.i, -0.5 - 1.20711i\}$

Odwrotna transformata Fouriera:
InverseFourier[a]
$\{1., 1., 2., 2., 1., 1., 0., -1.57009245868377\text{5*^-16}\}$ W każdym razie działa.

Teraz jakiś przebieg okresowy.

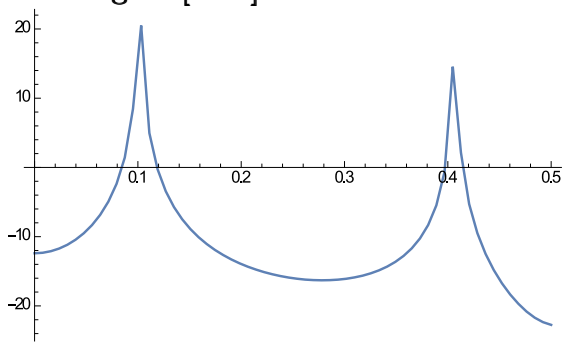
```
err = .0001;
data = Table[2Sin[0.2πn] + Sin[0.8πn] + RandomReal[{−err, err}], {n, 0, 127}];
fftdata = Fourier[data];
ListPlot[Abs[fftdata]]
```

Jak zinterpretować ten wykres? Czemu są cztery ekstrema (piki)?



Można również skorzystać z funkcji **Periodogram**.

```
Periodogram[data]
```



Jak zinterpretować ten rezultat?

Jak na podstawie powyższych wykresów zidentyfikować częstości przebiegu?

1. **Hint1**: Wygenerować jeden okres przebiegu o częstości 1 Hz i pokazać jego transformatę Fouriera oraz periodogram.
2. **Hint2**: Funkcja periodogram ma dodatkowy parametr **SampleRate**. Mówi on ile próbek na jednostkę czasu wykonano. Używany jest do skalowania osi X.

8.3. Matlab

8.3.1. Interpolacja trygonometryczna

Podstawowym narzędziem jest funkcja **fft**:

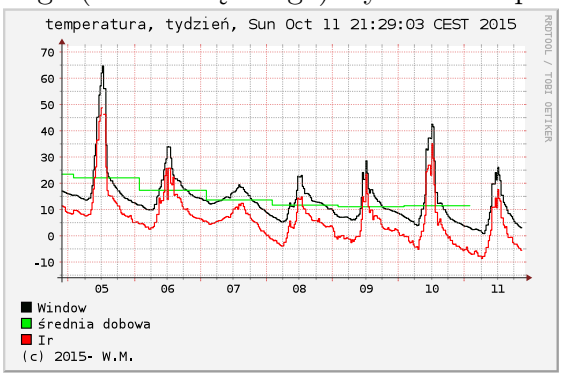
```
Fs = 1000;                    % Sampling frequency
T = 1/Fs;                     % Sample time
L = 1000;                     % Length of signal
```

```
t = (0:L-1)*T;                % Time vector
% Sum of a~50 Hz sinusoid and a~120 Hz sinusoid
x = 0.7*sin(2*pi*50*t) + sin(2*pi*120*t);
y = x + 2*randn(size(t));     % Sinusoids plus noise
plot(Fs*t(1:50),y(1:50))
title('Signal Corrupted with Zero-Mean Random Noise')
xlabel('time (milliseconds)')
NFFT = 2^nextpow2(L); % Next power of 2 from length of y
Y = fft(y,NFFT)/L;
f = Fs/2*linspace(0,1,NFFT/2+1);
% Plot single-sided amplitude spectrum.
plot(f,2*abs(Y(1:NFFT/2+1)))
title('Single-Sided Amplitude Spectrum of y(t)')
xlabel('Frequency (Hz)')
ylabel('|Y(f)|')
```

Zwracam uwagę, że pokazane rozwiązanie sugeruje, żeby długość ciągu danych była potęgą dwójki (polecenie: `NFFT = 2^nextpow2(L);`) czego nie wymaga Mathematica.

8.4. Zadania

1. Użyć interpolacji trygonometrycznej do znalezienia okresu zmienności tygodniowego (lub miesięcznego) wykresu temperatury. Wygląda on jakoś tak:



2. Przed rozpoczęciem tego zadania sugeruję zapoznanie się z informacjami zawartymi w rozdziale o Jupyterze oraz przeanalizowanie przykładowych notatników na temat wczytywania danych, eliminacji błędnych danych i szybkiej transformaty Fouriera.
3. Mając transformatę Fouriera warto spróbować przeprowadzić „naiwną, ręczną filtrację danych", która polegać będzie na wyzerowani nieistotnych składowych

transformaty i po zostawieniu tylko tych które wnoszą istotny wkład w przebieg oraz porównaniu przebiegów.

4. Opisać od czego zależy rozdzielczość (zdolność do identyfikowania w przebiegu złożonym składowych o bardzo bliskich częstotliwościach) szybkiej transformaty Fouriera. Można posiłkować się notatnikiem Jupyter.

9. Laboratorium 3: Aproksymacja

9.1. Wstęp

Podstawowym problemem interpolacji jest to, że stara się przeprowadzić funkcję przez wszystkie dane, które posiadamy (węzły). Ma to sens tylko i wyłącznie wtedy, gdy dane są dokładne i niezaburzone. Gdy jest inaczej — powinniśmy myśleć o jakimś ich uśrednieniu.

9.2. Aproksymacja

Aproksymacja to taka metoda „przybliżania" danych, w której zadaną funkcję stara się tak poprowadzić, żeby była **jak najbliżej** posiadanych punktów.

Osobną kwestią jest ustalenie co to znaczy „jak najbliżej"? Jeżeli mamy zestaw danych pomiarowych (par) $(x_i, y_i),\ i = 1, 2, \ldots, N$, szukamy takiej funkcji $f(x)$ aby:

$$Q = \sum_{i=1}^{N} (f(x_i) - y_i)^2 \to \min!$$

Tak postawione zadanie jest bardzo trudne — minimalizacja polega na wybraniu funkcji takiej, żeby... Znacznie prościej jest rozwiązywać zadanie następujące. Niech $f(x) = g(x, a)$ gdzie a jest wektorem parametrów $a = (a_1, a_2, \ldots, a_M),\ M \leq N$

$$Q = \sum_{j=1}^{N} (g(x_j, a) - y_i)^2 \to \min!$$

Teraz zadanie optymalizacji jest łatwiejsze — musimy wybrać wektor liczb. Kolejne uproszczenie polega na rozważaniu zadanie liniowego względem parametrów:

$$g(x, a) = \sum_{j=1}^{M} a_j \varphi_j(x),$$

a zadanie optymalizacji wygląda tak:

$$Q = \sum_{i=1}^{N} \left(\sum_{j=1}^{M} a_j \varphi_j(x_i) - y_i \right)^2 \to \min!$$

Jego rozwiązanie jest stosunkowo proste — wystarczy wyliczyć pochodne cząstkowe $\frac{\partial Q}{\partial a_j}$ i rozwiązać układ równań:

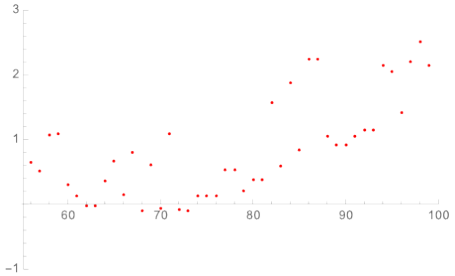
$$\frac{\partial Q}{\partial a_j} = 0; \quad j = 1, 2, \ldots, M$$

Zadanie dalej się upraszcza gdy przyjąć, że funkcja $\varphi_j(x) = x^j$.

9.3. Aproksymacja a interpolacja

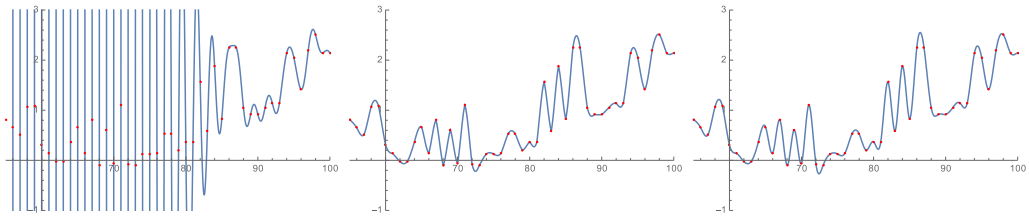
W przypadku zadania interpolacji żądamy, aby funkcja interpolująca przeszła przez wszystkie punkty (węzły interpolacyjne).

Poniżej przedstawiam zestaw punktów (pomiary temperatury termometrem IR).



Rysunek 9.1. Kilka punktów uzyskanych z pomiarów temperatury

Krzywe interpolacyjne mogą wyglądać tak jak na kolejnym rysunku. Czerwonymi kropkami zaznaczone są węzły interpolacji. Zwracam uwagę, że różnica między interpolacją Hermite'a a krzywymi sklejanymi nie jest specjalnie wielka. Niepokojąco natomiast wyglądają różnice pochodnych — pochodna interpolacji splajnami sześciennymi jest gładka.



Rysunek 9.2. Przykłady interpolacji (od lewej wielomian Newtona, sklejany Hermita i sklejany trzeciego stopnia)

9.4. Aproksymacja — Mathematica

Do realizacji aproksymacji wykorzystać można w Mathematici funkcję **Fit**. Jej wywołanie jest następujące:
Fit[*data*, *funs*, *vars*]
gdzie *data* to zestaw danych (par punktów), *funs* funkcja lub wektor funkcji którymi przybliżamy. Na przykład: $\{1, x, x^2, x^3, x^4, x^5, x^6, x^7, x^8, x^9, x^{10}, x^{11}, x^{12}\}$, *vars* — zmienna lub zmienne niezależne.

