Obliczenia Naukowe

O arytmetyce komputerów,
Czyli jak nie dać się zaskoczyć

Bartek Wilczyński
bartek@mimuw.edu.pl

27. lutego 2020

Plan semestru

- Arytmetyka komputerów, wektory, macierze i operacje na nich
- Kreślenie funkcji, graficzna reprezentacja danych
- Podstawy przetwarzania sygnałów, filtry, reprezentacja obrazów
- Kompresja danych, przykłady kompresji stratnej i bezstratnej
- Rozwiązywanie układów równań liniowych, metoda najmniejszych kwadratów
- Wartości własne i wektory własne macierzy
- Interpolacje i aproksymacje funkcji
- Rozwiązywanie równań nieliniowych
- Rozwiązywanie równań różniczkowych, kwadratury
- Obliczenia symboliczne

Założenia i wymagania

- Mamy zapoznać się z narzędziami ułatwiającymi dokonywanie obliczeń w programach komputerowych
- Przy okazji szlifujemy nasze umiejętności programistyczne
- Celem jest praktyczna znajomość narzędzi przy jednoczesnej znajomości podstaw działania metod numerycznych
- Opieramy się głównie na języku python (w wersji 3)
- Podstawą do zaliczenia będą **projekty programistyczne** wykonywane samodzielnie, **kolokwium** oraz **egzamin**

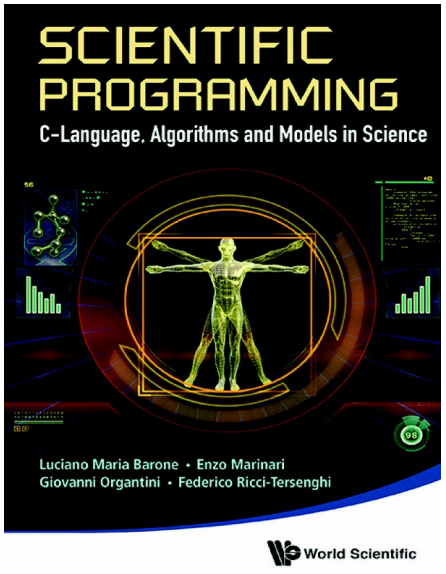
Kryteria zaliczenia

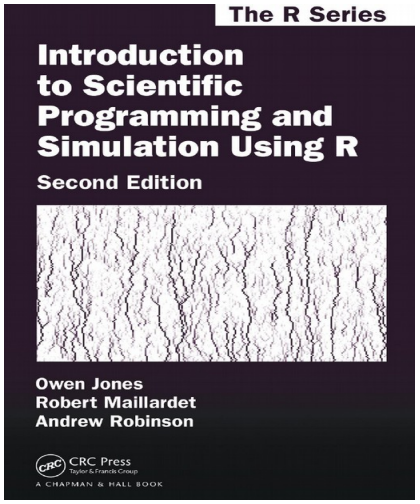
- I Projekt – do 25. III – 10 pkt
- II Projekt – do 30. IV – 15 pkt
- III Projekt – do . V – 25 pkt
- Za projekty oddane po terminie – max. 50% punktów
- Kolokwium – 4. VI – 15 pkt
- Egzamin pisemny – 35 pkt
- Do zaliczenia potrzeba 60 punktów
- Zadanie bonusowe – ok. 5 punktów

