

Entrée [5]:

```

1  #packages
2
3  # using Pkg
4  # Pkg.add("LinearAlgebra")
5  # Pkg.add("LinearSolve")
6  # Pkg.add("Plots")
7  # Pkg.add("Printf")
8  # Pkg.add("LaTeXStrings")
9  # Pkg.add("Interpolations")
10 # Pkg.add("NPZ")
11
12
13 using LinearAlgebra
14 using LinearSolve
15 using Plots
16 using LaTeXStrings
17 using Interpolations
18 using NPZ
19 using Printf
20
21
22 #-----
23 #   dressing of function f (given as array f_discr, occupation ratio given as )
24 #-----
25
26 function dress.gb, lam_discr, n_discr, f_discr)
27     L = size(lam_discr)[1]
28     varphi(lam) = 2*gb/( gb^2 + lam^2 )
29     varphimat = zeros(L,L)
30     dlam = zeros(L)
31
32     for i = 1:L
33         for j = 1:i
34             varphimat[i,j] = varphi( lam_discr[i]-lam_discr[j] )
35             varphimat[j,i] = varphi( lam_discr[i]-lam_discr[j] )
36         end
37     end
38     varphimat = Symmetric(varphimat)
39
40     for i = 2:L-1
41         dlam[i] = 0.5*(lam_discr[i+1]-lam_discr[i-1])
42     end
43     dlam[1] = 0.5*(lam_discr[2]-lam_discr[1])
44     dlam[L] = 0.5*(lam_discr[L]-lam_discr[L-1])
45     A = I - 0.5/pi * varphimat*Diagonal(n_discr)*Diagonal(dlam)
46     A\f_discr
47 end
48
49 #-----
50 #   to evaluate charge density associated with function f
51 #-----
52
53 function charge_density.gb, lam_discr, n_discr, f_discr)
54     L = size(lam_discr)[1]
55     varphi(lam) = 2*gb/( gb^2 + lam^2 )
56     varphimat = zeros(L,L)
57     dlam = zeros(L)
58
59     for i = 1:L

```

```

60     for j = 1:i
61         varphimat[i,j] = varphi( lam_discr[i]-lam_discr[j] )
62         varphimat[j,i] = varphi( lam_discr[i]-lam_discr[j] )
63     end
64 end
65 varphimat = Symmetric(varphimat)
66
67 for i = 2:L-1
68     dlam[i] = 0.5*(lam_discr[i+1]-lam_discr[i-1])
69 end
70 dlam[1] = 0.5*(lam_discr[2]-lam_discr[1])
71 dlam[L] = 0.5*(lam_discr[L]-lam_discr[L-1])
72 A = I - 0.5/pi * varphimat*Diagonal(n_discr)*Diagonal(dlam)
73 0.5/pi * dot(dlam , Diagonal(n_discr) * (A \ f_discr) )
74 end
75
76 #-----
77 # solve Yang-Yang equation
78 #-----
79
80 function fun1(z)
81     if z>0
82         return log(1. +exp(-z))
83     else
84         return log(1. +exp(z)) - z
85     end
86 end
87
88 function fun2(z)
89     if z<0
90         return 1. /(1. +exp(z))
91     else
92         return exp(-z)/(1. +exp(-z))
93     end
94 end
95
96 function yangyang(gb, beta, lam_discr)
97     L = size(lam_discr)[1]
98     varphi(lam) = 2*gb/( gb^2 + lam^2 )
99     varphimat = zeros(L,L)
100     dlam = zeros(L)
101
102     for i = 1:L
103         for j = 1:i
104             varphimat[i,j] = varphi( lam_discr[i]-lam_discr[j] )
105             varphimat[j,i] = varphi( lam_discr[i]-lam_discr[j] )
106         end
107     end
108     varphimat = Symmetric(varphimat)
109
110     for i = 2:L-1
111         dlam[i] = 0.5*(lam_discr[i+1]-lam_discr[i-1])
112     end
113     dlam[1] = 0.5*(lam_discr[2]-lam_discr[1])
114     dlam[L] = 0.5*(lam_discr[L]-lam_discr[L-1])
115
116     bare_E = 0.5 * beta[3] * lam_discr.^2 + beta[2] * lam_discr + beta[1] * one
117     eps = copy(bare_E)
118     n = fun2.(eps)
119
120     #iterative solution