Powyższy zestaw jednomianów w różnych potęgach można łatwo wygenerować automatycznie:

$$\text{funs} = \text{Table}\left[x^i, \{i, 0, 10\}\right]$$

$$\left\{1, x, x^2, x^3, x^4, x^5, x^6, x^7, x^8, x^9, x^{10}\right\}$$

i dalej:

$$\text{funkcja1} = \text{Fit}[\text{dane}[[\text{All}, 2]], \text{funs}, x]$$

w wyniku dostajemy współczynniki wielomianu:

$$\begin{aligned}
&- 8.003515809018834\text{*^-}20x^{10} + 1.1265995371896338\text{*^-}16x^9 \\
&- 6.645297813196042\text{*^-}14x^8 + 2.1252203010076843\text{*^-}11x^7 \\
&- 3.981375997713063\text{*^-}9x^6 + 4.408253297181539\text{*^-}7x^5 \\
&- 0.0000278354x^4 + 0.00093783x^3 - 0.0165472x^2 + 0.22931x - 1.59072S
\end{aligned}$$

Korzysta się z otrzymanej funkcji aproksymacyjnej dosyć łatwo, na przykład:

$$\text{Plot}[\text{funkcja1}, \{x, 0, 288\}, \text{PlotStyle} \to \text{Blue}]$$

albo

$$\text{ff1}[\text{x\_}] = \text{funkcja1};$$

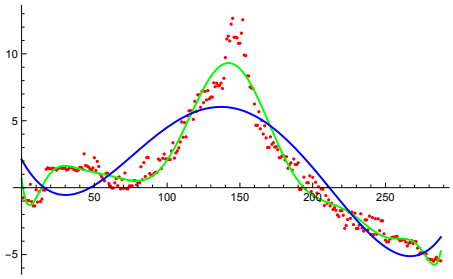
i

$$\text{Plot}[\text{ff1}[x], \{x, 0, 288\}]$$

W przypadku bardziej skomplikowanych zadań wykorzystać można również funkcje **FindFit** (funkcja aproksymująca nie musi liniowo zależeć od parametrów), **LinearModelFit** (tylko dla modeli liniowych) i chyba najogólniejszą: **NonlinearModelFit**.

Na poniższej ilustracji przykład aproksymacji dobowych zmian temperatury z termometru IR wielomianami stopnia 12 (zielony) i 4 (niebieski)).

Jak widać — cały problem sprowadza się do wyboru odpowiedniej funkcji aproksymacyjnej.



Rysunek 9.3. Aproksymacja danych wielomianami różnego stopnia: niebieski — 4, zielony —12

9.5. Zadanie do wykonania

Wybrać jakiś przebieg dobowy i przybliżyć go za pomocą jakiejś sprytnej funkcji (która dobrze będzie oddawała istotę zmienności przebiegu.

Uwagi:

1. Dane otrzymane z pomiarów zawierają bardzo duże wartości współrzędnych x. Może to stanowić problem podczas aproksymacji. Stanowczo więc zalecam przesunięcie czasu do zera (to znaczy pierwszy pomiar dokonywany jest w chwili 0, a następne co 300 sekund). Można to osiągnąć tak:
   dane = Import[AVERAGE300.dat];
   xmin = $[Min$[dane[[All, 1]]];
   dane[[All, 1]] = dane[[All, 1]] − xmin

   Można też porównać otrzymane wyniki z wynikami aproksymacji tylko wartości y (x przyjmuje wartości 1,2,...):
   $funkcja1 = Fit[dane[[All, 2]], funs, x]$
2. Jeżeli chodzi o wybór funkcji — sugeruję zacząć od wielomianów. Teoretycznie, im wyższy stopień wielomianu — tym przybliżenie lepsze. Tylko nie wiadomo czy sensowniejsze. Ambitni mogą wymyślić jakąś funkcję nieliniową lub złożyć z kawałków (patrz tutorial).

9.6. Matlab

Możliwości matlaba w zakresie aproksymacji wydają się być mniejsze. Toolbox Curve Fitting zawiera funkcję o nazwie fit i wywołaniu:

```
fit(x,y,fitType)
```

x i y to dane wejściowe. jako `fitType` podać należy łańcuch znaków określający rodzaj aproksymacji. Możliwości opisuje dokumentacja. Są tam wielomiany do stopnia 9 i parę innych funkcji.

Najprostsze użycie (korzystające z dostarczonych z matlabem danych przykładowych) wyglądać może tak:

```
load census;
f=fit(cdate,pop,'poly2')
plot(f,cdate,pop)
```

9.7. Aproksymacja a regresja

Wyobraźmy sobie, że mamy n pomiarów x_i jakiegoś parametru i chcemy zaaproksymować je wartością stałą $\bar{x}$. Chcielibyśmy, aby ta stała była jak najbliższa wszystkim pomiarom. Interesuje nas zatem taki problem:

$$Q = \sum_{i-1}^{n} (x_i - \bar{x})^2 \to \min!$$

czyli szukamy takiej wartości $\bar{x}$, która minimalizuje Q. Policzmy więc pierwszą pochodną $\frac{dQ}{d\bar{x}}$ (będziemy przyrównywać ją do zera):

$$\frac{dQ}{d\bar{x}} = \sum_{i=1}^{n} 2(x_i - \bar{x}) = 2\sum_{i=1}^{n} x_i - 2n\bar{x} = 0$$

zatem

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i.$$

Wzór ten przypomina nam znany ze statystyki wzór na średnią.

Nie od rzeczy będzie wspomnieć, że aproksymacja ma bardzo wiele wspólnego ze znaną ze statystyki regresją. W pewnym sensie jest to to samo (choć nie należy mówić tego głośno) — w przypadku regresji jest cała otoczka związana z probabilistyką (w szczególności zakłada się, że x_i są to obserwacje pewnej **zmiennej losowej** X, a to bardzo silne założenie — mówi ono, o tym, że istnieje rozkład prawdopodobieństwa zmiennej losowej X). Można w takim przypadku pokazać, że wyliczona wartość $\bar{x}$ ma pewne pożądane właściwości — wraz ze wzrostem n, $\bar{x}$ zmierza do wartości średniej rozkładu (jest estymatorem wartości średniej) i, że jest to estymator nieobciążony.

W technice wykorzystuje się średnią do „polepszania" wyników pomiarów. Zakładamy, że wartość a mierzona jest z pewnym addytywnym błędem, czyli: $x_i = a + \zeta_i$; zaburzenia ζ_i są niezależnymi realizacjami obserwacji pewnej zmiennej

losowej Z o średniej 0. Zatem wyliczając $\sum_{i=1}^{n} x_i$, po dokonaniu odpowiednio wielu pomiarów „odkryć" możemy prawdziwą wartość a.

Podobne interpretacje można zaprezentować również dla innych zadań, w których stosujemy aproksymację.

10. Laboratorium 4: Arytmetyka zmiennoprzecinkowa komputerów

10.1. Wstęp

Te zajęcia nawiązują do sposobu zapisu liczb opisanego w „części teoretycznej”. Ich celem jest odświeżenie tej wiedzy, zaprojektowanie prostego eksperymentu pozwalającego sprawdzić jak obliczenia są wykonywane (zwłaszcza w arkuszu kalkulacyjnym).

Druga część ma pokazać w jaki sposób można te ograniczenia omijać korzystając ze specjalnych bibliotek.

10.2. Proste obliczenia

Jest taki program [16][1]

```
#include <stdio.h>
int main()
{
    float s;
    double d;
    long double e;
    int i;
    s = d = e = 0.5;
    for(i=1;i<=100;i++)
    {
        s = 3.8F * s * (1.F - s);
        d = 3.8 * d * (1. - d);
        e = 3.8L * e * (1.L - e);
        if (i%10==0) printf("%10i %16.5f %16.5lf %16.5Lf\n", i, s, d, e);
    }
```

[1] Problem pojawił się po raz pierwszy podczas prób symulacji izolowanej populacji insektów.

```
//    printf("%ld\n", sizeof(long double));
    return 0;
}
```

nie prowadzi on żadnych skomplikowanych obliczeń przeprowadzając jedynie proste działanie:

$$x_{i+1} = \alpha x_i(1 - x_i)$$

gdzie $\alpha = 3.8$, a $x_0 = 0.5$.

Wykonuje on obliczenia na liczbach typu **float** (32 bity), **double** (64 bity) i **long double** (128 bitów). Program wykonuje 100 iteracji drukując co dziesiątą z nich. Wyniki programu (można go łatwo zapisać na dysku jako `caos.c` i skompilować poleceniem `gcc -o caos caos.c`.

Wyniki wyglądają jakoś tak:

10	0.18510	0.18510	0.18510
20	0.23951	0.23963	0.23963
30	0.88445	0.90200	0.90200
40	0.23023	0.82493	0.82493
50	0.76124	0.53714	0.53714
60	0.72004	0.66878	0.66879
70	0.89952	0.53189	0.53203
80	0.61599	0.93573	0.93275
90	0.84367	0.68312	0.79884
100	0.94858	0.65620	0.23138

Jak widać rezultaty różnią się w sposób znaczący. Jedynym wytłumaczeniem jest różna liczba bitów uwzględnianych w obliczeniach. Zatem przypuszczać można że wersja 128-bitowa jest najbliższa rzeczywistości.

Jeżeli wierzyć rozważaniom z [16] „poprawne wyniki” (uzyskane w arytmetyce o 1000 cyfr) są następujące:

10	0.18509
20	0.23963
30	0.90200
40	0.82492
50	0.53713
60	0.66878
70	0.53202
80	0.93275
90	0.79885
100	0.23161

i jak widać odrobinę różnią się od najlepszego uzyskanego wyniku.

Okazuje się, że dla $0 \leq \alpha < 3$ zachowanie obliczeń jest deterministyczne, a dla $3 \leq \alpha < 4$ — chaotyczne. Takie systemy (chaotyczne) są bardzo czułe na dokładność obliczeń.

10.3. Mathematica

Mathematica dla każdego obliczenia potrafi podać precyzję wyniku. Służy do tego funkcja `Precision[]`. Zazwyczaj obliczenia wykonywane są z „maszynową precyzją“ (co oznacza obliczenia na liczbach podwójnej precyzji (64 bity).

Dodatkowo można zadeklarować w jakiej precyzji mają być wykonywane obliczenia.

```
Block[{$MinPrecision = 5,$MaxPrecision = 5},x = a~* x * (1 - x)]
```

Znalazłem też instrukcję (której do końca nie rozumiem) powodującą, że wszystkie obliczenia w notatniku będą odbywać się z zadaną precyzją. Wygląda ona jakoś tak:

```
$PreRead=(#/.s_String/; StringMatchQ[s, NumberString] &&
   Precision @ ToExpression @ s==MachinePrecision:> s<>"‘1000."&);
```

i powinna być umieszczona na początku notatnika[2]. Precyzja określona jest stałą tekstową na samym końcu polecenia (`"‘1000.`"). Demonstruje to prosty przykład (notatnik Mathematici).

Powinno to wystarczyć do zaprogramowania opisywanego przykładu.

Mathematica to pełnowymiarowy język programowania, wyposażony w instrukcje warunkowe (`If` czy służące do tworzenia pętli (`For, Do, While`). Prosty przykład programiku z tymi instrukcjami zawiera notatnik.

Jak ktoś ciągle ma kłopoty może skorzystać z kolejnego przykładowego programu.

