Próg Fredericksza w ciekłym krysztale nematycznym

Michał Łukomski

Styczeń 2021

Symulacje zostały przeprowadzone na sieci o rozmiarze L^2, gdzie $L = 20$. ξ^* oraz E^* zostały wyrażone w jednostkach zredukowanych $\xi^* = \xi/(k_B T)$, $E^* = E/(k_B T)$.

Symulacje startowano z konfiguracją uporządkowaną - wszystkie kąty zostały ustawione na 0°.

Sieci zostały poddane ewolucji przez $30000MCS$, ($1MCS$ = iteracyjne przejście po wszystkich węzłach układu) a następnie przez kolejne $200000MCS$ podczas których n_{eff} (a w jednym przejściu także $< \cos^2(\phi) > (z)$) było próbkowane co $100MCS$. Otrzymano więc próbki o rozmiarach 2000, z których obliczono średnie. Wyniki przedstawiono na wykresach.

Kąty mogły przyjmować całkowite wartości z przedziału $[-90° : 90°]$.

Zakres natężenia zewnętrznego (pionowego) pola użyty w symulacjach to

- $[0 : 3.5]$ ze skokiem $\Delta E^* = 0.001$.

Inne parametry wykorzystane we wszystkich symulacjach to

- $\xi^* = 20$
- $n_0 = 1.5$
- $n_e = 1.7$
- $\Delta\phi = 10°$

Próg Fredericksza odczytano z wykresu jako $E_F^* = 0.73$

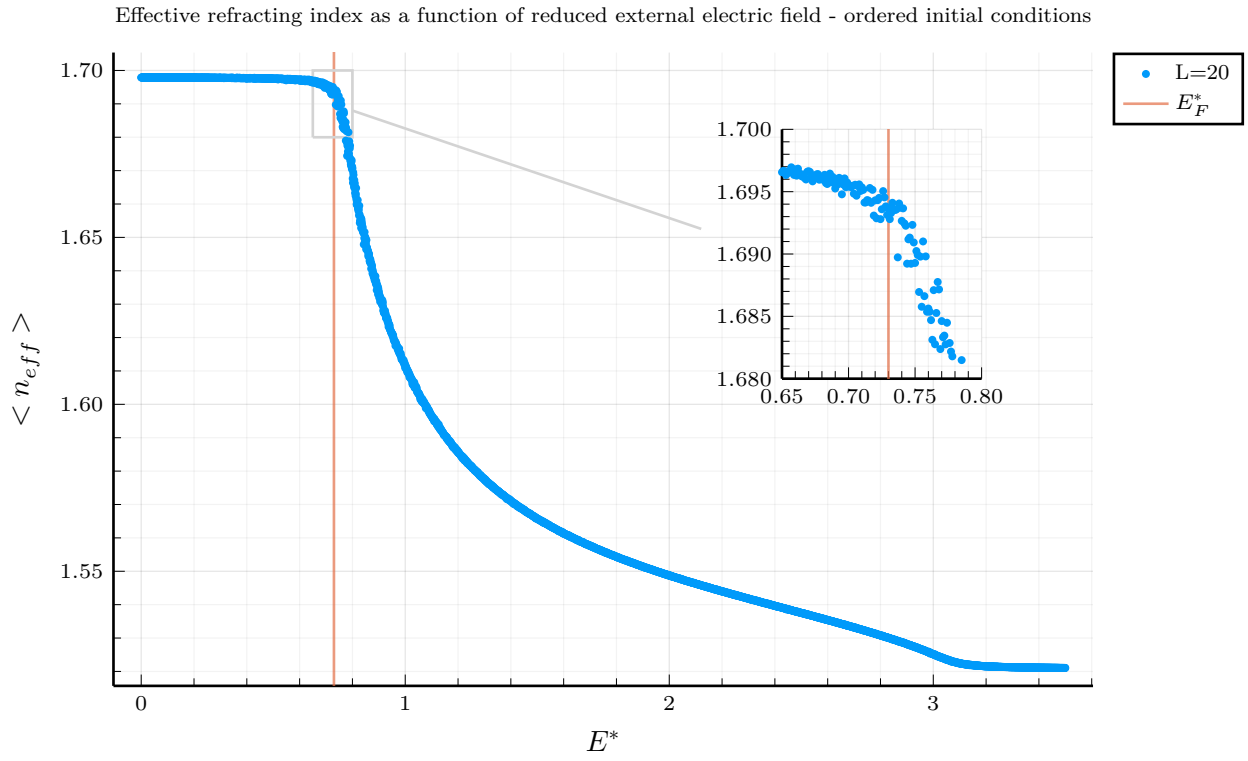

Figure 1: Zależność średniej wartości efektywnego współczynnika załamania światła w komórce NLC od natężenia zewnętrznego pola elektrycznego w jednostkach zredukowanych. Próg Freedericksza odczytano jako $E_F^* = 0.73$ i zaznaczono na wykresie.