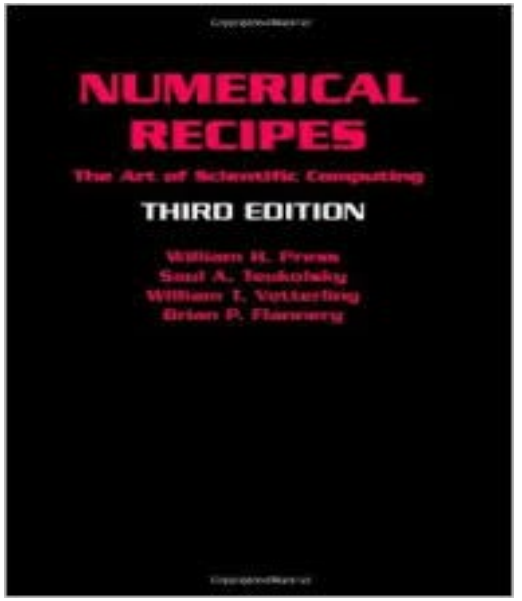
Materiały

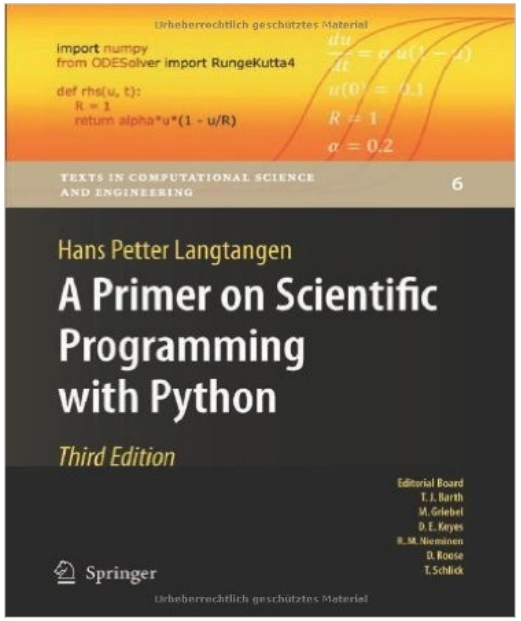
- Dostępnych jest wiele książek pokrywających tematykę metod numerycznych
- Będziemy też korzystać z materiałów online: http://smurf.mimuw.edu.pl/?q=metody_numeryczne
- Moje slajdy i zadania publikowane będą na stronie www:

  http://regulomics.mimuw.edu.pl/wp/category/teaching/ona/









O co chodzi z tą arytmetyką?

- Wiedzą Państwo z zajęć WDI w poprzednim semestrze, że komputery dysponują skończoną pamięcią
- Skądinąd, wiedzą Państwo, że liczby mogą być dowolnie duże, więc nie każda liczba zmieści się w pamięci komputera
- Stąd potrzebujemy ustalić reguły jak wykonywać obliczenia w komputerze, mimo pewnych nieuchronnych ograniczeń
- Dobrze byłoby, gdyby zbyt często nie powodowały one zbyt dużych błędów

Pomyłki w obliczeniach?

- W 1991 r. wyrzutnia rakiet patriot w Arabii Saudyjskiej chybiła celu, w związku z czym zginęło 28 żołnierzy, powód – niedokładna reprezentacja liczby 0.1 w komputerze
- W 1995 r. rakieta Ariane rozpadła się podczas startu z powodu przepełnienia wartości numerycznej
- W 1991 r. zapadła się platforma wiertnicza Sleipner, z powodu błędu w obliczeniach konstrukcyjnych (niedoszacowanie obciążenia o 47% z powodu błędu numerycznego).
- Sonda marsjańska MCO rozbiła się zamiast wylądować w 1999
- Więcej:

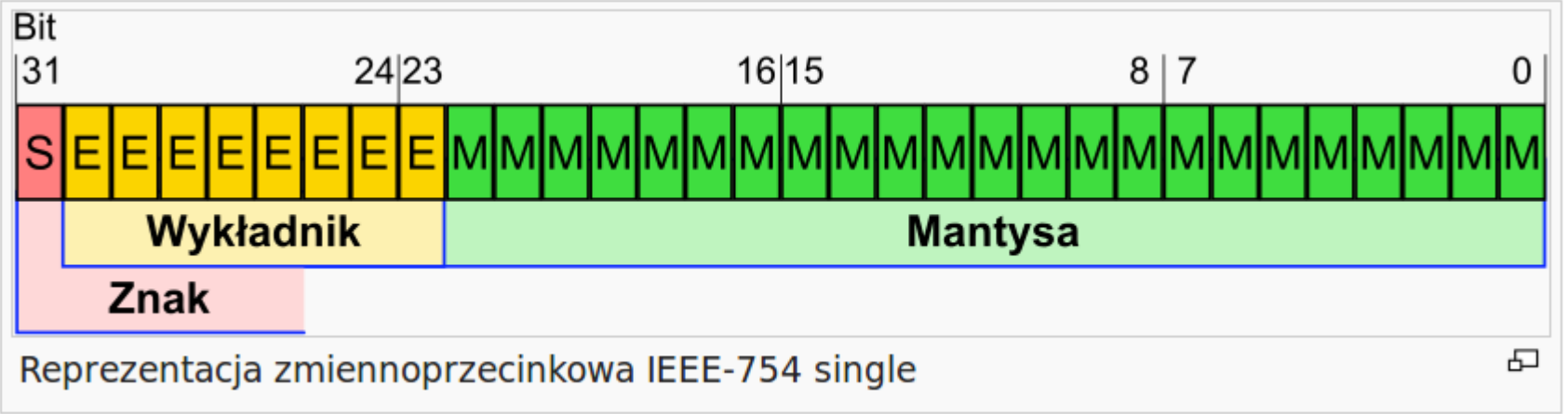
  https://www.ima.umn.edu/~arnold/disasters/disasters.html

Jak komputer liczy?