10.3.1. Pakiet Computer Arithmetics

Mathematica wyposażona jest w pakiet Computer Aruithmetics służący do symulowania obliczeń korzystając z jakiejś „wymyślonej” arytmetyki komputerowe. Pozwala to badać jaki wpływ na na precyzję obliczeń może mieć liczba dostępnych bitów czy zakres zmian wartości wykładnika.

W szczególności pakiet pozwala na symulowanie komputerów o arytmetyce dziesiętnej (nie binarnej). Wydaje się że do prowadzenia złożonych obliczeń inżynierskich taki rodzaj arytmetyki może być lepszy niż stosowana dziś arytmetyka binarna.

[2] Ustalona precyzja będzie obowiązywała we wszystkich równocześnie otwartych notebookach

Aby z pakietu skorzystać, trzeba go załadować. Służy d tego polecenie i`Needs`.

`Needs["ComputerArithmetic`"].`

Następnie definiujemy rodzaj arytmetyki poleceniem `SetArithmetic[d,b]` gdzie `d` to liczba używanych cyfr (niestety z zakresu 1 do 10), a `b` to podstawa systemu liczenia z zakresu od 2 do 16. Nie można zatem poszaleć z liczbą używanych w obliczeniach cyfr (co utrudnia nieco nasze zadanie).

Kolejne polecenie to `ComputerNumber` definiujące obiekt w wybranej arytmetyce:

```
x = ComputerNumber[0.5]
a = ComputerNumber[3.8]
```

(Standardowo nie można wykonywać obliczeń na liczbach mieszanych więc trzeba uważać na zapis:

```
x = x * a * (1 - x)
```

gdyż jedynka jest z innego świata (obiektem innego typu). Czyli powyższe trzeba zapisać jako:

```
x = x * a * (ComputerNumber[1] - x)
```

albo

```
one = ComputerNumber[1]
x = x * a * (one - x)
```

Takie obliczenia również można wykonać aby sprawdzić zachowanie naszego problemu. Od razu podpowiadam, że można wykorzystać arytmetykę szesnastkową o 10 cyfrach (uzyskamy większą precyzję niż w przypadku arytmetyki 10 o 10 cyfrach). Ale i tak to za mało.

10.4. Python

Język programowania python może być wyposażony w bibliotekę mpmath ([17]), pozwalającą na obliczenia w dowolnej precyzji.

Korzystanie z niej jest stosunkowo proste. Trzeba ją najpierw „załadować”

```
from mpmath import *
```

Polecenia `mp.prec` i `mp.dps` definiują precyzję obliczeń. Pierwsze określa ją w bitach, drugie w cyfrach dziesiętnych. Wystarczy korzystać tylko z jednego, gdyż ich wartości zależne są od siebie: prec $\sim$ 3.33 dps

```
>>> mp.dps=100
>>> mp.prec
336
```

Największy kłopot jest z deklaracją stałych, które nie mogą być standardowymi literałami.

```
>>> x=mpf(4)
>>> x
mpf('4.0')
>>> x/mpf('3.')
mpf('1.3333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333
>>> mp.dps=30
>>> x/mpf('3.')
mpf('1.33333333333333333333333333333327')
```

Wprowadzając dane trzeba pamiętać o błędach konwersji z układu dziesiętnego do dwójkowego. W związku z tym spodziewać można się dziwnych zjawisk

```
>>> mpf(.1)
mpf('0.10000000000000000555111515123125783')
>>> mpf('.1')
mpf('0.09999999999999999999999999999951')
>>> mpf(1)/mpf(10)
mpf('0.09999999999999999999999999999951')
```

w pierwszym przypadku najpierw robiona jest konwersja z liczby dziesiętnej do binarnej, a później wynik traktowany jest jako liczba wielokrotnej precyzji. W drugim wypadku do funkcji `mpf` przekazujemy tekst, który jest interpretowany i konwertowany od razu na liczbę wielokrotnej precyzji.

Reszta jest już stosunkowo prosta.

1. Dodawanie: `fadd()`
2. Odejmowanie: `fsub()`
3. Negacja (zmiana znaku): `fneg()`
4. Iloczyn: `fmul()`
5. Iloraz: `fdiv()`

Chociaż jak już zmienne zostały „zadeklarowane” jako dużej precyzji można korzystać ze standardowych operatorów `+-/*`.

```
>>> mp.dps = 100
>>> x = mpf('1.')
>>> y = mpf('3.')
>>> z~= x / y
>>> nprint(z,20)
0.33333333333333333333
```

10.4.1. Idle

Można ułatwić sobie pracę w pythonie uruchamiając proste środowisko graficzne zwane **idle**. Pozwala ono zapisać program korzystając z prostego edytora i bardzo łatwo uruchamiać go.

10.4.2. Jupyter

Znacznie bardziej wygodne może być skorzystanie z notatnika Jupyter. Przygotowałem odpowiedni przykład, który można pobrać (wraz z innymi) stąd.

10.5. Zadania do wykonania

1. Zaproponować eksperyment obliczeniowy wyznaczający liczbę bitów i dokładność prowadzonych obliczeń.
2. Przetestować go na matlabie i Mathematici, w arkuszu kalkulacyjnym LibreOffice calc, Blockly i — jeżeli ktoś konto Google posiada dla arkusza kalkulacyjnego Google (i ewentualnie dla Microsoft Office Online).
3. Pobawić się w pythonie[3] i Mathematici prowadzeniem obliczeń w dużej precyzji programując podany na początku przykład i porównując (jakieś wykresy?) wyniki uzyskane dla różnych precyzji obliczeń.

[3] Programowanie w pythonie jest stosunkowo proste ([18], [19], [20], [21], [22]). Można również zerknąć do instrukcji laboratoryjnych do zajęć z Technologii Informacyjnych. Na stronach TI dla informatykji znajduje się również obszerny spis literatury. Natomiast, jeżeli ktoś boi się zaczynać, powinien zerknąc na przygotowaną stronę w Blockly zawierający program i pozwalającą na podejrzenie wersji skonwertowanej do pythona. Skopiować z tego okienka jest trudno, ale się da. Klikamy w nie i w ciemno Ctrl-A, Ctrl-C i zawartość wklejamy do edytora. Później trzeba tylko jeszcze podzielić na linie (i zadbać o właściwe wcięcia, ale jak robić to ostrożne – udaje się).

11. Laboratorium 5: Optymalizacja

11.1. Wstęp

Zadania optymalizacyjne to podstawa całej teorii sterowania. Zazwyczaj nie tylko nam chodzi, żeby osiągnąć zadany cel, ale również, aby osiągnąć go albo w najkrótszym czasie, albo z wykorzystaniem najmniejszej liczby zasobów.

Zadanie optymalizacji pojawić się może tylko wtedy, gdy możliwe jest więcej niż jedno rozwiązanie.

11.2. Minimalizacja funkcji jednej zmiennej

Problem jest – w zasadzie – bardzo prosty: należy znaleźć takie x, dla którego $f(x)$ przyjmuje wartość minimalną.

Mówimy też, że $\tilde{x}$ jest minimum lokalnym jeżeli

$$f(\tilde{x}) < f(x), \qquad ||x - \tilde{x}|| \leq \varepsilon$$

Funkcja może mieć wiele minimów lokalnych. Jeżeli dla $\tilde{x} \in X$ zachodzi:

$$f(\tilde{x}) = \inf_{x \in X} f(x)$$

wówczas mówimy, że $\tilde{x}$ jest minimum globalnym w zbiorze X.

Zadanie w najogólniejszym przypadku jest bardzo trudne! Jeżeli przyjąć dodatkowe założenia – że funkcja $f(x)$ jest ciągła (co najmniej odcinkami ciągła), a zbiór X jest zwarty i domknięty[1] (co to znaczy?) to można kusić się o tworzenie jakichkolwiek sensownych algorytmów szukania minimum. (Jeżeli funkcja $f(x)$ nie jest co najmniej odcinkami ciągła – może okazać się, że jedyną metodą poszukiwania minimum jest przegląd wartości funkcji dla wszystkich dopuszczalnych argumentów.)

Jeżeli funkcja jest różniczkowalna szukanie minimum może sprowadzić się do rozwiązania równania (układu równań):

$$f'(x) = 0$$

[1] Zbiór jest **domknięty**, gdy granica każdego ciągu, którego elementy należą do zbioru również należy do zbioru. Gdy dodatkowo zbiór jest ograniczony — jest **zawarty**.

i przeanalizowania wszystkich „podejrzanych punktów", w których zeruje się pierwsza pochodna funkcji.

Ale, szczerze mówiąc problem sprowadza się do zastąpienia jednego trudnego zadania innym trudnym zadaniem.

Jedyna ogólna metoda aby znaleźć minimum globalne to znaleźć wszystkie minima lokalne i wybrać z nich najmniejsze. Zatem, tak na prawdę, potrzebujemy dobrych algorytmów wyszukiwania minimów lokalnych.

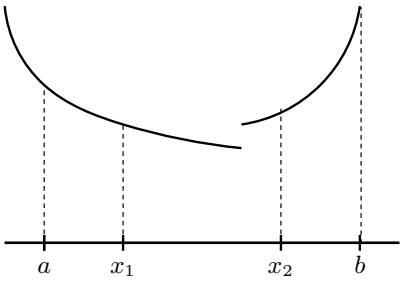
Na ogół X jest pewnym podzbiorem większej przestrzeni (mówiąc bardzo nieprecyzyjnie). Zadanie wyszukiwania minimum w zbiorze X jest wówczas zadaniem poszukiwania minimum z ograniczeniami. W pewnych przypadkach, gdy ograniczenia zadane są za pomocą równości stosując metodę mnożników Lagrange'a można zadanie z ograniczeniami sprowadzić do zadania poszukiwania minimum bez ograniczeń.

Gdy nie da się problemu rozwiązać analitycznie (na przykład rozwiązując układ równań), pozostaje stosowanie metod iteracyjnych. Jedną z nich opiszę poniżej.

11.2.1. Metoda złotego podziału

Zakładamy, że odcinkami ciągła funkcja funkcja $f(x)$ określona dla $a \leq x \leq b$ ma tylko jedno minimum lokalne. Chcemy stworzyć algorytm, który będzie generował ciąg liczb zbieżny do tego minimum.

Wyliczamy wartości funkcji na końcach przedziału oraz w dwu punktach wewnętrznych x_1 i x_2, porównamy wszystkie (4) wartości funkcji i wybierzemy najmniejszą z nich. Dla skupienia uwagi niech będzie to punkt x_1. Mamy trzy przedziały $[a, x_1]$, $[x_1, x_2]$ oraz $[x_2, b]$. Jasne jest, że możemy odrzucić przedział $[x_2, b]$. (Czemu!?)