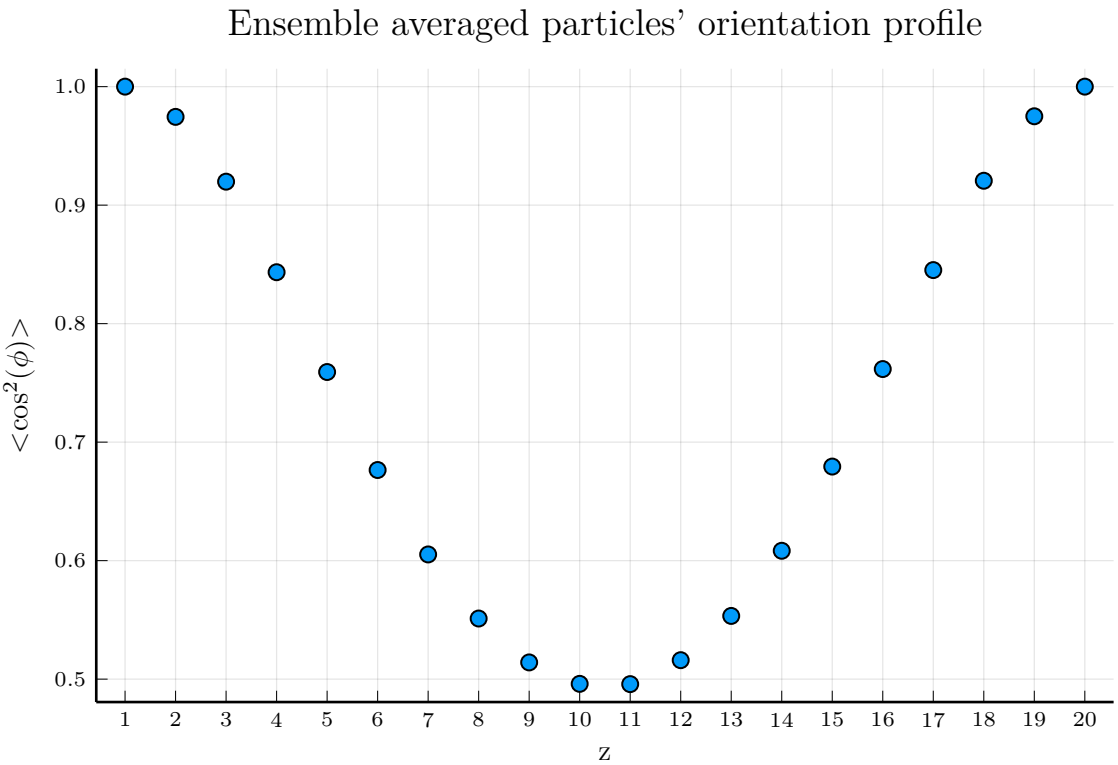

Figure 2: Profil orientacji cząsteczek NLC (uśredniony po zespole statystycznym profil $< \cos^2(\phi) > (z)$, gdzie z jest numerem rzędu) dla wartości natężenia pola $E^* = 1.2E_F^* = 0.876$

