Entrée [5]:

```
#packages

# using Pkg
# Pkg.add("LinearAlgebra")
# Pkg.add("LinearSolve")
# Pkg.add("Plots")
# Pkg.add("Printf")
# Pkg.add("LaTeXStrings")
# Pkg.add("Interpolations")
# Pkg.add("NPZ")


using LinearAlgebra
using LinearSolve
using Plots
using LaTeXStrings
using Interpolations
using NPZ
using Printf


#--------------------------------------------------------------------------------
#   dressing of function f (given as array f_discr, occupation ratio given as n
#--------------------------------------------------------------------------------

function dress(gb, lam_discr, n_discr, f_discr)
    L = size(lam_discr)[1]
    varphi(lam) = 2*gb/( gb^2 + lam^2 )
    varphimat = zeros(L,L)
    dlam = zeros(L)

    for i = 1:L
        for j = 1:i
            varphimat[i,j] = varphi( lam_discr[i]-lam_discr[j] )
            varphimat[j,i] = varphi( lam_discr[i]-lam_discr[j] )
        end
    end
    varphimat = Symmetric(varphimat)

    for i = 2:L-1
        dlam[i] = 0.5*(lam_discr[i+1]-lam_discr[i-1])
    end
    dlam[1] = 0.5*(lam_discr[2]-lam_discr[1])
    dlam[L] = 0.5*(lam_discr[L]-lam_discr[L-1])
    A = I - 0.5/pi * varphimat*Diagonal(n_discr)*Diagonal(dlam)
    A\f_discr
end

#--------------------------------------------------------------------------------
#   to evaluate charge density associated with function f
#--------------------------------------------------------------------------------

function charge_density(gb, lam_discr, n_discr, f_discr)
    L = size(lam_discr)[1]
    varphi(lam) = 2*gb/( gb^2 + lam^2 )
    varphimat = zeros(L,L)
    dlam = zeros(L)

    for i = 1:L
```

```julia
        for j = 1:i
            varphimat[i,j] = varphi( lam_discr[i]-lam_discr[j] )
            varphimat[j,i] = varphi( lam_discr[i]-lam_discr[j] )
        end
    end
    varphimat = Symmetric(varphimat)

    for i = 2:L-1
        dlam[i] = 0.5*(lam_discr[i+1]-lam_discr[i-1])
    end
    dlam[1] = 0.5*(lam_discr[2]-lam_discr[1])
    dlam[L] = 0.5*(lam_discr[L]-lam_discr[L-1])
    A = I - 0.5/pi * varphimat*Diagonal(n_discr)*Diagonal(dlam)
    0.5/pi * dot(dlam , Diagonal(n_discr) * (A \ f_discr) )
end

#----------------------------------------------------------------------
#   solve Yang-Yang equation
#----------------------------------------------------------------------

function fun1(z)
    if z>0
        return log(1. +exp(-z))
    else
        return log(1. +exp(z)) - z
    end
end

function fun2(z)
    if z<0
        return 1. /(1. +exp(z))
    else
        return exp(-z)/(1. +exp(-z))
    end
end

function yangyang(gb, beta, lam_discr)
    L = size(lam_discr)[1]
    varphi(lam) = 2*gb/( gb^2 + lam^2 )
    varphimat = zeros(L,L)
    dlam = zeros(L)

    for i = 1:L
        for j = 1:i
            varphimat[i,j] = varphi( lam_discr[i]-lam_discr[j] )
            varphimat[j,i] = varphi( lam_discr[i]-lam_discr[j] )
        end
    end
    varphimat = Symmetric(varphimat)

    for i = 2:L-1
        dlam[i] = 0.5*(lam_discr[i+1]-lam_discr[i-1])
    end
    dlam[1] = 0.5*(lam_discr[2]-lam_discr[1])
    dlam[L] = 0.5*(lam_discr[L]-lam_discr[L-1])

    bare_E = 0.5 * beta[3] * lam_discr.^2 + beta[2] * lam_discr + beta[1] * one
    eps = copy(bare_E)
    n = fun2.(eps)

    #iterative solution
    diff = 1.
    while diff>1e-12
        old_n = copy(n)
        eps = bare_E - 0.5/pi * varphimat*Diagonal( fun1.(eps) )*dlam
        n = fun2.(eps)
        diff = norm(n-old_n)
    end
    return n
end

#-------------------------------------------------------------------------------
#   compute effective velocity
#-------------------------------------------------------------------------------

function veff(gb, lam_discr, n_discr)
    L = size(lam_discr)[1]
    varphi(lam) = 2*gb/( gb^2 + lam^2 )
    varphimat = zeros(L,L)
    dlam = zeros(L)

    for i = 1:L
        for j = 1:i
            varphimat[i,j] = varphi( lam_discr[i]-lam_discr[j] )
            varphimat[j,i] = varphi( lam_discr[i]-lam_discr[j] )
        end
    end
    varphimat = Symmetric(varphimat)

    for i = 2:L-1
        dlam[i] = 0.5*(lam_discr[i+1]-lam_discr[i-1])
    end
    dlam[1] = 0.5*(lam_discr[2]-lam_discr[1])
    dlam[L] = 0.5*(lam_discr[L]-lam_discr[L-1])
    dresser = I - 0.5/pi * varphimat*Diagonal(n_discr)*Diagonal(dlam)
    onedr = dresser\ones(L)
    iddr = dresser\copy(lam_discr)
    iddr ./ onedr
end

#-------------------------------------------------------------------------------
#   time evolution for box expansion
#-------------------------------------------------------------------------------

function evol_box_expansion(dt, x1, x2, gb, theta_tab, n_fun)
    npts_integral = 600
    L = size(x1)[1]

    veff1 = zeros(L)
    veff2 = zeros(L)
    f2 = linear_interpolation(x2, theta_tab)

    #velocity of first and last points is the one of free particles
    veff1[1] = theta_tab[1]
    veff2[L] = theta_tab[L]

    for j in 2:L
        thet1 = theta_tab[j]
        if x1[j]<x2[1]
            thet2 = theta_tab[1]
        else
            thet2 = f2(x1[j])
```

```julia
        end
        lam_discr = LinRange(thet2,thet1,npts_integral)
        n_discr = n_fun.(lam_discr)
        veff1[j] = last(veff(gb, lam_discr, n_discr))
        #velocities of right contour (using parity symmetry)
        veff2[L+1-j] = -veff1[j]
    end

    return x1 + dt*veff1, x2 + dt*veff2
end

#------------------------------------------------------------------------------
#   box expansion: calculate density (careful: return density in units of rapi
#    for density in mu^-1 multiply by mass/hbar)
#------------------------------------------------------------------------------

function eval_density(x1, x2, gb, theta_tab, n_fun, ratio_m_hbar)
    L = size(x1)[1]

    dens = zeros(2,2*L)
    f2 = linear_interpolation(x2, theta_tab)

    #density of first and last points is zero
    dens[1,1] = x1[1]
    dens[2,1] = 0
    dens[1,L+1] = x2[L]
    dens[2,L+1] = 0

    for j in 2:L
        thet1 = theta_tab[j]
        if x1[j]<x2[1]
            thet2 = theta_tab[1]
        else
            thet2 = f2(x1[j])
        end
        lam_discr = LinRange(thet2,thet1,600)
        n_discr = n_fun.(lam_discr)
        dens[1,j] = x1[j]
        dens[2,j] = ratio_m_hbar * charge_density(gb, lam_discr, n_discr, ones
        #velocities of right contour (using parity symmetry)
        dens[1,L+j] = x2[L+1-j]
        dens[2,L+j] = dens[2,j]
    end
    #sort densities according to positions, and return
    dens[:,sortperm(dens[1, :])]
end

#------------------------------------------------------------------------------
#   function for simulating box expansion. Writes results in file 'data/density
#     where 't' is the time (takes a list of times to be saved as argument)
#------------------------------------------------------------------------------

function box_expansion(n_fun, dt, list_t_save, theta_max, npts, gb, ratio_m_hba
    tmax = maximum(list_t_save)
    theta_tab = LinRange(-theta_max, theta_max, npts)
    x1 = -0.5 * ones(npts) + 0.0000001*theta_tab
    x2 = 0.5 * ones(npts) + 0.0000001*theta_tab

    t = 0
    while t<=tmax
        x1, x2 = evol_box_expansion(dt, x1, x2, gb, theta_tab, n_fun)
        t += dt

        for ts in list_t_save
            if t-dt<ts && ts<=t
                mydensity = eval_density(x1, x2, gb, theta_tab, n_fun, ratio_m_
                println(@sprintf "t = %.3f saved" t)
                filename = @sprintf "2023_11_08/01_density_expansion_%.3f.npz"
                npzwrite(filename, mydensity)
            end
        end
    end
end