Dalsza procedura polega na tym, żeby na odcinku $[a, x_2]$ wybrać kolejne dwa wewnętrzne punkty, wyznaczyć w nich wartości funkcji i procedurę powtórzyć...

Warto jednak uwzględnić fakt, że mamy już wyliczone wartości funkcji w trzech punktach: a, x_1 i x_2 – warto więc tak wybrać czwarty punkt aby wykorzystać wszystko to co już mamy.

Kolejny punkt tak dodajemy, aby kolejny (krótszy) odcinek podzielony był w sposób „podobny” do wyjściowego. Powinny być spełnione następujące zależności:

$$b - x_2 = x_1 - a, \qquad \frac{x_1 - a}{b - a} = \frac{x_2 - x_1}{x_2 - a}$$

Rozwiązanie jest następujące:

$$\frac{b - x_2}{b - a} = \frac{x_1 - a}{b - a} = \xi, \qquad \xi = \frac{2}{3 + \sqrt{5}} \approx 0{,}38$$

W każdej iteracji długość odcinka skraca się $1 - \xi \approx 0{,}62$ raza; zatem po n wyliczeniach wartości funkcji, długość wynikowego odcinek wynosi $(1 - \xi)^{n-3}$ początkowej długości. Gdy $n \to \infty$ długość odcinka zmierza do zera co gwarantuje zbieżność metody...

Dla ujednolicenia przyjmiemy $a = x_0$, $b = x_1$; na pierwszym kroku wyliczamy:

$$x_2 = x_0 + \xi(x_1 - x_0), \qquad x_3 = x_1 - \xi(x_1 - x_0).$$

Po wyliczeniu wartości funkcji i wybraniu minimum – odrzucony może być punkt z dowolnym indeksem... Może to sprawiać problem.

Jeżeli na pewnym etapie mamy cztery punkty x_i, x_j, x_k, x_l (któreś z nich są końcami przedziału). Załóżmy, że w punkcie x_i funkcja f osiąga minimalną wartość. Odrzucamy ten punkt, który jest najbardziej oddalony od x_i (Czemu takie działanie jest uprawnione? I w jakich warunkach?). Niech będzie to punkt x_l. Trzy pozostałe punkty szeregujemy w kolejności:

$$x_k < x_i < x_j.$$

Kolejny (wewnętrzny punkt) wyznaczamy ze wzoru:

$$x = x_j + x_k - x_i$$

(Czy na pewno spełnia on wymagania złotego podziału?)

Obliczenia kończymy gdy

$$|x_j - x_k| < \delta$$

11.2.2. Naiwna metoda Monte-Carlo

Losowo wybieramy punkty z zadanego obszaru. Po kilku iteracjach (ilu?) porównujemy wartości funkcji i wybieramy najmniejszą.

11.2.3. Minimum funkcji wielu zmiennych

Najprostsza metoda to „cięcie” funkcji wielu zmiennych „po osiach”. W każdym kierunku dokonujemy minimalizacji funkcji jednej zmiennej.

11.2.4. Zadania

1. Zaimplementować (matlab, python, C, C++, Mathematica) metodę złotego podziału i przetestować na funkcji $y = x^2$ w przedziale $[-2, 2]$, funkcji $y = \sin(x)$ w przedziale $[0, 3\pi]$. Zliczać liczbę obliczeń wartości funkcji do osiągnięcia zadanej dokładności. Porównać osiągnięty wynik z Naiwną Metodą Monte-Carlo (dla tej samej liczby obliczeń wartości funkcji)
2. Powtórzyć powyższe w przypadku funkcji dwu zmiennych: $F(x, y) = 10(y - \sin(x))^2 + 0{,}1x^2$

11.3. Programowanie liniowe

Programowanie liniowe to szczególny przypadek poszukiwania minimum. Funkcja celu jest liniowa, na przykład postaci:

$$f'x = \sum_{i=1}^{N} f_i x_i$$

gdzie f to wektor współczynników funkcji, a x wektor zmiennych niezależnych.

Ponieważ natura funkcji liniowej jest taka, że jest to funkcja monotoniczna (albo maleje, albo rośnie[2] wraz ze wzrostem wartości składowej x_i) — może przyjąć dowolnie wielkie lub dowolnie małe wartości. Minimalizacja/maksymalizacja nie mają sensu.

Zadanie zaczyna mieć sens, gdy wprowadzimy ograniczenia na zmienne. Najprostszy przypadek to ograniczenia w postaci zestawu nierówności:

$$Ax \leq b$$

(A to jakaś macierz) ale nierówności mogą mieć znacznie bardziej skomplikowaną postać:

$$A_{\text{eq}}x = b_{\text{eq}}$$

(A_{eq} to pewna macierz, a b_{eq} — wektor).

W matlabie problem rozwiązuje funkcja linprog. Zadanie może mieć wiele różnych postaci, najprostsze to $Ax \leq b$. Aby problem rozwiązać:

```
x = linprog(f,A,b)
```

Zadanie z ograniczeniami nierównościowymi i równościowymi:

```
x = linprog(f,A,b,Aeq,beq)
```

[2] Pomijam przypadek, gdy wartość jest stała.

`Aeq` to macierz A_{eq}, a `beq` to wektor $b_t exteq$.

Znaleźć mamy maksimum funkcji cx przy ograniczeniach $Ax = b$ i $x \geq 0$. A jest macierzą $m \times n$, $\text{rank}(A) = m$, i $b \geq 0$ (b, x, c to wektory odpowiednich rozmiarów).

W Mathematici, można użyć jednej z funkcji: **LinearProgramming**, **FindMinimum**, **FindMaximum**, **NMinimize**, **NMaximize**, **Minimize** oraz **Maximize**. Pełnię możliwości ma LinearProgramming. Opis w dokumentacji.

11.3.1. Ćwiczenia

Właściciel ciężarówki może przewieźć z miejscowości A do B cukier, mąkę i chipsy. W ciężarówce mieści się towar o objętości co najwyżej 7000 litrów i wadze co najwyżej 5 ton. 1 kg cukru zajmuje objętość 1,5 litra, 1 kg mąki 2 litry, natomiast 1 kg chipsów zajmuje objętość 4 litrów. Zysk od przewozu poszczególnych towarów jest następujący:
— za 100 kg cukru 8 zł,
— za 100 kg mąki 10 zł,
— za 100 kg chipsów 25 zł.

Ile cukru, mąki i chipsów powinien załadować właściciel ciężarówki aby zmaksymalizować swój zysk? Matematyczny model tak postawionego zadania jest następujący: Oznaczmy przez:
— x_1 – ilość cukru,
— x_2 – ilość mąki,
— x_3 – ilość chipsów
(za każdym razem w setkach kilogramów). Skoro ciężarówka może zabrać co najwyżej 5 ton towarów, musi zachodzić nierówność:

$$100x_1 + 100x_2 + 100x_3 \leq 5000$$

Z kolei ograniczenie objętości wyraża się wzorem

$$150x_1 + 200x_2 + 400x_3 \leq 7000$$

Zysk właściciela wynosi:

$$z \ = 8x_1 + 10x_2 + +25x_3$$

x_1, x_2 i x_3 muszą być oczywiście nieujemne. Po uproszczeniach otrzymamy więc problem programowania matematycznego:

$f(x_1, x_2, x_3) = 8x_1 + 10x_2 + 25x_3 \to \max$

$$\text{w zbiorze} \begin{cases} x_1 & +x_2 & +x_3 & \leq 50 \\ 1,5x_1 & +2x_2 & +4x_3 & \leq 70 \\ & x_j \geq 0 & \text{dla} & j = 1, 2, 3. \end{cases}$$

Kolejne ćwiczenie będzie równie proste.

$$
\begin{array}{rcrrrll}
 & & x_1 & +2x_2 & +3x_3 & \leq 5 & \\
 & & 2x_1 & +3x_2 & +5x_3 & \leq 8 & \\
 & & 3x_1 & +x_2 & +3x_3 & \leq 4 & \\
 & & & & & x_i \geq 0 & (i = 1, ..., 3) \\
\hline
z & = & 2x_1 & +x_2 & +3x_3 & \rightarrow & \max
\end{array}
\tag{11.1}
$$

Trzeba je rozwiązać za pomocą programu **linprog** a następnie „przedyskutować" uzyskane rozwiązanie (to znaczy sprawdzić czy spełnione są wszystkie ograniczenia, i zastanowić się, czy rozwiązanie jest optymalne[3]...) Można również użyć Mathematici.

[3] Jak to wykazać?

12. Laboratorium 6: Sztuczna inteligencja (AI)

12.1. Wstęp

Zacznijmy od prostego przykładu z dokumentacji funkcji **Classify** programu Mathematica.

Wczytujemy dane korzystając z serwisu Gutenberg zawierającego pełne teksty utworów literackich dostępne na wolnych licencjach.

Najpierw Szekspir:

```
Othello =
      Import["http://www.gutenberg.org/cache/epub/2267/pg2267.txt"];
Hamlet =
      Import["http://www.gutenberg.org/cache/epub/2265/pg2265.txt"];
Macbeth =
      Import["http://www.gutenberg.org/cache/epub/2264/pg2264.txt"];
```

Później Hugo:

```
LesMiserables =
      Import["http://www.gutenberg.org/cache/epub/135/pg135.txt"];
NotreDamede =
      Import["http://www.gutenberg.org/cache/epub/2610/pg2610.txt"];
TheManWho =
      Import["http://www.gutenberg.org/cache/epub/12587/pg12587.txt"];
```

I wreszcie Wilde:

```
TheImportance =
      Import["http://www.gutenberg.org/cache/epub/844/pg844.txt"];
ThePicture =
      Import["http://www.gutenberg.org/cache/epub/174/pg174.txt"];
AnIdeal =
      Import["http://www.gutenberg.org/files/885/885-0.txt"];
```

Tworzymy funkcję klasyfikującą wiążąc po dwa teksty z Autorem.

```
author = Classify[<|"William Shakespeare" -> {Othello, Hamlet},
"Oscar Wilde" -> {TheImportance, ThePictureo},
"Victor Hugo" -> {LesMiserables, NotreDamede}|>]
```