Kod programu - Julia

```julia
using Statistics
using OffsetArrays
using JLD2
using Plots
pgfplotsx()

const L = 20
MCS = 230_000

const Δϕ = 10 #°
# const T = 2
const ξ = 20.0
const n₀ = 1.5
const nₑ = 1.7

const P_func(β) = (3 * cosd(β)^2 -1)/2
const P₂ = OffsetArray(map(P_func, -180:180), -180:180)
Iₙ = collect(2:L+1)
Iₚ = collect(0:L-1)
Iₙ[20] = 1
Iₚ[1] = L

const randoms_Δϕ = Int16.(cat(-Δϕ÷2:-1, 1:Δϕ÷2, dims=1))

n_eff(φ::Number, n₀ = n₀, nₑ = nₑ)::Float64 = n₀ * nₑ / √(n₀^2*cosd(φ)^2 + nₑ^2*sind(φ)^2)

function nlc(E::Number, MCS::Integer=230_000, L::Integer=20, skipfirst::Integer=30_000, probe_every::Integer=100,
    P₂=P₂, Iₙ=Iₙ, Iₚ=Iₚ, ξ=ξ, randoms_Δϕ=randoms_Δϕ)::Float64
    #ordered initial conditions - all ϕ=0°
    ϕ = zeros(Int16,L,L)
    n_eff_sum = 0.0

    for k ∈ 1:MCS
        for j ∈ 1:L, i ∈ 2:L-1
            #metropolis algorithm
            @inbounds ϕ_new::Int16 = ϕ[i,j] + rand(randoms_Δϕ)
            ϕ_new > 90  && (ϕ_new -= 180) # a && b  => if a then run b
            ϕ_new < -90 && (ϕ_new += 180)

            @inbounds U_old::Float64 = -ξ * (P₂[ϕ[i,j] - ϕ[Iₙ[i],j]] + P₂[ϕ[i,j] - ϕ[Iₚ[i],j]] + P₂[ϕ[i,j] -
             ϕ[i,Iₙ[j]]] + P₂[ϕ[i,j] - ϕ[i,Iₚ[j]]]) - E^2 * P₂[90-ϕ[i,j]]

            @inbounds U_new::Float64 = -ξ * (P₂[ϕ_new - ϕ[Iₙ[i],j]] + P₂[ϕ_new - ϕ[Iₚ[i],j]] + P₂[ϕ_new -
             ϕ[i,Iₙ[j]]] + P₂[ϕ_new - ϕ[i,Iₚ[j]]]) - E^2 * P₂[90-ϕ_new]

            ΔU::Float64 = U_new - U_old
            if ΔU < 0  rand() ≤ exp(-ΔU)
                @inbounds ϕ[i,j] = ϕ_new
            end
        end
        if k > skipfirst && k%probe_every == 0
            n_eff_sum += sum(n_eff, ϕ)/L^2
        end
    end
    return n_eff_sum / ((MCS-skipfirst)÷probe_every) #return mean <n_eff>
    #return ϕ
```

```
end

function nlc_cos2_profile(E::Number, MCS::Integer=230_000, L::Integer=20, skipfirst::Integer=30_000,
    probe_every::Integer=100, P₂=P₂, Iₙ=Iₙ, Iₚ=Iₚ, ξ=ξ, randoms_Δϕ=randoms_Δϕ)::Array{Float64}
    #ordered initial conditions - all ϕ=0°
    ϕ = zeros(Int16,L,L)
    cos2_profile = zeros(L)

    for k ∈ 1:MCS
        for j ∈ 1:L, i ∈ 2:L-1
            #metropolis algorithm
            @inbounds ϕ_new::Int16 = ϕ[i,j] + rand(randoms_Δϕ)
            ϕ_new > 90  && (ϕ_new -= 180) # a && b  => if a then run b
            ϕ_new < -90 && (ϕ_new += 180)

            @inbounds U_old::Float64 = -ξ * (P₂[ϕ[i,j] - ϕ[Iₙ[i],j]] + P₂[ϕ[i,j] - ϕ[Iₚ[i],j]] + P₂[ϕ[i,j] -
             ϕ[i,Iₙ[j]]] + P₂[ϕ[i,j] - ϕ[i,Iₚ[j]]]) - E^2 * P₂[90-ϕ[i,j]]

            @inbounds U_new::Float64 = -ξ * (P₂[ϕ_new - ϕ[Iₙ[i],j]] + P₂[ϕ_new - ϕ[Iₚ[i],j]] + P₂[ϕ_new -
             ϕ[i,Iₙ[j]]] + P₂[ϕ_new - ϕ[i,Iₚ[j]]]) - E^2 * P₂[90-ϕ_new]

            ΔU::Float64 = U_new - U_old
            if ΔU < 0  rand() ≤ exp(-ΔU)
                @inbounds ϕ[i,j] = ϕ_new
            end
        end
        if k > skipfirst && k%probe_every == 0
            cos2_profile += mean(x -> cosd(x)^2, ϕ, dims=2)
        end
    end
    return cos2_profile ./ ((MCS-skipfirst)÷probe_every)
end

function run_nlc(Es)
    result = Dict{Float64,Float64}()
    Threads.@threads for E ∈ Es
        result[E] = @time nlc(E)
    end
    return result
end

Es = 0:0.001:3.5

result_ret = @time run_nlc(Es)
@save "result_002.jld2" result_ret
@load "result_002.jld2" result_ret

# serialize("result_002.jls", result)

scatter(result_ret, markershape=:auto, markersize=2, xlabel="\$E^*\$", ylabel="\$n_{eff}\$",
        title="Effective refracting index as a function of reduced external electric field - ordered initial
         conditions",
             titlefontsize=8, markerstrokealpha=0.0, label="L=20",  legend=true)
lens!([0.65,0.8],[1.68,1.7], minorticks=5, minorgrid=true, legend=false, inset = (1, bbox(0.6, 0.1, 0.3, 0.5)))
savefig("neff_E.tex")

#@save "result.jld2" result_ret
#@load "result.jld2"

```

```julia
using Profile
using StatProfilerHTML

# Profile.clear()
# result_ret = @@profilehtml run_nlc(Es)


cos2_profile = nlc_cos2_profile(1.2*0.73)
scatter(cos2_profile, grid=:both, xticks=1:20, xlabel="z", ylabel="<\$\\cos^2(\\phi)\$>", legend=:none,
        title="Ensemble averaged particles' orientation profile")
savefig("cos2_profile.tex")
```