```

Out[5]:

```
box_expansion (generic function with 1 method)
```

Entrée [3]:

```
hbar = 1.05457182e-25   # um^2.kg/ms
mass =  1.44e-25        # kg (mass of Rubidium 87)
kB = 1.380649e-26       # um^2.ms^-2.kg.nK^{-1}
a3D = 5.3e-3            # um

om_perp = 2*pi*2.56   #kHz (transverse frequency)


g = 2*hbar*a3D * om_perp        #effective 1d repulsion strength
c = mass/hbar * 2*a3D*om_perp   #um^{-1}
gbar = hbar/mass * c            #um.ms^{-1}


#----------------------------------------
#    parameters for T/mu = 1.79
#----------------------------------------

#mu=71.5        #chemical potential     (nK)
#T=120.             #temperature            (nK)


#----------------------------------------
#    parameters for T/mu = 2.14
#----------------------------------------

mu=47.0     #chemical potential     (nK)
T=81.0          #temperature            (nK)
Taille = 37.    # Taille de la boîte (microns)

lam_disc = LinRange(-20,20,4000)

beta = [ (-mu/T) 0 mass/kB/T ]
@time n = yangyang(gbar, beta, lam_disc)
@time rhos = 0.5/pi * dress(gbar, lam_disc, n, ones(size(lam_disc)[1]))
rhop = mass/hbar * rhos .* n
#print atom density
#println( sum(rhop) * (lam_disc[2]-lam_disc[1]) )

#save rho_p in file
tab_rhop = zeros(2,size(lam_disc)[1])
for j in 1:size(lam_disc)[1]
    tab_rhop[1,j] = lam_disc[j]
    tab_rhop[2,j] = rhop[j]
end
npzwrite("2023_11_08/01_rhop.npz",tab_rhop)


#simulation of expansion
n_fun = linear_interpolation(lam_disc, n)
box_expansion(n_fun, 0.001, [10.0/Taille, 20.0/Taille, 30.0/Taille, 40.0/Taille
```

```
  1.460862 seconds (590.30 k allocations: 290.878 MiB, 7.00% gc tim
e, 39.15% compilation time)
  1.321331 seconds (263.50 k allocations: 750.337 MiB, 4.80% gc tim
e, 24.74% compilation time)
t = 0.271 saved
t = 0.541 saved
t = 0.811 saved
```

```
t = 1.082 saved
t = 1.352 saved
t = 1.622 saved
```

Entrée [4]:

```
25*14*32/750
```

Out[4]:

```
14.933333333333334
```

Entrée [ ]:

```

```