Sprawdzamy pozostałe teksty

```
author[{Macbeth, AnIdeal, TheManWho}]
```

12.2. Zadania

1. Na podstawie dokumentacji opisz jak rozumiesz działanie funkcji `Classify` (niekoniecznie w przypadku rozpoznawania autorów tekstów, ale, być może, patrząc na prostsze przykłady — patrz rozdz. 3.3.2). **Oczekuję sprawozdania!**
2. Powtórz inne przykłady z dokumentacji lub twórczo wykorzystaj gotowe funkcje klasyfikujące.
3. Spróbuj korzystając z serwisu Wolne Lektury powtórzyć zadanie klasyfikacji wybierając jakichś trzech pisarzy. Wolne Lektury również oferują pliki w prostym formacie tekstowym.
4. Rozważmy, że mamy następujący problem (serię uczącą): 500 bananów, 300 pomarańczy i 200 innych owoców. Metodą organoleptyczną sprawdziliśmy które z nich są słodkie. Dodatkowo dokonalismy ich (wzrokowej) klasyfikacji na owoce „długie” i „krótkie”. Wyniki przedstawia następująca tabelka:

Owoc	„Długi”	„Krótki”	Słodki	Nie słodki
Banan	400	100	350	150
Pomarańcza	0	300	150	150
Inne	100	100	150	50

Zadanie znakomicie nadaje się do zastosowania naiwnej metody Bayesa[1] do odpowiedzi na następujące pytanie: Jeżeli wylosujemy owoc, który jest długi, ale nie jest słodki, to jakie są szanse, że jest to banan?
Obliczenia można przeprowadzić bezpośrednio na prawdopodobienstwach [5], ale zaproponuj jak powinien wyglądać eksperyment (losowanie) uczenia maszynowego.
Mathematica ma funkcję losowego wyboru **RandomChoice**. Można jej użyć do wylosowania owocu, a później do przyporządkowania mu cech (zgodnie z zadanymi prawdopodoobieństwami). W ten sposób można wygenerować serię uczacą, która posłuży do wygenerowania funkcji klasyfikującej.
Warto sprawdzić jak funkcja klasyfikuje: gennerować elementy serii sprawdzającej (rodzaj owocu oraz jego cechy) i poddawać je klasyfikacji. Niestety, efekty

[1] Opisy są tu, tu czy tu.

klasyfikacji są zgodne jedynie w sensie probabilistycznym[2], a warto pamiętać (wiedzieć?), że tego typu klasyfikatorów używano do wskazywania celów dla dronów działjących na bliskim wschodzie...
Tak na marginesie: metody rozpoznawania spamu bardzo często oparte są na naiwnej metdozie klasyfikacji Bayesa.

12.3. Sprawozdanie

Laboratorium to może być relizowane na dwu zajeciach, ale powinno być zakończone obfitym sprawozdaniem i ciekawymi wynikami.

[2] Co to znaczy?

Część IV

Dodatki

A. Wielomian dwudziestego stopnia

W rozdziale 2.2 rozpatrywany jest wielomian o postaci:

$$w(x) = \prod_{i=1}^{20}(x-i) = \sum_{i=0}^{20} a_i x^i$$

Aby wyznaczyć jego współczynniki użyłem programu Mathematica, który znakomicie nadaje się do przeprowadzania obliczeń symbolicznych (to znaczy potrafi przeprowadzić wszystkie obliczenia w sposób „tradycyjny" — wykonując odpowiednie przekształcenia.

Wielomian zapisałem w postaci bardzo klasycznej:
Product[$x - i$, {i, 1, 20}]]
Następnie poprosiłem aby ten zwięzły zapis przeliczył do postaci tradycyjnej:
Expand[Product[$x - i$, {i, 1, 20}]]
W wyniku otrzymałem współczynniki wielomianu:
$2432902008176640000 - 8752948036761600000x + 1380375975364070400 0x^2 - 1287093124515098880 0x^3 +$
$803781182264505177 6x^4 -$
$359997951794760720 0x^5 + 120664780378037336 0x^6 - 311333643161390640x^7 + 6303081209929489 6x^8 -$
$101422998655114 50x^9 + 130753501054039 5x^{10} - 135585182899530x^{11} + 11310276995381x^{12} -$
$756111184500x^{13} +$
$40171771630x^{14} - 1672280820x^{15} + 53327946x^{16} - 1256850x^{17} + 20615x^{18} - 210x^{19} + x^{20}$

Następnie dokonałem modyfikacji współczynnika stojącego prze dziewiętnastej potędze i kazałem rozwiązać tak zmodyfikowane zadanie:
result =
$N[$
Solve[$2432902008176640000 - 8752948036761600000x +$
$1380375975364070400 0x^2 - 1287093124515098880 0x^3 +$
$803781182264505177 6x^4 - 359997951794760720 0x^5 +$
$120664780378037336 0x^6 - 311333643161390640x^7 + 6303081209929489 6x^8 -$
$101422998655114 50x^9 + 130753501054039 5x^{10} - 135585182899530x^{11} +$
$11310276995381x^{12} - 756111184500x^{13} + 40171771630x^{14} - 1672280820x^{15} + 53327946x^{16} -$
$1256850x^{17} + 20615x^{18} - (210 + 2^{-23})x^{19} + x^{20} == 0, x]]$
(Zmodyfikowany współczynnik oznaczony jest innym kolorem.)

Otrzymałem wynik typowy dla programu Mathematica:
$\{\{x \to 1.\}, \{x \to 2.\}, \{x \to 3.\}, \{x \to 4.\}, \{x \to 5.\}, \{x \to 6.00001\}, \{x \to$

$6.9997\}, \{x \to 8.00727\}, \{x \to 8.91725\}, \{x \to 20.8469\}, \{x \to 10.0953 - 0.643501i\}, \{x \to 10.0953 + 0.643501i\}, \{x \to 11.7936 - 1.65233i\}, \{x \to 11.7936 + 1.65233i\}, \{x \to 13.9924 - 2.51883i\}, \{x \to 13.9924 + 2.51883i\}, \{x \to 16.7307 - 2.81262i\}, \{x \to 16.7307 + 2.81262i\}, \{x \to 19.5024 - 1.94033i\}, \{x \to 19.5024 + 1.94033i\}\}$

Jak widać, część rozwiązań jest rzeczywista: 1, 2, 3, 4, 5, 6, ~ 7, ~ 8, ~ 9, ~ 21. Pozostałe są zespolone. Pytanie na ile ich moduły różnią się od kolejnych liczb naturalnych[1]?

Aby z wyniku (zapisanego w zmiennej result wydobyć poszczególne wartości należy użyć specjalnego operatora /. (Zamień wszystko[2]), a następnie wyliczamy wartość bezwzględną

Abs[*x*/.result]

Wynik:

{1., 2., 3., 4., 5., 6.00001, 6.9997, 8.00727, 8.91725, 20.8469, 10.1158, 10.1158, 11.9088, 11.9088, 14.2173, 14.2173, 16.9655, 16.9655, 19.5987, 19.5987}

Widać wyraźnie, że tak nieistotna — na pierwszy rzut oka — zmiana parametru, tak odmieniła wynik

[1] Zwracam uwagę, że wielomian stopnia dwudziestego może mieć co najwyżej dwadzieścia miejsc zerowych. W przypadku, gdy niektóre z nich są liczbami zespolonymi, to również liczba sprzężona będzie pierwiastkiem. Więc na pewno pozostałe pierwiastki nie wpasują się w kolejne liczby naturalne.

[2] http://reference.wolfram.com/language/ref/ReplaceAll.html

B. Metoda Hornera wyliczania wartości wielomianów

B.1. Wielomian

Jak wiadomo, wielomian (stopnia N), to wyrażenie postaci:

$$w(x) = \sum_{i=0}^{N} a_i x^I \tag{B.1}$$

B.2. Wyliczanie wartości wielomianu

W naturalny sposób, kod (C) wyliczający jego wartość wyglądać może tak:

```
double
wielomian (int N, double *a, double x)
{
  double w = 0.;
  int i;
  for (i = 0; i < N; i++)
    w += pow (x, i) * a[i];
}
```

Sprytny student zauważy, że kod ten można „zoptymalizować” w sposób następujący:

```
double
wielomian (int N, double *a, double x)
{
  double w = a[0];
  int i;
  for (i = 1; i < N; i++)
    w += pow (x, i) * a[i];
}
```

oszczędzając jeden obieg pętli (i jedno wywołanie funkcji pow). Czy możliwe są dalsze optymalizacje? Oczywiście, można zrezygnować z funkcji pow:

```
double
wielomian (int N, double *a, double x)
{
  double w = 0.;
  int i;
  double pow = 1;
  for (i = 0; i < N; i++)
    {
      w += pow * a[i];
      pow *= x;
    }
}
```

(Prawdę mówiąc warto rezygnować z funkcji pow, gdyż jest to armata wystawiona na wróbelka: podnosimy x do potęgi całkowitej.)

Czy są możliwe dalsze optymalizacje? Czy może to mieć wpływ na dokładność obliczeń? Czy jakąś własność numeryczną algorytmu?

„Tradycyjna" metoda obliczeń wymaga

$$1 + 2 + \cdots + (N-1) + N = \frac{N(N+1)}{2}$$

mnożeń i N dodawań.

B.3. Metoda Hornera

Zapisując wielomian nieco inaczej:

$$w(x) = a_0 + x(a_1 + x(a_2 + \cdots + x(a_{N-1} + xa_N)\ldots)) \tag{B.2}$$

redukujemy liczbę operacji do N dodawań i N mnożeń.

Co więcej algorytm Hornera jest numerycznie poprawny.

Algorytm Hornera może być również przydatny podczas dzielenia wielomianu $w(y)$ przez dwumian $y - x$ dając w wyniku współczynniki wyniku (wielomianu będącego ilorazem $\frac{w(y)}{y-x}$).

C. FAQ czyli Czasami Zadawane Pytania

C.1. Mathematica

1. W jaki sposób z tablicy wydobyć cały wiersz albo całą kolumnę?
   Niech mat będzie tablicą dwuwymiarową, utworzoną, na przykład poleceniem:
   ```
   mat = Table[Subscript[m, i, j], {i, 5}, {j, 5}];
   mat // MatrixForm
   ```
   $$\begin{pmatrix} m_{1,1} & m_{1,2} & m_{1,3} & m_{1,4} & m_{1,5} \\ m_{2,1} & m_{2,2} & m_{2,3} & m_{2,4} & m_{2,5} \\ m_{3,1} & m_{3,2} & m_{3,3} & m_{3,4} & m_{3,5} \\ m_{4,1} & m_{4,2} & m_{4,3} & m_{4,4} & m_{4,5} \\ m_{5,1} & m_{5,2} & m_{5,3} & m_{5,4} & m_{5,5} \end{pmatrix}$$
   Aby dobrać się do wybranego elementu używamy funkcji Part:
   `Part[mat, 1, 2]`
   albo „skrótu” `[[...]]`
   `mat[[1,2]]`
   Aby wydobyć cały wiersz wystarczy użyć polecenia:
   `mat[[4]]`
   $\{m_{4,1}, m_{4,2}, m_{4,3}, m_{4,4}, m_{4,5}\}$
   natomiast aby wydobyć kolumnę:
   `mat[[All,4]]`
   $\{m_{1,4}, m_{2,4}, m_{3,4}, m_{4,4}, m_{5,4}\}$
   Dodatkowo funkcja Span (której skrótem jest `;;` pozwala wybrać dowolną „podtablicę” z tablicy:
   `mat[[1 ;; 2, 3 ;; 5]]`
   $$\begin{pmatrix} m_{1,3} & m_{1,4} & m_{1,5} \\ m_{2,3} & m_{2,4} & m_{2,5} \end{pmatrix}$$
2. Jak utworzyć macierz o wymiarach $n \times m$, której zawartością są same zera?

