Program 3 - Faktoryzacja LU

April 13, 2023

Program 3: Faktoryzacja *LU*

Autorzy: Maciej Czyjt, Tomasz Słonina

Dołączenie potrzebnych bibliotek:

```
using LinearAlgebra
```

Plik *matrix_utils.jl* zawiera funkcje z poprzednich programów, potrzebne do zrealizowania faktoryzacji: rekurencyjne mnożenie oraz odwracanie macierzy.

```
include("matrix_utils.jl")
```

LU Factorization

Implementację faktoryzacji LU opisuje pseudokod:

```
// input: A
// output: L, U

function lu_factor_algorithm(A, L, U) {
    if (A.size > 2) {
        a11, a12, a21, a22 = A[range_A].slice_into_quarters();
        L11, U11 = lu_factor_algorithm(a11);
        L21 = a21 × inverse(U11);
        U12 = inverse(L11) × a12;

        S = a22 - (a21 × inverse(U11) × inverse(L11) × a12);

        L22, U22 = lu_factor_algorithm(S);
    } else {
        L, U = lu(A);
    }
}
```

Poniżej właściwa implementacja:

```
function lu_factorization(A)
    len = size(A, 1)
    L = zeros(len, len)
    U = zeros(len, len)
    lu_factorization_(A, L, U)
```

```
    return L, U
end


function lu_factorization_(A, L, U)
    if size(A, 1) == 2
        l, u = lu(A, NoPivot())
        L[:, :] = l
        U[:, :] = u
        return
    end
    len = (size(A, 1) ÷ 2)
    a11 = [1:len, 1:len]
    a12 = [1:len, len+1:len*2]
    a21 = [len+1:len*2, 1:len]
    a22 = [len+1:len*2, len+1:len*2]

    L11, U11 = lu_factorization(A[a11...])
    inv_U11 = my_inv(U11)
    inv_L11 = my_inv(L11)

    L21 = multiply_rec(A[a21...], inv_U11)
    U12 = multiply_rec(inv_L11, A[a12...])
    S = A[a22...] - multiply_rec(
        A[a21...],
        multiply_rec(
            inv_U11,
            multiply_rec(
                inv_L11,
                A[a12...]
            )
        )
    )
    L22, U22 = lu_factorization(S)

    L[a11...] = L11
    L[a21...] = L21
    L[a22...] = L22
    U[a11...] = U11
    U[a12...] = U12
    U[a22...] = U22
end
```

Testy

Funkcja służąca do testowania implementacji:

```
function test_lu(N)
    for n in 1:N
        A = rand(2^n, 2^n)
        L, U = lu_factorization(A)
        if isapprox(A, L * U)
            println("Test passed for matrix of size ", 2^n, "x", 2^n)
        else
            println("Test failed for matrix of size ", 2^n, "x", 2^n)
        end
    end
end

test_lu(9)
```

Poniżej zostało przedstawione porównanie wyników zaimplementowanej faktoryzaji oraz wersji bibliotecznej. Brane pod uwagę były macierze o rozmiarze 2^k, dla parametru $k = 1, 2, ..., 9$.

```
function lu_det(A)
    L, U = lu_factorization(A)
    det = 1
    for i in 1:size(A, 1)
        det *= L[i, i] * U[i, i]
    end
    return det
end

lu_dets = []
true_dets = []

N = 9
for n in 1:N
    A = rand(2^n, 2^n)
    push!(lu_dets, lu_det(A))
    push!(true_dets, det(A))
end
```

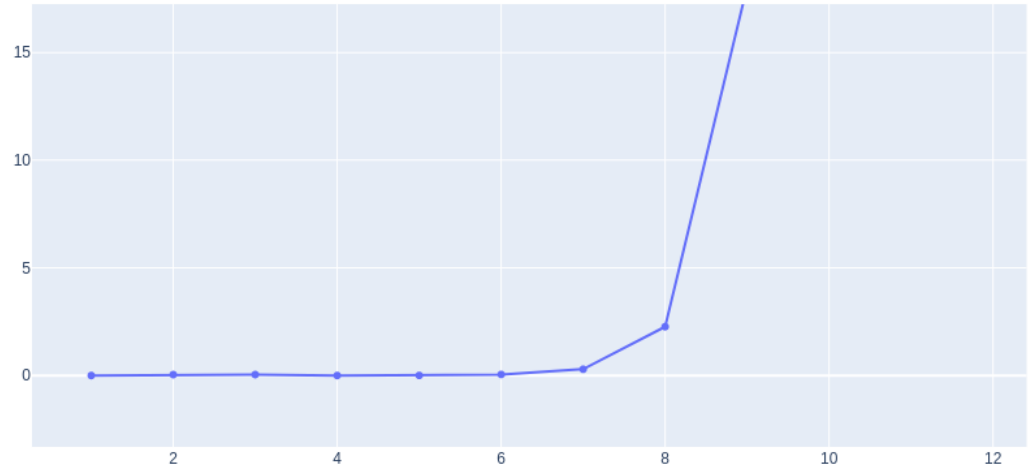
```
using PlotlyJS

plot(
    table(
        header_values=["N", "LU det", "Julia det", "Error"],
        cells_values=[
            2 .^ (1:N),
            lu_dets,
            true_dets,
            abs.(lu_dets .- true_dets)
        ]
    )
)
```

