Techniki obliczeniowe w nauce i technice

Teleinformatyka, rok II, semestr zimowy

prowadzący:

wykład: dr inż. Tomasz Twardowski
ttward@agh.edu.pl

laboratorium: mgr inż. Jacek Wszołek
wszolek@agh.edu.pl

Profil zajęć

- Rozwiązywanie praktycznych problemów techniki z użyciem komputera i algorytmów numerycznych.
- Wykład: podstawy teoretyczne, analiza implementacji, przykłady zastosowań.
  Laboratorium: samodzielne rozwiązywanie problemów.
- Koncentrujemy się na świadomym korzystaniu z gotowych implementacji algorytmów.
- Nie zajmujemy się dowodami matematycznymi i samodzielnym implementowaniem algorytmów.
- Nie wchodzimy w obszar pojęć innych przedmiotów (np. statystyki, DSP, grafiki, elektroniki), ale używamy jako przykładów problemów zaczerpniętych z tych dziedzin.
- Pracujemy w środowiskach programowania: Matlab, język C/C++ z bibliotekami numerycznymi
  - Octave
  - Julia
  - Python + MathLibs + PlotLibs

Tematy wykładów

1. Wprowadzenie do obliczeń numerycznych: historia, zapis zmiennopozycyjny, dokładność obliczeń
2. Macierzowy opis problemów techniki
3. Rozwiązywanie układów równań liniowych
4. Interpolacja wielomianowa, trygonometryczna, funkcje sklejane, ekstrapolacja
5. Całkowanie i różniczkowanie funkcji
6. Aproksymacja i nadokreślone układy równań liniowych
7. Wartości własne i wartości osobliwe – obliczanie i zastosowania
8. Rozwiązywanie równań i układów równań nieliniowych
9. Optymalizacja i minimalizacja funkcji
10. Rozwiązywanie zagadnień początkowych dla równań różniczkowych zwyczajnych
11. Rozwiązywanie zagadnień brzegowych dla równań różniczkowych cząstkowych
12. Generowanie liczb pseudolosowych i metody Monte Carlo
13. Algorytmy wyznaczania wartości funkcji standardowych
14. Obliczenia numeryczne stałoprzecinkowe
15. Biblioteki numeryczne

Literatura uzupełniająca (dla zainteresowanych szerzej tematem)

- Fortuna Z., Macukow B., Wąsowski L., „Metody numeryczne”, WNT 2005
- Guziak T. i inni „Metody numeryczne w elektrotechnice”, Polit. Lubelska 2002
- E. Kącki, A. Małolepszy, A. Romanowicz, „Metody numeryczne dla inżynierów”, Polit. Łódzka 2001

- Mathews J.H., “Numerical Methods for Mathematics, Science, and Engineering”, Prentice Hall 1992
- Hoffman J.D., “Numerical methods for engineers and scientists”, CRC Press 2013
- Bjorck A., Dahlquist G., „Numerical Methods in Scientific Computing”, SIAM 2008
- Hamming R.W., “Numerical Methods for Scientists and Engineers”, Dover Books 1987
- T.K. Moon, W.C. Stirling: Mathematical Methods and Algorithms for Signal Processing. Prentice Hall, 2000
- Press, Flannery, Teukolsky, Vetterling, “Numerical Recipes in (…)”, Cambridge University Press 2007, (dostępna jako dokumenty PDF pod adresem www.nr.com)
- Mathews J.H., “Numerical Methods Using Matlab”, Pearson 2004
- Heath M.T.“Scientific Computing, An Introductory Survey”, McGraw-Hill 2002