- Ile to będzie
  - 2019/10e309?
  - 2.019/10e306 ?
  - (2019/1000) / (10e309/1000) ?
  - 2019/10**309 ?
  - 2019/10.**309 ?
- A co z ułamkami?
  - 0.1 ?
  - 0.1 + 0.1 ?
  - 0.1 + 0.1 + 0.1 ?
  - 3 * 0.1 ?
  - 5 * 0.1 ?

Cecha i mantysa

- Liczby możemy reprezentować jako ciąg bitów:



- Wartością takiej liczby jest $(-1)^S * M..M^{E..E}$
- Ta liczba ma reprezentację binarną (nie dziesiętną)
- Ułatwia nam to operacje na takich liczbach

Standard IEEE-754

- Standard IEEE-754 (ostatnia wersja z 2008r.) opisuje jak zapisywane są liczby w większości procesorów na świecie
- Dodatkowo, mamy tu różne precyzje liczb i obciążenie (ang. “bias”), które pozwala łatwiej porównywać liczby w różnych reprezentacjach

Name	Common name	Base	Digits	Decimal digits	Exponent bits	Decimal E max	Exponent bias[6]	E min	E max	Notes
binary16	Half precision	2	11	3.31	5	4.51	$2^4-1=15$	−14	+15	not basic
binary32	Single precision	2	24	7.22	8	38.23	$2^7-1=127$	−126	+127	
binary64	Double precision	2	53	15.95	11	307.95	$2^{10}-1=1023$	−1022	+1023	
binary128	Quadruple precision	2	113	34.02	15	4931.77	$2^{14}-1=16383$	−16382	+16383	
decimal32		10	7	7	7.58	96	101	−95	+96	not basic
decimal64		10	16	16	9.58	384	398	−383	+384	
decimal128		10	34	34	13.58	6144	6176	−6143	+6144	

Problemy arytmetyki zmiennoprzecinkowej

- Błąd reprezentacji:
  - Epsilon maszynowy
  - Nadmiary i niedomiary
- Podczas wykonywania działań
  - Utrata cyfr znaczących podczas dodawania
  - Utrata cyfr znaczących podczas odejmowania
- Porównania liczb zmiennoprzecinkowych dają niespodziewane wyniki...

Epsilon maszynowy

- Warto zauważyć, że każda arytmetyka zmiennoprzecinkowa może reprezentować jedynie skończenie wiele liczb, więc istnieją “przerwy” pomiędzy nimi
- epsilon maszynowy to najmniejsza liczba, którą można “efektywnie” dodać do jedynki w danej realizacji arytmetyki
- Odległość między najbliższymi liczbami reprezentowanymi w tej arytmetyce w okolicy liczby x, to szacunkowo 2*epsilon*|x|

Wartości specjalne

- Co się stanie, jeśli chcemy reprezentować liczby większe niż możemy zmieścić w reprezentacji bitowej? → wartości *Inf* oraz *-Inf*
- A co, jeśli liczby te będą za małe? → 0.0
- Co jeśli nadal będziemy wykonywać operacje na liczbach “nieskończonych”? → Not a Number, czyli *NaN*
- To nie musi dziać się z liczbami całkowitymi: w pythonie, liczby całkowite mają dużo większą precyzję.

“Lepsze” reprezentacje liczb w pythonie

- Liczby całkowite w pythonie 3 – mogą być arbitralnie duże (tak jak bignum w pythonie 2.x). Oczywiście pamięć nadal nas ogranicza.
- Pakiet **Decimal**, opisujący liczby dziesiętne w formacie cecha+mantysa w zadanej dokładności
- Pakiet **Fractions** – opisujący liczby wymierne jako ułamki liczb całkowitych
- Oba działają znacznie wolniej od arytmetyki IEEE-754

Pakiet mpmath

- Zewnętrzny względem python'a pakiet liczb zmiennoprzecinkowych
- Dowolna precyzja, zoptymalizowany kod
- Dodatkowe funkcje pozwalające na mniejsze błędy zaokrągleń (zwiększenie precyzji podczas wykonywania operacji).