```

```

121     diff = 1.
122     while diff>1e-12
123         old_n = copy(n)
124         eps = bare_E - 0.5/pi * varphimat*Diagonal( fun1.(eps) )*dlam
125         n = fun2.(eps)
126         diff = norm(n-old_n)
127     end
128     return n
129 end
130
131 #-----
132 #   compute effective velocity
133 #-----
134
135 function veff(gb, lam_discr, n_discr)
136     L = size(lam_discr)[1]
137     varphi(lam) = 2*gb/( gb^2 + lam^2 )
138     varphimat = zeros(L,L)
139     dlam = zeros(L)
140
141     for i = 1:L
142         for j = 1:i
143             varphimat[i,j] = varphi( lam_discr[i]-lam_discr[j] )
144             varphimat[j,i] = varphi( lam_discr[i]-lam_discr[j] )
145         end
146     end
147     varphimat = Symmetric(varphimat)
148
149     for i = 2:L-1
150         dlam[i] = 0.5*(lam_discr[i+1]-lam_discr[i-1])
151     end
152     dlam[1] = 0.5*(lam_discr[2]-lam_discr[1])
153     dlam[L] = 0.5*(lam_discr[L]-lam_discr[L-1])
154     dresser = I - 0.5/pi * varphimat*Diagonal(n_discr)*Diagonal(dlam)
155     onedr = dresser\ones(L)
156     iddr = dresser\copy(lam_discr)
157     iddr ./ onedr
158 end
159
160 #-----
161 #   time evolution for box expansion
162 #-----
163
164 function evol_box_expansion(dt, x1, x2, gb, theta_tab, n_fun)
165     npts_integral = 600
166     L = size(x1)[1]
167
168     veff1 = zeros(L)
169     veff2 = zeros(L)
170     f2 = linear_interpolation(x2, theta_tab)
171
172     #velocity of first and last points is the one of free particles
173     veff1[1] = theta_tab[1]
174     veff2[L] = theta_tab[L]
175
176     for j in 2:L
177         thet1 = theta_tab[j]
178         if x1[j]<x2[1]
179             thet2 = theta_tab[1]
180         else
181             thet2 = f2(x1[j])

```

```

182     end
183     lam_discr = LinRange(thet2,thet1,npts_integral)
184     n_discr = n_fun.(lam_discr)
185     veff1[j] = last(veff(gb, lam_discr, n_discr))
186     #velocities of right contour (using parity symmetry)
187     veff2[L+1-j] = -veff1[j]
188 end
189
190 return x1 + dt*veff1, x2 + dt*veff2
191 end
192
193 #-----
194 # box expansion: calculate density (careful: return density in units of rapidity)
195 # for density in  $\mu^{-1}$  multiply by mass/hbar)
196 #-----
197
198 function eval_density(x1, x2, gb, theta_tab, n_fun, ratio_m_hbar)
199     L = size(x1)[1]
200
201     dens = zeros(2,2*L)
202     f2 = linear_interpolation(x2, theta_tab)
203
204     #density of first and last points is zero
205     dens[1,1] = x1[1]
206     dens[2,1] = 0
207     dens[1,L+1] = x2[L]
208     dens[2,L+1] = 0
209
210     for j in 2:L
211         thet1 = theta_tab[j]
212         if x1[j]<x2[1]
213             thet2 = theta_tab[1]
214         else
215             thet2 = f2(x1[j])
216         end
217         lam_discr = LinRange(thet2,thet1,600)
218         n_discr = n_fun.(lam_discr)
219         dens[1,j] = x1[j]
220         dens[2,j] = ratio_m_hbar * charge_density(gb, lam_discr, n_discr, ones(n_discr))
221         #velocities of right contour (using parity symmetry)
222         dens[1,L+j] = x2[L+1-j]
223         dens[2,L+j] = dens[2,j]
224     end
225     #sort densities according to positions, and return
226     dens[:,sortperm(dens[1, :])]
227 end
228
229 #-----
230 # function for simulating box expansion. Writes results in file 'data/density'
231 # where 't' is the time (takes a list of times to be saved as argument)
232 #-----
233
234 function box_expansion(n_fun, dt, list_t_save, theta_max, npts, gb, ratio_m_hbar)
235     tmax = maximum(list_t_save)
236     theta_tab = LinRange(-theta_max, theta_max, npts)
237     x1 = -0.5 * ones(npts) + 0.0000001*theta_tab
238     x2 = 0.5 * ones(npts) + 0.0000001*theta_tab
239
240     t = 0
241     while t<=tmax
242         x1, x2 = evol_box_expansion(dt, x1, x2, gb, theta_tab, n_fun)