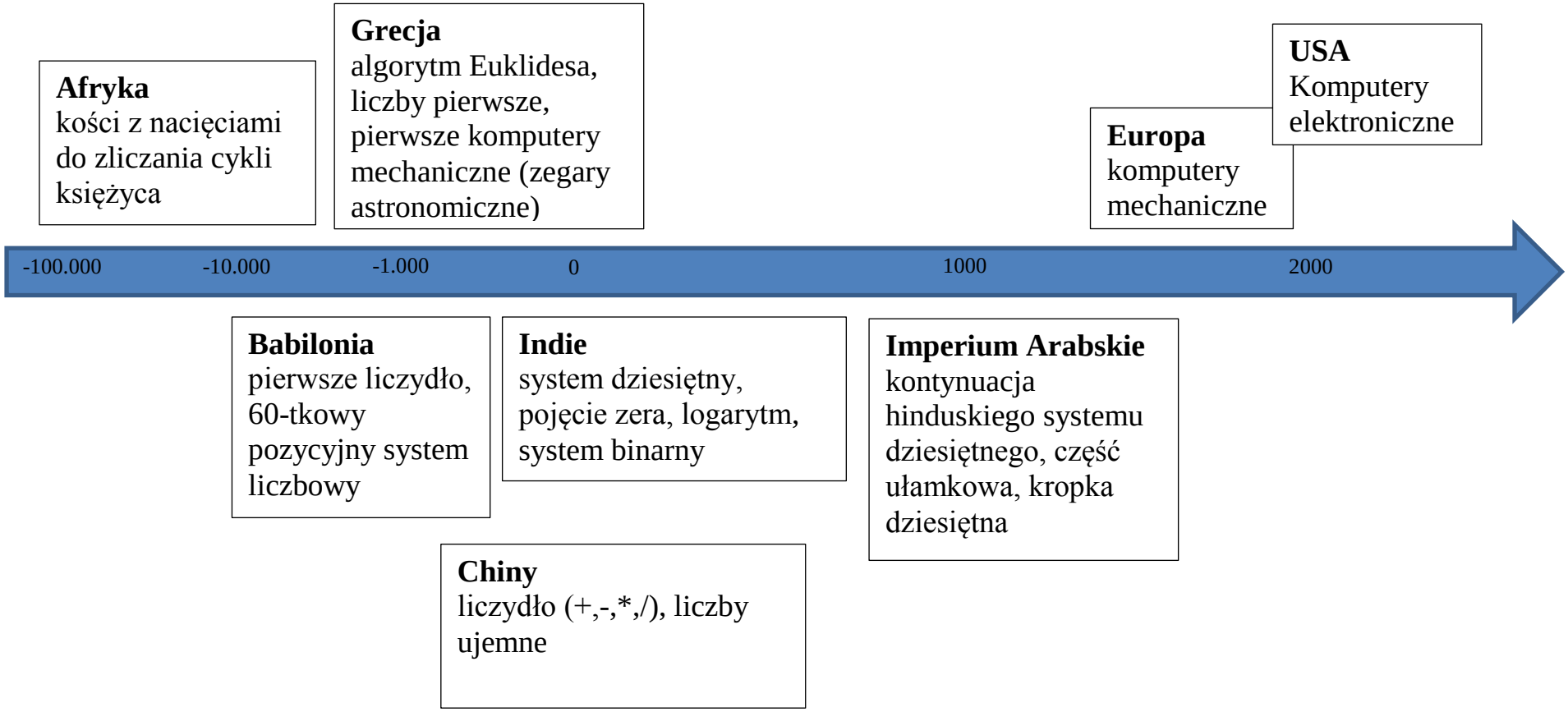
Materiały w Internecie: “scientific computing”, “numerical analysis”, “numerical methods”

Wykład I

Wprowadzenie do obliczeń numerycznych

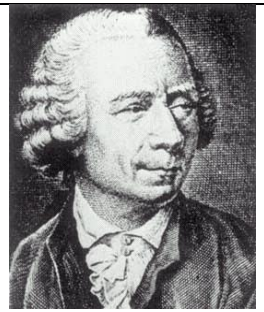
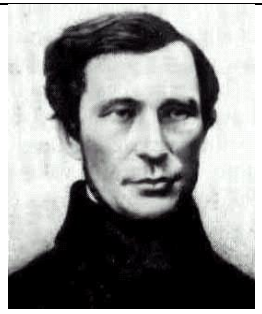
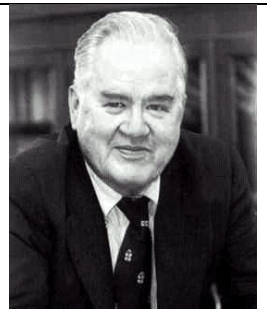
- Historia obliczeń numerycznych
- Współczesne inżynierskie obliczenia komputerowe
- Zapis liczb w pamięci komputera
- Błędy w obliczeniach komputerowych

