Pakiety matematyczne MAT1349 - 2021

Lista 4: Inne zastosowania tablic

1. **Wyświetlacz liczb binarnych.** Stwórz macierze, których komórki będą się układały we wzór cyfr ”1” lub ”0”. Używając ich napisz program, który będzie pokazywał na ekranie kolejne liczby w systemie binarnym niczym prosty wyświetlacz cyfrowy. Możesz też stworzyć ich animację.

   Opcjonalne: Zaimplementuj wyświetlanie na ekran dodawania oraz odejmowania pisemnego liczb binarnych, tzn. w górnej linii wyświetli liczbę binarną a, niżej $+$ lub $-$ liczbę binarną b, niżej poziomą kreskę i wynik operacji, również binarnie. Może uda się rownież zaimplementować mnożenie?

2. **Znajdowanie optymalnej trasy.** Niech tablica $n \times m$ opisuje prostokątny fragment trudnego do przebycia terenu, a wartości w jej komórkach niech symbolizują koszt/ilość paliwa potrzebną do przebycia tego fragmentu obszaru. Chcemy dostać się z dolnego wiersza do górnego wiersza najmniejszym możliwym kosztem. Poruszamy się ruchem pionka szachowego. Znajdź algorytm znajdujący optymalną trasę oraz jej koszt i wypróbuj go dla tablic, gdzie optymalna trasa jest oczywista oraz tablic z losowymi wartościami.

   Podpowiedź: Zacznij od kosztów wejścia na komórki z dolnego wiersza. Na tej podstawie znajdź najmniejszy koszt dostania się do każdej z komórek wiersza wyżej, po czym wiersza wyżej, i tak aż do górnego wiersza. W ten sposób znajdziesz najmniejszy koszt dotarcia do górnego wiersza i możesz również odtworzyć odpowiadającą mu trajektorię. Ten sposób rozwiązywania złożonych problemów poprzez zapamiętywanie rozwiązań prostszych podproblemów nazywamy *programowaniem dynamicznym.*

   Opcjonalnie: Rozwiąż ten problem, kiedy poruszamy się ruchem skoczka szachowego, ale bez cofania się. (Z możliwością cofania się problem jest dużo trudniejszy.)

3. **Numeryczne aspekty rozwiązywania równań liniowych.** Będziemy badać stabilność oraz szybkość numerycznego rozwiązywania równań liniowych. W tym celu dla ustalonej macierzy A oraz wektora x obliczamy $y = Ax$, po czym mając y staramy się odtworzyć x rozwiązując ten liniowy układ równań. Będziemy używali pakietu `LinearAlgebra`.

   (a) **Stabilność.** Rozwiązywanie układów równań staje się trudne dla macierzy o wyznaczniku bliskim numerycznego 0 lub $\pm\infty$. Dla takich przypadków funkcja `cond` mierzy dla danej macierzy, ile w przybliżeniu razy przy rozwiązwaniu układu równań zwiększają się niedokładności numeryczne. Sprawdź dla *źle uwarunkowanych* macierzy (mówimy o `cond` rzędu dziesiątek lub setek tysięcy) na ile odtworzony x będzie bliski oryginalnemu. Sprawdź też, jak wynik zależy od typu, wypróbuj `Float16, Float64, BigFloat, Rational{Int64}, Rational{BigInt}`.

Przykłady źle uwarunkowanych macierzy to macierze Hilberta $H_{ij} = 1/(i+j-1)$ oraz duże macierze z losowymi wartościami.

Uwaga: Ostrożnie testuj coraz większe macierze, obliczenia szybko mogą stać się bardzo wymagające. Obliczenia w Julii można zabić Ctrl+C.

(b) **Szybkość.** Porównaj szybkość rozwiązywania układów równań używając różnych rodzajów tablic. Dla małych macierzy porównaj tablice zwykłe oraz statyczne. Dla dużych tablic wypełnionych w większości zerami porównaj tablice zwykłe oraz rzadkie.

Uwaga: Dla dużych tablic z wieloma zerami musisz zagwarantować, że tablica jest odwracalna. Najłatwiej wpisać jedynki na przekątnej, po czym w nieliczne inne komórki wpisać losowe liczby.

4. **Przetwarzanie obrazów.** Zainstaluj pakiety `Images`, `ImageMagick`, `TestImages`. W pierwszych dwóch są metody otwierania oraz przetwarzania obrazów, w ostatniej kolekcja przykładowych obrazów, ich lista jest na

https://testimages.juliaimages.org/

Testowe obrazy ładujemy komendą `testimage("nazwa")`. Julia w VS Code oraz w Jupyterze powinna wyświetlić je automatycznie. Będzie nas interesowało przetwarzanie tylko prostych obrazów w skali szarości, możemy je uzyskać broadcastem rzutowania `Gray{Float64}`. Taką macierz możemy przetworzyć na tablicę liczbową broadcastem rzutowania `Float64`, w drugą stronę z powrotem broadcastem rzutowania `Gray`.

(a) Spróbuj sztucznie zaszumić obrazy dodając do pikseli różnego rodzaju losowe wartości. Sprawdź efekty.

(b) Przetwórz piksele różnymi funkcjami $[0,1] \mapsto [0,1]$ i sprawdź efekty. Propozycje: $4(x-1/2)^3+1/2$, $1-x$, skoki jednostkowe, inne?

(c) Zastąp każdy piksel średnią z pikseli z kwadratu wokół niego o ustalonej wielkości. Jaki ma to wpływ na obraz i zaszumiony obraz?

(d) Sprawdź jaki będzie efekt nałożenia na obraz (znanego nam już) dyskretnego laplasjanu. W 2D jest równy

$$\nabla^2_{i,j} f_{i,j} = f_{i+1,j} + f_{i-1,j} + f_{i,j+1} + f_{i,j-1} - 4f_{ij}.$$

A co stanie się, kiedy nałożymy laplasjan 1D wzdłuż wierszy lub wzdłuż kolumn?

W przypadkach kiedy korzystasz z sąsiadów pikseli nie musisz pisać specjalnego kodu dla brzegów obrazu, możnesz je ominąć.

Opcjonalne: Komendą `histogram(tab[:])` z `Plots` możesz zwizualizować, ile pikseli w tablicy `tab` ma jaki poziom szarości. Sprawdź to dla oryginalnych i przetworzonych obrazów. Potrafisz wyjaśnić obserwowane zmiany?