   ```
   A = Table[0, {i, n}, {i, m}]
   ```

   $$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$
3. W jaki sposów wstrzymać wypisywanie zawartości wyników obliczeń (na przykład tablic) na ekranie?

Na końcu polecenia należy umieścić średnik.

4. W jaki sposób dodać (usunąć) element listy?
   Służą do tego polecenia **Append** (dodaj na końcu), **Prepend** (dodaj na początku), **Insert** (wstaw), **Riffle** (wstaw między elementami listy), **ReplacePart** (zamień) **Delete** (usuń). Dobry tutorial jest tu w helpie Mathematici oraz na stronach WWW.
   Dodanie elementu a na końcu listy l: Append[l,a]:

   $$\boldsymbol{l = \{1, 2, 3\}}$$

   $$\{1, 2, 3\}$$

   $$\boldsymbol{a = 4;}$$

   $$\boldsymbol{l = \text{Append}[l, a];}$$

   $$\boldsymbol{l}$$

   $$\{1, 2, 3, 4\}$$

   W szczególności działa coś takiego:

   $$\boldsymbol{L = \{\}}$$

   $$\{\}$$

   $$\boldsymbol{\text{For}[i = 0, i < 10, i{+}{+}, L = \text{Append}[L, i]]}$$

   $$\boldsymbol{L}$$

   $$\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$$

C.2. Matlab

1. ~~Po uruchomieniu matlaba bardzo często ekran (lub jego część) staje się nieczytelny? Co zrobić?~~

~~Problem jest znany. Być może uruchomienie matlaba z parametrem~~[1] ~~–softwareopengl sprawę załatwi. (Sprawdzamy czy można coś zrobić ze sterownikami karty graficznej).~~

~~Można poprosić aby standardowo matlab tak się uruchamiał, wykonując w matlabie polecenie:~~

```
opengl('save','software')
```

~~**Przepraszamy za usterki.**~~

2. Jak utworzyć macierz o wymiarach $n \times m$, której zawartością są same zera?
   Służy do tego funkcja `zeros`:
   ```
   A = zeros(n, m)
   ```
3. W jaki sposób wstrzymać wypisywanie zawartości wyników obliczeń (na przykład tablic) na ekranie?
   Na końcu polecenia należy umieścić średnik.
4. W jaki sposób dodać element do tablicy?
   Matlab jest dosyć odporny na działania na tablicach. W szczególności można napisać tak:
   ```
   >> z=[1,2,3]
   z =
        1     2     3
   >> z(4,4)=7
   z =
        1     2     3     0
        0     0     0     0
        0     0     0     0
        0     0     0     7
   ```
   najpierw nadaliśmy wartości tablicy o jednym wierszu i trzech kolumnach, a następnie dodaliśmy element w czwartym wierszu i kolumnie — tablica została automatycznie rozbudowana.
   Można też tak:
   ```
   >> for n=1:3
   d(n)=n
   end
   d =
        1
   d =
        1     2
   d =
        1     2     3
   ```

[1] W tym celu otwieramy terminal (naciskając równocześnie klawisze Ctrl-Alt-T i wpisujemy **matlab –softwareopengl**. Ale niekoniecznie to działa.

albo tak:

```
e=[]
>> for n=1:3
e(end+1)=n
end
e =
     1
e =
     1     2
e =
     1     2     3
```

5. Jak wczytać dane i pozbyć się linii zawierających NaN?
   a) Ściągamy dane, na przykład plik `temperatura_29012017_0001.csv.xz`: wget https://temisto.immt.pwr.wroc.pl/~myszka/Dane_Numeryczne/temperatura_29012017_0001.csv.xz
   b) Rozpakowujemy `xz -d temperatura_29012017_0001.csv.xz` i dostajemy plik `temperatura_29012017_0001.csv`.
   c) W pliku tym pierwszy wiersz to `--- AVERAGE 300`, kolejnych 288 wartości to pomiary (temperatury) z krokiem 300 sekund. Następnie (w lini 290) jest wiersz `--- AVERAGE 1800`, w linii 627 `--- AVERAGE 86400`, a w linii 993 — `--- AVERAGE 2678400`.
   d) Od linii 1114 sytuacja zaczyna się powtarzać, ale teraz będą wartości minimalne (agregacja danych polega nie na uśrednianiu, ale na wyliczaniu minimum z pomiarów). Czyli `--- MIN 300`, 1800, 86400, 2678400.
   e) Od linii 2227 są wartości maksymalne.
   f) Przetwarzanie najlepiej robić na pomiarach z doby (uśrednianie co 5 minut) lub tygodnia (uśrednianie co 30 minut).
   g) Plik można automatycznie „pociąć” na mniejsze kawałki używając dostępnego programu napisanego w awk. Pobieramy program:
   wget http://kmim.wm.pwr.edu.pl/myszka/wp-content/uploads/sites/2/2015/10/cut.awk.
   Uruchamiamy `gawk -f cut.awk temperatura_29012017_0001.csv`. W wyniku jego działania dostajemy pliki o nazwach:
   `AVERAGE300.dat`
   `AVERAGE1800.dat`
   `AVERAGE86400.dat`
   `AVERAGE2678400.dat`
   `MIN300.dat`
   `MIN1800.dat`
   `MIN86400.dat`

`MIN2678400.dat`
`MAX300.dat MAX1800.dat`
`MAX86400.dat`
`MAX2678400.dat`

h) Dane zawierać mogą wartości NaN. W tym przypadku oznacza to, że w danej chwili nie udało się dokonać pomiarów. Ogólne znaczenie tego symbolu opisuje, na przykład, Wikipedia: https://pl.wikipedia.org/wiki/NaN Po zaimportowaniu danych trzeba się tych wierszy pozbyć. W opisywanym przypadku (plik `temperatura_29012017_0001.csv`) wartości NaN występują w pliku AVERAGE86400 `grep NaN AVERAGE86400 | wc -l` Jest ich 15 sztuk.

i) Uruchamiamy Matlab i wczytujemy dane. Poprosiłem Matlaba o wygenerowanie skrypu dokonującego importu:

```
%% Import data from text file.
% Script for importing data from the following text file:
%
%    /home/myszka/przyklad/AVERAGE86400.dat
%
filename = '/home/myszka/przyklad/AVERAGE86400.dat';
delimiter = ' ';
fileID = fopen(filename,'r');
formatSpec = '%f%f%[^\n\r]';
dataArray = textscan(fileID, formatSpec, 'Delimiter', delimiter,
            'MultipleDelimsAsOne', true, 'EmptyValue' ,NaN,
            'ReturnOnError', false);
fclose(fileID);
czas = dataArray{:, 1};
temperatura = dataArray{:, 2};
clearvars filename delimiter formatSpec fileID dataArray ans;
```

j) Dalsze przetwarzanie jest dosyć proste. Trzeba znaleźć wszystkie te miejsca, w których pojawiają się NaNy (Dokumentacja Matlaba) (albo raczej te, w których NaNów nie ma): `nans = ~isnan(temperatura);` (nans to tablica zawierająca zera tam gdzie są NaNy i 1 w pozostałych miejscach). Selekcjonujemy dane:

```
Temperatura = temperatura(nans)
Czas = czas(nans)
```

Przy okazji sprawdźmy minimalną i maksymalną wartość zmierzonej temperatury: `max(Temperatura)` poda, że ta wartość to 3.8584e+04 a `min(Temperatura)` to -100.8635 Jak widać dane zawierają błędy pomiarowe.

`find(Temperatura==min(Temperatura))` zwraca wartość 347 (to numer wiersza) a `Temperatura(find(Temperatura==min(Temperatura))=[];` pozwala usunąć tę wartość. Ale lepiej robić to etapami (bo trzeba również usunąć odpowiadający jej znacznik czasowy)

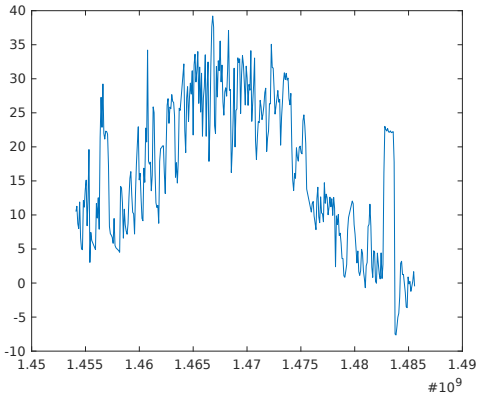
```
i=find(Temperatura==min(Temperatura);
Temperatura(i)=[];
Czas(i)=[];
```

. Podobnie z wartością maksymalną:

```
i=find(Temperatura==max(Temperatura));
Temperatura(i)=[];
Czas(i)=[];
```

Niestety, dane dalej zawierają błędy: max(T) wynosi 2.1329e+04 (ale min(T) to już sensowne -7.6313).

k) Na koniec można zrobić wykres danych z miesiaca: `plot(Czas, Temperatura);`. Niestety, wykres nie jest piękny (zwłaszcza jeżeli chodzi o opis osi X — rys. C.1). Problemem jest format zapisu czasu. Matlab używa formatu POSIX.

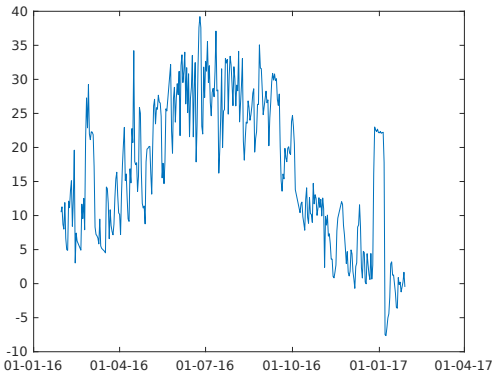


Rysunek C.1. Unix timestamp na osi X

Zależność między formatami jest następująca:
`posixtime = unix_time/86400 + datenum(1970,1,1);` Teraz wykres `plot(Czas / 86400` i po zdefiniowaniu formatu osi X `datetick('x','dd-mm-yy')` wygląda znacznie lepiej (rys. C.2).