Historia wynalazków obliczeniowych



Historia obliczeń numerycznych

Podstawy współczesnych obliczeń komputerowych

					
I.Newton (1643-1727)	B.Taylor (1685-1731)	L.Euler (1707-1783)	T.Simpson (1710-1761)	P.S.Laplace (1749-1827)	J.B.J.Fourier (1768-1830)
					
J.F.C.Gauss (1777-1855)	P.L.Czebyszew (1821-1894)	Ch.Hermite (1822-1901)	J.W.S.Rayleigh (1842-1919)	W.M.Kutta (1867-1944)	J.Tukey (1915-2000)

Portrety z serwisu internetowego "MacTutor History Archive"

Współczesne inżynierskie obliczenia komputerowe

Biblioteki numeryczne

LAPACK – algebra liniowa, Fortran,
BLAS (Basic Linear Algebra Subroutines) - szybkie obliczenia macierzowe, Fortran
NAG, C/Fortran/MatlabToolbox
Numerical Recipes in C/Fortran/Pascal/C++
GSL (GNU Scientific Library, free software), C/C++
FFTW – popularna implementacja algorytmów FFT
CUDA – obliczenia numeryczne na kartach graficznych nVidia

Środowiska obliczeń inżynierskich

Zorientowane na obliczenia numeryczne (podstawowy typ to macierz wartości zespolonych):
- Matlab (zbudowany na bazie BLAS i LAPACK, siła w specjalizowanych *toolboxach*)
- Octave, SciLab (klony Matlaba, przenośność programów)

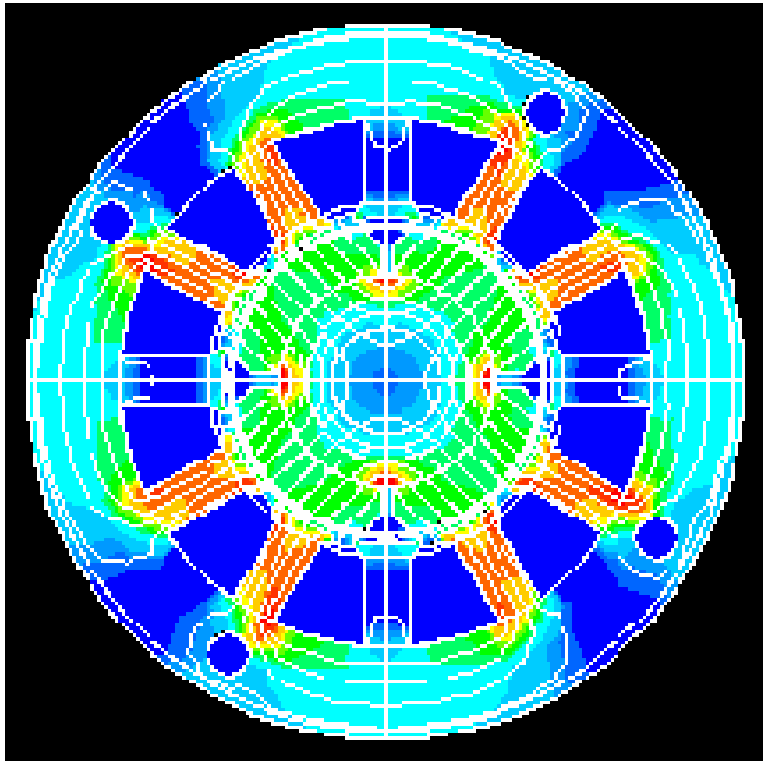
Zorientowane na obliczenia symboliczne:
- Mathematica, Maple

Julia
Python + numPy + sciPy + pyplot

Specjalizowane
- Np. ABAQUS, ANSYS, OPERA, NASTRAN - do rozwiązywania problemów polowych

Współczesne zastosowania obliczeń numerycznych

Modelowanie (generowanie informacji na podstawie znanego opisu)



zdjęcie z http://www.fem.info.gifu-u.ac.jp

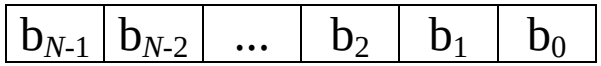
Przetwarzanie danych/informacji ze świata rzeczywistego

zdjęcie z http://www.cedara.com/

Reprezentacja liczb w obliczeniach komputerowych

Omawiane dalej formaty reprezentacji liczb są kolejnymi podejściami do przybliżenia skali liczb rzeczywistych zbiorem skończonej ilości wartości zapisywanych binarnie (bitowo).