N	LU det	Julia det	Error
2	-0.05324115813432733	-0.053241158134327365	3.469446951953614e-17
4	-0.07383607675297599	-0.07383607675297599	0
8	0.024555517869351163	0.024555517869351156	6.938893903907228e-18
16	-0.16390859300729385	-0.16390859300729121	2.6367796834847468e-15
32	1.977599375690634	1.977599375690413	2.2093438190040615e-13
64	-1526856801.1998003	-1526856801.1317983	0.06800198554992676
128	5.178742808093861e+39	5.1787427733185506e+39	3.477531032940638e+31
256	8.103951098206515e+113	8.103950847922709e+113	2.5028380589051585e+106
512	2.8059084170309253e+306	2.8059084854565758e+306	6.842565050242886e+298

Wykres czasu (w sekundach), potrzebnego do obliczenia macierzy L oraz U w zależności od parametru k.

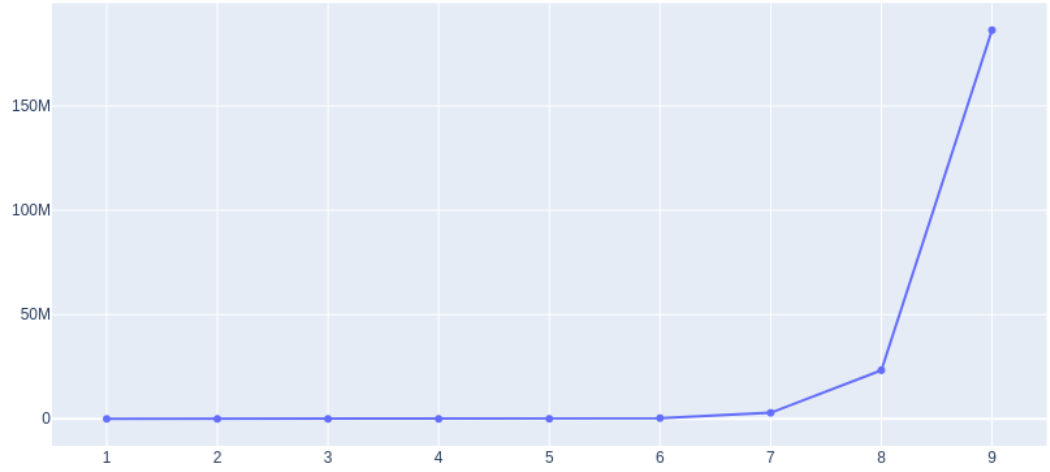
```
function test_time_lu(n)
    times = zeros(n)
    flops = zeros(n)
    for i in 1:n
        A = rand(2^i, 2^i)
        global FLOPS = 0
        times[i] = @elapsed lu_factorization(A)
        flops[i] = FLOPS
        println("n=$i, time=$(times[i]), flops=$(flops[i])")
    end
    return times, flops
end

times, flops = test_time_lu(9)
```

plot(times, label=”time”, xlabel=”log2(n)”, ylabel=”time (s)”)



Wykres ilości operacji zmiennoprzecinkowych, potrzebnych do obliczenia macierzy L oraz U w zależności od parametru k.

plot(flops, label="flops", xlabel="log2(n)", ylabel="flops")



Wnioski:

Własna implementacja dobrze sprawdza się dla małych rozmiarów macierzy (to jest mniejszych niż 64×64). Dla większych błąd jest na tyle duży, że może powodować błędy w obliczeniach. Do zastosowań profesjonalnych należy wybierać wersję biblioteczną ze względu na liczne opytmalizacje oraz użycie mechanizmów, dzięki którym algorytm ten jest stabilny numerycznie.