6. Jak zrobić wykres, którego kolory zależą od jakiejś wartości?
Niech

```
x = 0:0.01:10;
```



Rysunek C.2. POSIX timestamp na osi X

```
y = sin(x);
```
to dane do naszego wykresu.
Chcemy sprawić akby wartości większe od zadanej wartość α były w innym kolorze niż mniejsze. (Kwestię wzrostu/spadku można rozwiązać analogicznie).

```
alfa=0.2
```

Wynikiem operacji
```
y > alfa
```
jest wektor współrzędnych, dla których ten warunek jest spełniony; zatem
```
wieksze = (y > alfa)
\end{enumerate}

Teraz musimy zrobić dwie kopie kolumny y; zmodyfiujemy je tak, zby jedna zawiera
\begin{verbatim}
gora = y;
dol = y;
gora(wieksze) = NaN;
dol(~wieksze) = NaN;
```
Sam wykres to już zabawa:
```
plot(x, dol, 'r', x, gora, 'g')
\section{Jupyter/Python}

\begin{enumerate}
\item W jaki sposób zapisać otrzymany w Jupyterze wykres (czy ogólniej obrazek)

\begin{enumerate}
```

```
\item
Najprościej tak jak w przeglądarce zapisuje się obrazki: klikamy prawym klawisze
\item
Inna metoda to użycie funkcji \verb|savefig()| tam gdzie (w przykładowych notatn
\end{enumerate}
\item
W jaki sposób zapisać tablicę danych numpy do pliku CSV?

Jeżeli nasza tablica z danymi nazywa się $a$ to następujące polecanie zapiszą ja
\begin{verbatim}
import numpy
numpy.savetxt("foo.csv",  a,  delimiter = ",")
```

C.3. Ogólne

1. Czy do programowania należy wykorzystywać **wyłącznie** Matlab i/lub Mathematicę?
   Nie. Można użyć dowolnego języka programowania dostępne w laboratorium (C, C++, Python, Jupyter, Mathematica, Matlab, Scilab, Blockly,...). W miarę potrzeby (i dostępności) mogą zostać doinstalowane kolejne. Proszę zgłaszać potrzeby.

D. Jupyter

D.1. Wstęp

Jupyter jest projektem Open Source, który zdobywa obecnie sporą popularność. Jego historia sięga roku 2011. Wystepuje w trzech smakach:
— Jupyter Notebook — klasyczne rozwiązanie bazujące na IPython Notebook, pozwalające łatwo edytować kod źródłowy.
— Jupyter Lab — najnowszy (2018) interfejs ma nieco większe możliwości.
— Jupyter Hub — serwer przystosowany do równoczesnej pracy wielu użytkowników.
Na pierwszy rzut oka jest to coś podobnego do Mathematici: interfejs pozwalający tworzyć notatnik oraz prowadzić w nim obliczenia czy wizualizować wyniki. O ile jądrem obliczeniowym Mathematici jest jej jądro o sporych możliwościach obliczeń symbolicznych i numerycznych, to jądrem obliczeniowym Jupytera może być jeden z wielu języków programowania[1]:
— głównie Python wraz z bibliotekami NumPy i SciPy,
— R,
— Julia,
ale również inne:
— JavaScript,
— Java,
— Scala,
— Go,
— C#,
— Ruby,
— Kotlin,
— Haskell.
Ale to nie koniec. Jąder obliczeniowych jest w sumie około stu, a wśród nich zarówno Matlab jak i Mathematica.

[1] Nazwa Jupyter pochodzi od Ju(lia) + Py(thon) + (e)R.

D.2. Instalacja

Sposób instalacji nie jest przedmiotem niniejszego krótkiego tutoriala. Znaleźć można go na stronach z dokumentacją jupytera[2] lub tu. Ale nie tylko tam. Program jest tak popularny, że praktycznie wszędzie tam gdzie sugerują korzystanie z niego podają też sposób instalacji.

Oprogramowanie bez problemu powinno pracować w każdym popularnym środowisku[3]

Wydaje mi się, że instalacja anacondy może być najlepszym rozwiazaniem niż instalacja pojedynczych pakietów za pomocą programu pip. W jednym (sporawym) pakiecie dostaje się to co może być przydatne w trakcie zajęć.

W szczególności w Windows (nie ma tam środowiska Python) dostajemy ten język programowania z wielu dodatkowymi pakietami. SW systemie Linux (tam python zazwyczaj już jest) Anaconda duplikuje wiele pakietów, ale ułatwia instalowanie nowych. W szczególności do różnego rodzaju testów. Wraz z usunięciem Anacondy otrzymujemy nienaruszone orginalne środowisko.

Deinstalacja anacondy usuwa wszystko i można o niej zapomnieć.

D.3. Uruchomienie w laboratorium 604 B1

W przypadku chęci skorzystania z jupytera (i innych narzędzi związanych z Anacondą) należy **jednorazowo** wykonać następujące czynności:

1. Otworzyć terminet (najprościej, na klawiaturze nacisnąć równocześnie trzy klawisze: Ctrl-Alt-T).
2. W terminalu napisać:
   `source /opt/anaconda3/bin/activate`
3. A następnie:
   `conda init`
   i zamknąć terminal (na przykład naciskając klawisze Ctrl-D).

Od tej chwili będzie można już korzystać z anacondy i/lub jupytera. Kolejne otworzenie terminala pokaże zmodyfikowany napis zachęty (*prompt*), który będzie wyglądał jakoś tak:

```
(base) 123456@piwko012:~$
```

Dobrym sosobem na rozpoczęcie może być uruchomienie nawigatora, czyli napisanie w terminalu:

```
anaconda-navigator
```

[2] https://jupyter.org/install
[3] Linux, Window, IOS.

(W chwili obecnej nie można instalować dodatkowych pakietów, ani tworzyć środowisk!)

D.4. Uruchomienie

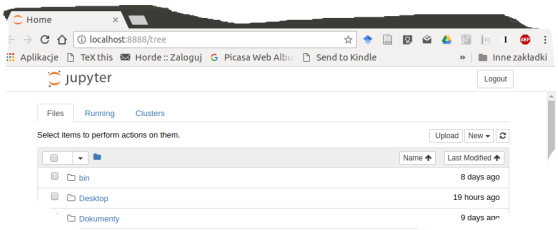
Notatnik jupyter pracuje w układzie serwer (jądro) — klient (klientem jest przeglądarka www). Uruchamiamy go wpisując w terminalu[4]:

```
jupyter notebook
```

Po uruchomieniu jądra, uruchamiana jest przeglądarka (rys. D.1) i automatycznie kierowana pod adres `http://localhost:8888`. W przypadku gdy zamkniemy przeglądarkę, wystarczy w tym oknie, w którym uruchomione jest polecenie `jupyter notebook` nacisnąć równocześnie klawisze Ctrl-C. Próba przerwania jupytera zaowocuje zadaniem pytania czy na pewno przerwać, ale dodatkowo wyświetli również informację pod jakim portem szukać jupuytera:

```
The Jupyter Notebook is running at:
 http://localhost:8888/?token=5ac2de4791858399254a74de588cd4016a381618ce12ce27
 Shutdown this notebook server (y/[n])?
```

(Niestety) ten token jest niezbędny!



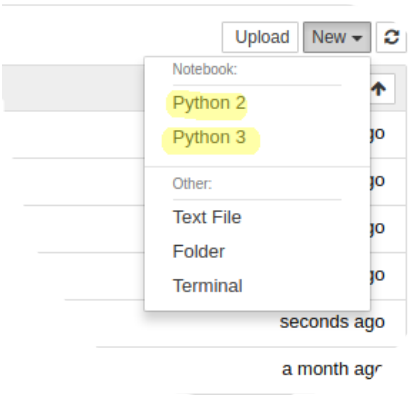
Rysunek D.1. Okno przeglądarki po uruchomieniu notatnika Jupyter

D.5. Praca z notatnikiem

Nowy notatnik tworzony jest wybierając opcję „New” (rys. D.2). W chwili obecnej (w laboratorium) mamy dwie możliwości (dwa języki) do wyboru: Python 2 i Python 3. Na pierwszy rzut oka różnice między tymi dialektami nie są wielkie,

[4] Aby otworzyć terminal najprościej nacisnąć równocześnie trzy klawisze Ctrl-Alt-T albo znaleźć terminal na liście aplikacji.

ale nie każdy, nawet najprostszy kod napisany w Pythonie 2 uruchomi się gdy używamy interpretera Python 3[5].



Rysunek D.2. Tworzenie nowego notatnika

Korzystanie z notatnika jupyter jest bardzo podobne jak w Mathematici. Wpisujemy polecenia w języku, który rozumie jądro, a następnie naciskamy klawisze Shift+Enter. Jądro wykonuje natychmiast polecenie.

W każdej komórce można oprócz polecenia umieścić albo kod albo tekst. Tekst może być formatowany zgodnie z pewnym podzbiorem standardu markdown[6].

Kod musi spełniać wszystkie wymagania używanego języka programowania. Po przełączeniu komórki w tryb *Markdown* można wprowadzać tekst. Będzie on wprowadzany zgodnie z następującymi zasadami:

— Aby zacząć nowy ustęp (paragraf) tekstu należy zostawić co najmniej jedną linię pustą.
— Listy tworzymy w sposób naturalny.
  — Lista numerowana zaczyna się od liczby. Lista nienumerowana zaczyna się od punktora (może być nim znak + albo −.
  — Pod-lista powinna być „wcięta” w stosunku do pozycji macierzystej.
— Można korzystać z tabelek, ale ich tworzenie może być (na pierwszy rzut oka) niezbyt proste. Ale można się posiłkować generatorem tabelek.

[5] Najważniejsze różnice posumowane są na stronie: http://sebastianraschka.com/Articles/2014_python_2_3_key_diff.html. Standardowo dostępny jest program 2to3, który pokazuje różnice ale może też automatycznie dokonać odpowiednich korekt (gdy użyć go z parametrem **-w**): **2to3 -w a.py**.

[6] Pod tym odsyłaczem można znaleźć podstawowe informacje na temat języka Markdown. Swoją drogą Markdown to bardzo wygodny sposób do przygotowywania nawet dosyć rozbudowanych tekstów. Mimo, że wprowadzanie tekstu jest całkowicie tekstowe, można sobie pozwolić na wiele.