Słowo *N*-bitowe:



msb *lsb*

Formaty różnią się między sobą interpretacją poszczególnych bitów.

Przykład: jakie są wartości poniższych liczb reprezentowanych heksadecymalnie (bitowo) ?

16 bitów: $\texttt{ABBA}_h$ (bitowo: $\texttt{1010101110111010}_b$)

32 bity: $\texttt{FFFFFFFF}_h$

64 bity: $\texttt{0010000000000000}_h$

Pytanie: Co musimy wiedzieć, żeby zinterpretować zapis bitowy/bajtowy jako liczbę ?

Odpowiedź: Musimy znać konwencję (umowę) definiującą znaczenie poszczególnych bitów.

Reprezentacja całkowitoliczbowa

KOD	Wartość liczby	Zakres wartości	Rozdzielczość
Bez znaku	$v = \sum_{i=0}^{N-1} 2^i b_i$	$\langle 0, 2^N - 1 \rangle$	1
Znak-moduł	$v = (-1)^{b_{N-1}} \sum_{i=0}^{N-2} 2^i b_i$	$\langle -(2^{N-1}-1), (2^{N-1}-1) \rangle$	1
U2 (inwersja bitów +1)	$v = -2^{N-1} b_{N-1} + \sum_{i=0}^{N-2} 2^i b_i$	$\langle -2^{N-1}, 2^{N-1} - 1 \rangle$	1

Pytanie: jaki kod jest stosowany dla typów języka C *char*, *unsigned*, *int*, *long*, *int32_t*, *uint64_t* ?

Przykład: interpretacja zapisu bitowego w kodzie U2, N=16

waga	-2^{N-1}	2^{N-2}	2^{N-3}	2^{N-4}	2^{N-5}	2^{N-6}	2^{N-7}	2^{N-8}	2^{N-9}	2^{N-10}	2^{N-11}	2^{N-12}	2^{N-13}	2^{N-14}	2^{N-15}	2^{N-16}
bit	1	0	1	0	1	0	1	1	1	0	1	1	1	0	1	0
składnik	-2^{15}	0	2^{13}	0	2^{11}	0	2^9	2^8	2^7	0	2^5	2^4	2^3	0	2^1	0

Wynik (suma składników): **-21574**

Podstawowa niedogodność w tych reprezentacjach (formatach) to brak skalowania, czyli możliwości reprezentacji tym samym typem dużych i małych liczb.

Reprezentacja stałoprzecinkowa (stosowana w niektórych procesorach sygnałowych)

Niedogodność poprzedniego zapisu eliminuje reprezentacja stałoprzecinkowa z ogólnym formatem:

$$v = sq + b$$

gdzie:

s – współczynnik skalujący (*slope*) w formacie $s = f \cdot 2^e$ ($1 \le f < 2$, e ustala pozycję kropki)

b – współczynnik przesuwający (*bias*)

q – właściwy (pamiętany) zapis binarny (bez znaku lub U2)

Obydwa współczynniki nie są częścią zapisu liczby, a jedynie określają konwencję interpretacji zapisu binarnego q.

Przykłady: typowe formaty DSP

radix point scaling (f=1, e dowolne, b=0)

Q1.14, N=16, e=-14, np. q=$\mathrm{FFFF_h}$, $v = 2^{-14}(-1) = -0.00006103515625$

fractionals (f=1, e=-N, b=0)

Q0.8, N=8, np. q=$\mathrm{01010001_b}$, $v = 2^{-8}(2^6 + 2^4 + 2^0) = 1/4 + 1/16 + 1/256 = 0.31640625$

Niedogodność w tym formacie to brak dynamicznego skalowania, czyli możliwości reprezentacji tą samą zmienną dużych i małych liczb z porównywalną precyzją.