```

```
243         t += dt
244
245         for ts in list_t_save
246             if t-dt<ts && ts<=t
247                 mydensity = eval_density(x1, x2, gb, theta_tab, n_fun, ratio_m
248                 println(@sprintf "t = %.3f saved" t)
249                 filename = @sprintf "2023_11_08/01_density_expansion_%.3f.npz"
250                 npzwrite(filename, mydensity)
251             end
252         end
253     end
254 end
255
```

Out[5]:

box_expansion (generic function with 1 method)

Entrée [3]:

```

1 hbar = 1.05457182e-25 # um^2.kg/ms
2 mass = 1.44e-25 # kg (mass of Rubidium 87)
3 kB = 1.380649e-26 # um^2.ms^-2.kg.nK^{-1}
4 a3D = 5.3e-3 # um
5
6 om_perp = 2*pi*2.56 #kHz (transverse frequency)
7
8
9 g = 2*hbar*a3D * om_perp #effective 1d repulsion strength
10 c = mass/hbar * 2*a3D*om_perp #um^{-1}
11 gbar = hbar/mass * c #um.ms^{-1}
12
13
14 #-----
15 # parameters for T/mu = 1.79
16 #-----
17
18 #mu=71.5 #chemical potential (nK)
19 #T=120. #temperature (nK)
20
21
22 #-----
23 # parameters for T/mu = 2.14
24 #-----
25
26 mu=47.0 #chemical potential (nK)
27 T=81.0 #temperature (nK)
28 Taille = 37. # Taille de la boîte (microns)
29
30 lam_disc = LinRange(-20,20,4000)
31
32 beta = [ (-mu/T) 0 mass/kB/T ]
33 @time n = yangyang(gbar, beta, lam_disc)
34 @time rhos = 0.5/pi * dress(gbar, lam_disc, n, ones(size(lam_disc)[1]))
35 rhop = mass/hbar * rhos .* n
36 #print atom density
37 #println( sum(rhop) * (lam_disc[2]-lam_disc[1]) )
38
39 #save rho_p in file
40 tab_rhop = zeros(2,size(lam_disc)[1])
41 for j in 1:size(lam_disc)[1]
42     tab_rhop[1,j] = lam_disc[j]
43     tab_rhop[2,j] = rhop[j]
44 end
45 npzwrite("2023_11_08/01_rhop.npz",tab_rhop)
46
47
48 #simulation of expansion
49 n_fun = linear_interpolation(lam_disc, n)
50 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 time, 39.15% compilation time)

1.321331 seconds (263.50 k allocations: 750.337 MiB, 4.80% gc time, 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]:

1	$25 \cdot 14 \cdot 32 / 750$
---	------------------------------

Out[4]:

14.933333333333334

Entrée []:

1	
---	--