— Wzory matematyczne — wprowadzane zgodnie ze standardami LaTeXa. Nie jest on potrzebny (nie musi być zainstalowany) — do wyświetlania używana jest znakomita biblioteka MathJax. Żeby wzory umieszczać — potrzebna jest dosyć elementarna znajomość LaTeXa. Można skorzystać z jednego z wielu dostępnych w sieci serwisów on-line pozwalających wzór „wyklikać". Może to być, na przykład, Sciweavers.
— Obrazki wstawia się zgodnie z zasadami Markdown:

```
![tekst altenatywny](jup_ekran.png "Tytuł obrazka")
```

Tak zwany *rich content* może być bardzo łatwo włączany do notatników Jupyter. Opisuje to dokumentacja.

Najważniejsze skróty klawiaturowe zawiera ściąga.

D.6. Dokumentacja

Ponieważ Jupyter jest dziś produktem bardzo modnym, wszędzie można znaleźć (czasami różnej jakości) przykłady tłumaczące zasady pracy z tym programem. W związku z tym tu, dokumentacja nie zostanie umieszczona.

W pewnym sensie najlepszym źródłem informacji jest serwis bazowy.

D.7. Przykłady

1. Najlepszym źródłem przykładów będzie strona umieszczona w serwisie GitHub jupytera.
2. Warto sięgnąć do książki Python Data Science Handbook której źródła dostępne są jako szereg natatników jupyter umieszczonych (oczywiście) w serwisie GitHub.
3. Wspomnieć też muszę o serwisie Binder pozwalającym na udostępnianie notatników Jupytera on-line. Jest tam też podstawowy zestaw dokumentacji.
4. Na potrzeby zajęć przygotowałem kilka przykładowych notebooków. Na razie nie bardzo wiem jak je udostępnić najsensowniej. Zapewne skończy się utworzeniem własnego kąta w GitHubie. Na razie dostępne są (wraz z niezbędnymi plikami) jako jeden plik ZIP[7].

   Pokazują one rozwiązanie (lub może raczej dają jakiś rodzaj wędki) kilku problemów związanych z tematyką zajęć.

   a) Aproksymacja:
      — Aproksymacja wielomianowa — bardzo podstawowe informacje na temat aproksymacji.

[7] Ściągnięty plik należy gdzieś rozpakować, a następnie w tej kartotece uruchomić notatnik jupytera.

— Przykład realizacji aproksymacji liniowej: Aproksymacja liniowa.

b) Interpolacja:
   — Bardzo podstawowe dane na temat interpolacji: Interpolacja.
   — Coś jeszcze o interpolacji nieudolna próba pokazania w jaki sposób można walczyć z niewłaściwym zachowaniem funkcji interpolacyjnych, które „wariują” na końcach przedziału.

c) Różne:
   — Czytanie danych zawierających ciągi znaków NaN. Zawiera również informacje na temat odrzucania różnych rodzajów błędnych danych.
   — Informacje na temat wczytania do notatnik danych z „drugiego zestawu”[8]. Są tam również informacje na temat proponowanego sposobu „redukcji” danych (resampling).

d) FFT:
   — Szybka transformata Fouriera. Zawiera podstawowe informacje na temat szybkiej transformaty Fouriera.
   — Rozdzielczość transformaty Fouriera. Zadanie do rozwiązania związane z Laboratorium 2a.
   — Zagadka. Drobiazg do przemyśleń na temat tego co, tak na prawdę, pokazuje FFT.
   — Piki Przykład starający się pokazać jak losowość sygnały potrafi „wychodzić” spod zaburzeń.

e) Obliczenia dużej precyzji:
   — Obliczenia dużej precyzji w Pythonie. Przykład do laboratorium nr 4.

f) Optymalizacja
   — Przykład optymalizacji funkcji jednej i dwu zmiennych w Pythonie.

•

[8] Sugeruję zapoznanie się z odpowiednią instrukcją laboratoryjną. Same dane są tutaj.

E. Ciekawe lektury

Bardzo interesujący tekst dotyczący różnych technik (sprzętowych i programistycznych) zawiązanych z przetwarzaniem obrazów przez oprogramowanie telefonów komórkowych [23].

Zwracam uwagę, że ogromna popularność aplikacji fotograficznych dostępnych na telefonach komórkowych (plus ogromna popularność Sztucznej Inteigencji[1]) spowodowały pwastanie bardzo wielu algorytmów wykonujących praktycznie na bieżaco bardzo zaawansowane rzetwarzanie danych.

[1] Będę najczęściej używał skrótu AI (*Artificial Intelligence*), ale również skrót SI (Sztuczna Inteligencja) ma swoje miejsce w literaturze przedmiotu.

Bibliografia

[1] *IEEE standard for floating-point arithmetic*. 2008.

[2] Wikibooks, *C programming/c reference/float.h — wikibooks, the free textbook project*. 2012. [Online; accessed 3-September-2015].

[3] Jankowscy, J.i.M., *Przegląd metod i algorytmów numerycznych. Cz. 1* (Wydawnictwa Naukowo-Techniczne, 1988).

[4] Wikibooks, *Programowanie w systemie UNIX/GMP — wikibooks*. 2015. [Online; accessed 5-wrzesień-2015].

[5] Ferreira, H., *Basics of machine learning and a simple implementation of the naive bayes algorithm*. 2018.

[6] Tadeusiewicz, R., *Uporządkowane wiadomości na temat sztucznej inteligencji*. 2019.

[7] Tadeusiewicz, R., *Nowa kolekcja wiadomości na temat sztucznej inteligencji*. 2019.

[8] Goodfellow, I., Bengio, Y., Courville, A., red., *Deep Learning. Systemy uczące się* (Wydawnictwo Naupowe PWN, Warszawa, 2018).

[9] Koronacki, J., Ćwik, J., *Statystyczne systemy uczące się* (Akademicka Oficyna Wydawnicza EXIT Andrzej Lang, Warszawa, 2015).

[10] Raschka, S., *Python. Uczenie maszynowe* (Helion, 2017).

[11] Peyré, G., *Mathematical Foundations of Data Sciences* (2019).

[12] Zhang, A., Lipton, Z.C., Li, M., Smola, A.J., *Dive into Deep Learning* (2019).

[13] Gallier, J., Quaintance, J., *Algebra, Topology, Differential Calculus, and Optimization Theory For Computer Science and Machine Learning* (2019).

[14] Allen, R., *My curated list of ai and machine learning resources from around the web*. 2017.

[15] *Machine learning*. 2019.

[16] Ward, A., *Some issues on floating-point precision under linux*, Linux Gazette. 2000, , 53.

[17] Johansson, F., *mpmath's documentation*. 2015.

[18] Pilgrim, M., *Zanurkuj w Pythonie* (WikiBooks, 2010).

[19] *Python documentation index*, http://www.python.org/doc/. 2010.

[20] Shaw, Z.A., *Learn Python The Hard Way* (2010).

[21] *Python Programming* (WikiBooks, 2010).

[22] Tagliaferri, L., *How to code in Python 3* (DigitalOcean, 2018).

[23] Zubariew, W., *Fotografia obliczeniowa*. 2020.

[24] Björck, A., Dahlquist, G., *Metody numeryczne* (Państwowe Wydawnictwo Naukowe, 1983).

[25] Borges, L.E., *Python for Developers* (2017).

[26] Brzózka, J., Dorobczyński, L., *MATLAB: środowisko obliczeń naukowo-technicznych* (Wydawnictwo Naukowe PWN, Warszawa, 2008).
[27] Cowlishaw, M., *Decimal arithmetic encodings*. 2009.
[28] Goldberg, D., *Numerical Computation Guide*, rozdz. What Every Computer Scientist Should Know About Floating-Point Arithmetic (Sun Microsystems, Inc., 2000). Patrz również [29].
[29] Goldberg, D., *What every computer scientist should know about floating-point arithmetic*, ACM Computing Surveys (CSUR). 1991, tom 23, 1, str. 5–48.
[30] Jankowscy, J.i.M., Dryja, M., *Przegląd metod i algorytmów numerycznych. Cz. 2* (Wydawnictwa Naukowo-Techniczne, 1988).
[31] Kincaid, D., Cheney, E.W., *Analiza numeryczna* (Wydawnictwa Naukowo-Techniczne, Warszawa, 2005).
[32] Krzyżanowski, P., Plaskota, L., *Metody numeryczne*. Opracowanie programów nauczania na odległość na kierunku studiów wyższych – Informatyka.
[33] Lyons, R.G., *Wprowadzenie do cyfrowego przetwarzania sygnałów* (Wydawnictwa Komunikacji i Łączności, Warszawa, 1997).
[34] Makowiecki, M., *Metody numeryczne fizyki: Interpolacja*, Wikibooks. 2014.
[35] Mrozek, B., Mrozek, Z., *MATLAB i Simulink: poradnik użytkownika* (Helion, Gliwice, 2010).
[36] Pilgrim, M., *Dive Into Python* (Apress, 2004). Dostępne jest polskie tłumaczenie on-line: http://pl.wikibooks.org/wiki/Zanurkuj_w_Pythonie zatytułowane Znurkuj w Pytonie.
[37] Pratap, R., Korbecki, M., *MATLAB 7: dla naukowców i inżynierów* (Wydawnictwo Naukowe PWN, Warszawa, 2010).
[38] Treichel, W., Stachurski, M., *MATLAB dla studentów: ćwiczenia, zadania, rozwiązania* (Witkom (Salma Press) ;, Warszawa, 2009).
[39] VanderPlas, J., *Python Data Science Handbook: Essential Tools for Working with Data*, 1 edition wyd. (O'Reilly Media, 2016).
[40] Waldemar Sradomski, *MATLAB: praktyczny podręcznik modelowania* (Helion, Gliwice, 2015).
[41] Wolfram, S., *An Elementary Introduction to the Wolfram Language* (Wolfram Media, Inc., 2015). Free on the web: http://www.wolfram.com/language/elementary-introduction/.
[42] Drwal, G., red., *Mathematica 5* (Wydaw. Pracowni Komputerowej Jacka Skalmierskiego, Gliwice, 2004).
[43] Ewa Magnucka-Blandzi, red., *Metody numeryczne w MatLabie: wybrane zagadnienia* (Wydawnictwo Politechniki Poznańskiej, Poznań, 2013).
[44] Grzymkowski, R., red., *Mathematica 6* (Wydawnictwo Pracowni Komputerowej Jacka Skalmierskiego, Gliwice, 2008).
[45] Henryk Jerzy Gliński, red., *Mathematica 8* (Wydawnictwo Pracowni Komputerowej Jacka Skalmierskiego, Gliwice, 2012).
[46] Deisenroth, M.P., Faisal, A.A., Ong, C.S., *Mathematics for Machine Learning* (2019).
[47] Peyré, G., *An Introduction to Data Sciences* (2019).

[48] Rosołowski, E., *Metody numeryczne dla inżynierów* (2019).