Reprezentacja zmiennoprzecinkowa

Taką własność dynamicznego skalowania w inżynierskim zapisie liczb ma tzw. notacja naukowa o ogólnej postaci $v = f \cdot 10^e$, gdzie f jest tak skalowane, że zawiera jedną niezerową cyfrę przed separatorem dziesiętnym. Np. liczba Avogadro (ilość cząsteczek tworząca mol substancji) jest w tej notacji zapisywana jako $6.02252 \cdot 10^{23}[mol^{-1}]$, a ładunek elementarny to $1.602 \cdot 10^{-19}[C]$. W naturalny sposób notację tę można zastosować również przy cyfrach binarnych z podstawą 2. Wtedy: $v = (-1)^s \cdot f \cdot 2^e$, a poszczególne elementy reprezentacji są złożone z bitów:

s	e_{K-1}	e_{K-2}	...	e_1	e_0	f_{L-1}	f_{L-2}			...			f_2	f_1	f_0

s - bit znaku (*sign*): 0 – liczba dodatnia, 1 – liczba ujemna

e - wykładnik (cecha, *exponent*), K bitów, określa dynamikę, $e = \sum_{i=0,\ldots,K-1} e_i 2^i - b$,

Wartości ujemne dla liczb <1, dodatnie dla liczb ≥ 2, przesunięcie (*bias*) $b = 2^{K-1} - 1$.

f - mantysa (*fraction*), L bitów, określa precyzję reprezentacji

Żeby uniknąć niejednoznaczności w reprezentacji liczb mantysa jest normalizowana, tzn. ma postać **1.*f***. Pamiętana jest tylko część po przecinku ***f***. W szczególnym przypadku używana jest specjalnie sygnalizowana (o tym dalej) forma zdenormalizowana **0.*f***.

Całkowita ilość bitów $N=1+K+L$

Własności reprezentacji zmiennoprzecinkowej

Oszacujmy dokładność reprezentacji liczb rzeczywistych postacią zmiennoprzecinkową. Dla czytelności wyników posłużymy się niepraktycznie krótkim, 5-cio bitowym słowem kodowym, z N=5, K=2, L=2, b=1. Przykładowe wartości to (liczby znormalizowane, tj. mantysa formatu 1.f):

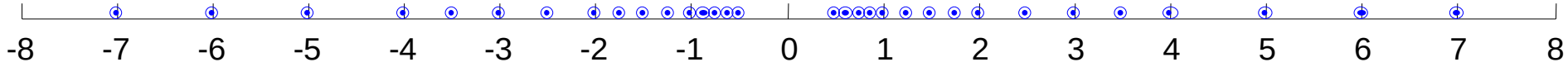
$0\ 01\ 00 = (-1)^0 \cdot 2^{1-1} \cdot (1+0) = 2^0 = 1.0$

$1\ 10\ 11 = (-1)^1 \cdot 2^{2-1} \cdot (1+2^{-1}+2^{-2}) = -2^1 \cdot \left(1+\frac{1}{2}+\frac{1}{4}\right) = -3.5$

$0\ 11\ 01 = (-1)^0 \cdot 2^{3-1} \cdot (1+2^{-2}) = 2^2 \cdot \left(1+\frac{1}{4}\right) = 5.0$

$0\ 00\ 10 = (-1)^0 \cdot 2^{0-1} \cdot (1+2^{-1}) = \frac{1}{2} \cdot \left(1+\frac{1}{2}\right) = 0.75$

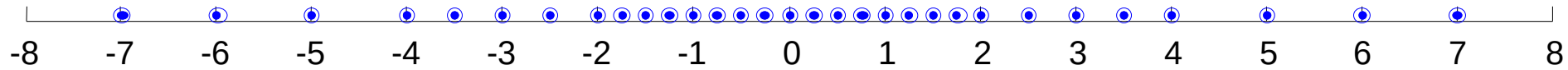
Wartości reprezentowane tym słowem kodowym przedstawiono na osi liczb rzeczywistych:



Wnioski:

- Odstęp pomiędzy wartościami nie jest stały, jest większy przy większych liczbach
- Zero i wartości wokół zera nie są reprezentowane

Ulepszenie: Po denormalizacji (sygnalizowana przez e=0, wtedy mantysa 0.f i skalowanie 2^{1-b}):



IEEE Standard 754 (1985)
Definiuje reguły zapisu reprezentacji zmiennoprzecinkowych o pojedynczej (*single*) i podwójnej (*double*) precyzji. Reguły te są powszechnie używane w sprzęcie i oprogramowaniu obliczeniowym. W języku C to typy *float* i *double*, w Matlabie domyślna jest podwójna precyzja.

Standard określa rozmiary N, K i L, które determinują wartości w pozostałych kolumnach.

Format	realmin (norm)	realmax	bias b	precision (eps)
single (N=32, K=8, L=23)	$2^{-126}\approx10^{-38}$	$(2-2^{-23})2^{127}\approx3\cdot10^{38}$	127	$2^{-23}\approx10^{-7}$
double (N=64, K=11, L=52)	$2^{-1022}\approx10^{-308}$	$(2-2^{-52})2^{1023}\approx1.7\cdot10^{308}$	1023	$2^{-52}\approx10^{-16}$

Przykłady dla podwójnej precyzji w zapisie heksadecymalnym (z użyciem **num2hex**):

1) najmniejsza liczba znormalizowana (realmin): `00 10 00 00 00 00 00 00`
2) największa liczba rzeczywista (realmax): `7f ef ff ff ff ff ff ff`
3) pierwsza liczba większa od 1 (1+eps): `3f f0 00 00 00 00 00 01`
4) –1: `bf f0 00 00 00 00 00 00`

liczby denormalizowane (sygnalizowane przez e=-1023):

5) 0: `00 00 00 00 00 00 00 00`
6) realmin/2: `00 08 00 00 00 00 00 00`

wyjątki (sygnalizowane przez e=1024):

7) Inf: `7f f0 00 00 00 00 00 00`
8) NaN: `ff f8 00 00 00 00 00 00`

Przykład: parametry typu float (single precision) w języku C

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

int main(void)
{
    float eps=1.0f;
    float v=1.0f+eps;
    uint32_t* p;
    while(v>1.0f)
    {
        p=(uint32_t*)&v;
        printf("f=%.8f (%08x)\n", v, *p);
        eps/=2.0f;
        v=1.0f+eps;
    }
    *p=0x7f7fffff;   printf("MAX \t f=%.8e \t (%08x)\n", v, *p);
    *p=0x007fffff;   printf("MIN \t f=%.8e \t (%08x)\n", v, *p);
    *p=0x7f800000;   printf("+Inf \t f=%.8e \t (%08x)\n", v, *p);
    *p=0xff800000;   printf("-Inf \t f=%.8e \t (%08x)\n", v, *p);
    *p=0x7fffffff;   printf("NaN \t f=%.8e \t (%08x)\n", v, *p);
    return 0;
}
```

Błędy w obliczeniach numerycznych

Źródłem błędów obliczeniowych związanym z obliczeniami komputerowymi jest przybliżona reprezentacja liczb rzeczywistych w postaci zmiennoprzecinkowej. Błąd związany z tym przybliżeniem zależy od konkretnej wartości, ale możemy podać jego wartość graniczną.

Zaokrąglanie liczby rzeczywistej x do najbliższej liczby reprezentowalnej zmiennoprzecinkowo $\tilde{x}$ (zobacz wcześniejszy rysunek z osią reprezentowalnych wartości znormalizowanych) wiąże się z błędem względnym:

$$\varepsilon = \left|\frac{x-\tilde{x}}{x}\right| = \left|\frac{(-1)^s 2^e \left(\left[1+\sum_{i=1}^{\infty} f_i 2^{-i}\right] - \left[1+\sum_{i=1}^{K} f_i 2^{-i}\right]\right)}{(-1)^s 2^e \left[1+\sum_{i=1}^{\infty} f_i 2^{-i}\right]}\right| = \left|\frac{\sum_{i=K+1}^{\infty} f_i 2^{-i}}{1+\sum_{i=1}^{\infty} f_i 2^{-i}}\right| \le 2^{-(K+1)} = eps/2$$

W dalszej analizie dokładności algorytmów numerycznych będziemy przyjmować względne „zaburzenie" danych, tj. $\tilde{x} = x(1+\varepsilon)$. To **małe zaburzenie** (na poziomie eps/2) może w przypadku niektórych zadań obliczeniowych prowadzić do **dużych błędów wyniku** obliczeń. Tak jest w przypadku wyznaczania różnicy dwóch bliskich dużych liczb (*catastrophic cancelation*):

$$\frac{\left|[x-y]-[x(1\pm\varepsilon)-y(1\pm\varepsilon)]\right|}{|x-y|} \le \frac{\varepsilon(|x|+|y|)}{(x-y)}$$ mała różnica liczb → względny błąd wyniku DUŻY !

Uwarunkowanie zadania obliczeniowego

Definicja (intuicyjna)
Jest to wrażliwość zadania obliczeniowego na zaburzenie danych wyrażana jako wzmocnienie propagacji tych zaburzeń danych na zaburzenie wyniku obliczeń (przenoszenie błędu).

Pierwszy przykład zadania źle uwarunkowanego

Różnica dwóch bliskich liczb pojawia się przy wyznaczaniu pierwiastków równania kwadratowego $ax^2+bx+c=0$ ze „szkolnego” wzoru: $x_1=\frac{-b-\sqrt{b^2-4ac}}{2a}$ $x_2=\frac{-b+\sqrt{b^2-4ac}}{2a}$

Licznik tych wyrażeń w przypadku, gdy $|b| \gg |ac|$, jest różnicą bliskich liczb (wzór na x_1 dla $b<0$, x_2 dla $b>0$). Zaburzenie b będzie skutkować dużym błędem wyznaczenia pierwiastka.

Zadania trudne obliczeniowo można rozwiązać z zadowalającą dokładnością **stosując większą precyzję** obliczeń (np. zmiana typu z *float* na *double*) lub **stosując zadanie równoważne** ale mniej wrażliwe na zaburzenie danych.
Np. powyższe zadanie można również rozwiązać stosując równoważne wzory:

$$x_1=\frac{-2c}{b-\sqrt{b^2-4ac}} \qquad x_2=\frac{-2c}{b+\sqrt{b^2-4ac}}$$

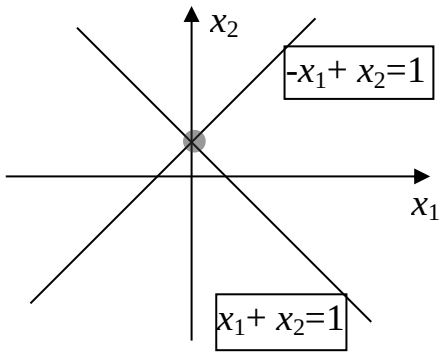
Wybierając, zależnie od wartości b, wersję wzoru uzyskujemy dokładniejsze rozwiązanie.

Drugi przykład zadania źle uwarunkowanego

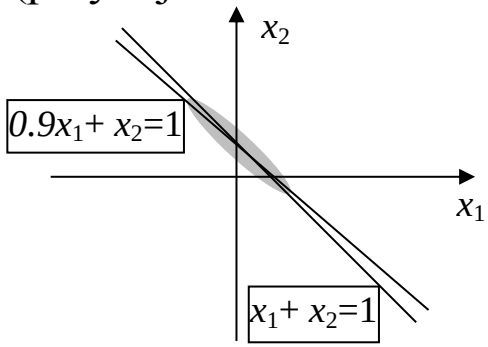
Układy równań liniowych z macierzą bliską osobliwej to klasyczny przykład zadania źle uwarunkowanego. Rozważmy układ dwóch równań liniowych z parametrem *a*:

$\begin{cases} x_1 + x_2 = 1 \\ a x_1 + x_2 = 1 \end{cases}$ co można zapisać macierzowo jako $\underbrace{\begin{bmatrix} 1 & 1 \\ a & 1 \end{bmatrix}}_{\mathbf{A}} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$. Dla a=1 układ jest zależny,

określa tę samą prostą w przestrzeni (x_1, x_2), i nie posiada jednego rozwiązania. Można podejrzewać, że dla *a* bliskiego 1 zadanie będzie wrażliwe na zaburzenie danych (tj. współczynników równań). Potwierdzają to podejrzenie poniższe rysunki, na których zaznaczono obszar rozrzutu wyznaczonych graficznie (przy tej rozdzielczości rysunku) rozwiązań.



a=-1



a=0.9

Próba rozwiązania układu metodą wyznaczników ujawnia znany już powód złego uwarunkowania w postaci różnicy bliskich liczb ($1/\det[\mathbf{A}] = 1/(1-a)$).

Trzeci przykład zadania źle uwarunkowanego

Oscylator programowy – generator przebiegu sinusoidalnego

Do generowania kolejnych wartości funkcji $y_n = \sin(n\omega_0)$ możemy użyć schematu iteracyjnego:

$$y_n = a y_{n-1} - y_{n-2}, \qquad a = 2\cos(\omega_0)$$

gdzie: $\omega_0 = 2\pi f_0/f_s$, $y_{-1} = \sin(\varphi - \omega_0)$, $y_{-2} = \sin(\varphi - 2\omega_0)$, φ - faza początkowa

Próba generowania sygnału sinusoidalnego 20Hz i 40Hz na karcie dźwiękowej o próbkowaniu 192kHz ze 16-bitowym zapisem Q2.14 współczynnika a daje niepożądany efekt:

$$a_{20Hz} = 2\cos\left(2\pi \frac{20}{192000}\right) \cdot 2^{14} = 32768, \qquad a_{50Hz} = 2\cos\left(2\pi \frac{40}{192000}\right) \cdot 2^{14} = 32768$$

Generowana jest taka sama częstotliwość ! Rozdzielczość wyniku (częstotliwości) w pobliżu zera jest o rząd gorsza niż rozdzielczość wielkości wejściowej (współczynnika).

Jaka jest przyczyna niepoprawnego zachowania generatora ? Czy można to ulepszyć ?
Temat na ćwiczenia. Zobacz np. http://users.abo.fi/htoivone/courses/sbappl/asp_chapter1.pdf

Taki generator sinusa to szczególny przypadek filtra IIR z biegunami blisko koła jednostkowego. Zobacz „Digital Signal Processing in Communication Systems”, M. Frerking, 1994.

Wskaźnik uwarunkowania

Wskaźnik uwarunkowania (*condition number*) określa liczbowo trudność obliczeniową zadania. Jest definiowany jako stosunek względnego zaburzenia wyniku do powodującego go względnego zaburzenia danych.

Oznaczając:

x - dane zadania, $\tilde{x}$ - zaburzone dane zadania,

$y = f(x)$ – rozwiązanie zadania powiązane zależnością $f()$ z danymi x,

$\tilde{y} = f(\tilde{x})$ - zaburzone rozwiązanie zadania dla zaburzonych danych,

$cond_f(x) = \left\| \frac{\frac{f(x)-f(\tilde{x})}{f(x)}}{\frac{x-\tilde{x}}{x}} \right\| \approx \left\| \frac{x \frac{df(x)}{dx}}{f(x)} \right\|$. W przypadku skalarnym normą jest moduł: $\boxed{cond_f(x) \approx \left| \frac{x \frac{df(x)}{dx}}{f(x)} \right|}$

Przykład: uwarunkowanie zad. wyznaczania pierwiastków równania kwadratowego (zał. a=0.5):

$\frac{dx_1}{db} = -1 - \frac{b}{\sqrt{b^2-2c}}$ $\quad cond_{x_1}(b) \approx \left| \frac{b}{\sqrt{b^2-2c}} \right|$. Dla b=1, $c \to 1/2$: $cond_{x_1}(b) \to \infty$

(w wielu książkach - wrażliwość wielomianu rzędu 20 z zerami całkowitymi – do poczytania).

Wskaźnik uwarunkowania dla zadań macierzowych, jak w drugim przykładzie, zależy od własności przetwarzanych macierzy. To będzie omówione na następnym wykładzie.

Podsumowanie, kluczowe elementy tego wykładu:

- Obliczenia numeryczne nie narodziły się razem ze współczesnym komputerem, były stosowane od zarania dziejów
- Współczesne metody numeryczne = matematyka (algebra) + komputer
- Komputer, dzięki szybkości działania, umożliwia wykonywanie bardziej złożonych obliczeń
- Liczby rzeczywiste w komputerze nie są reprezentowane dokładnie
- Reprezentacja zmiennoprzecinkowa daje stałą dokładność względną
- Maksymalny błąd reprezentacji zależy od ilości bitów typu zmiennoprzecinkowego
- Dokładność względna dla typu double to 16 miejsc po przecinku
- Błąd reprezentacji danych wejściowych propaguje się na wynik obliczeń
- Czasami błąd jest znacznie wzmacniany, zależy to od struktury zadania obliczeniowego
- W takim przypadku możemy ograniczyć błąd wyniku przez zwiększenie precyzji danych (zmiana typu na dłuższy) lub przez przeformułowanie zadania na równoważne ale mniej wrażliwe