Rotating Bosons Project

Casey E. Berger

April 7th, 2017 – February 14th, 2018

Contents

1 Python Scripts Used in Analysis

1.1 *Script to Compute Noninteracting Density and Field Modulus Squared*

A python script was used to perform the sum over functions of the eigenvalues of the diagonal matrix D representing the noninteracting action.

```
#!/usr/bin/env python

#computes the analytical solutions available to us
import sys
import numpy as np
import matplotlib.pyplot as plt
import cmath


def noninteracting_phisq(d,m,Nx,Nt,mu):
   if d==1:
      phisq = 0.
      count = 0
      for n0 in range(Nt):
         #w = 2.*np.pi*(float(n0) + 0.5)
         w = 2.*np.pi*float(n0)/float(Nt)
         for nx in range(Nx):
            #kx = 2.*np.pi*(float(nx) + 0.5)
            kx = 2.*np.pi*float(nx)/float(Nx)
            D = complex(2.*float(d+1) + m*m -
                2.*np.cos(kx)-2.*np.cosh(mu)*np.cos(w),2.0*np.sinh(mu)*np.sin(w))
            phisq += D.conjugate()/(abs(D)*abs(D))
            count += 1
      #print "%s diagonals in Dkk"%count
      return phisq/(float(Nt)*float(np.power(Nx,d)))
   elif d==2:
      phisq = 0.
      count = 0
      for n0 in range(Nt):
         #w = 2.*np.pi*(float(n0) + 0.5)
         w = 2.*np.pi*float(n0)/float(Nt)
```

```
        for nx in range(Nx):
           #kx = 2.*np.pi*(float(nx) + 0.5)
           kx = 2.*np.pi*float(nx)/float(Nx)
           for ny in range(Nx):
              #ky = 2.*np.pi*(float(ny) + 0.5)
              ky = 2.*np.pi*float(ny)/float(Nx)
              D = complex(2.*float(d+1) + m*m - 2.*np.cos(kx)-
                  2.*np.cos(ky)
                  -2.*np.cosh(mu)*np.cos(w),-2.0*np.sinh(mu)*np.sin(w))
              phisq += D.conjugate()/(abs(D)*abs(D))
              count += 1
     #print "%s diagonals in Dkk"%count
     return phisq/(float(Nt)*float(np.power(Nx,d)))
  elif d==3:
     phisq = 0.
     count = 0
     for n0 in range(Nt):
        #w = 2.*np.pi*(float(n0) + 0.5)
        w = 2.*np.pi*float(n0)/float(Nt)
        for nx in range(Nx):
           #kx = 2.*np.pi*(float(nx) + 0.5)
           kx = 2.*np.pi*float(nx)/float(Nx)
           for ny in range(Nx):
              #ky = 2.*np.pi*(float(ny) + 0.5)
              ky = 2.*np.pi*float(ny)/float(Nx)
              for nz in range(Nx):
                 #kz = 2.*np.pi*(float(nz) + 0.5)
                 kz = 2.*np.pi*float(nz)/float(Nx)
                 D = complex(2.*float(d+1) + m*m - 2.*np.cos(kx)-
                     2.*np.cos(ky) -
                     2.*np.cos(kz)-2.*np.cosh(mu)*np.cos(w),-2.0*np.sinh(mu)*np.sin(w))
                 phisq += D.conjugate()/(abs(D)*abs(D))
                 count += 1
     #print "%s diagonals in Dkk"%count
     return phisq/(float(Nt)*float(np.power(Nx,d)))
  else:
     print "invalid dimension, D = %s"%d
     sys.exit()

def noninteracting_density(d,m,Nx,Nt,mu):
  if d==1:
     density = 0.
     count = 0
     for n0 in range(Nt):
        #w = 2.*np.pi*(float(n0) + 0.5)
        w = 2.*np.pi*float(n0)/float(Nt)
        for nx in range(Nx):
           #kx = 2.*np.pi*(float(nx) + 0.5)
           kx = 2.*np.pi*float(nx)/float(Nx)
           D = complex(2.*float(d+1) + m*m -
               2.*np.cos(kx)-2.*np.cosh(mu)*np.cos(w),2.0*np.sinh(mu)*np.sin(w))
```

```
            dDdmu =
                complex(-2.0*np.cos(w)*np.sinh(mu),-2.0*np.sin(w)*np.cosh(mu))
            density += (D.conjugate()*dDdmu)/(abs(D)*abs(D))
            count += 1
      #print "%s diagonals in Dkk"%count
      return density/(float(Nt)*float(np.power(Nx,d)))
   elif d==2:
      density = 0.
      count = 0
      for n0 in range(Nt):
         #w = 2.*np.pi*(float(n0) + 0.5)
         w = 2.*np.pi*float(n0)/float(Nt)
         for nx in range(Nx):
            #kx = 2.*np.pi*(float(nx) + 0.5)
            kx = 2.*np.pi*float(nx)/float(Nx)
            for ny in range(Nx):
               #ky = 2.*np.pi*(float(ny) + 0.5)
               ky = 2.*np.pi*float(ny)/float(Nx)
               D = complex(2.*float(d+1) + m*m - 2.*np.cos(kx)-
                   2.*np.cos(ky)
                   -2.*np.cosh(mu)*np.cos(w),-2.0*np.sinh(mu)*np.sin(w))
               dDdmu =
                   complex(-2.0*np.cos(w)*np.sinh(mu),-2.0*np.sin(w)*np.cosh(mu))
               density += (D.conjugate()*dDdmu)/(abs(D)*abs(D))
               count += 1
      #print "%s diagonals in Dkk"%count
      return density/(float(Nt)*float(np.power(Nx,d)))
   elif d==3:
      density = 0.
      count = 0
      for n0 in range(Nt):
         #w = 2.*np.pi*(float(n0) + 0.5)
         w = 2.*np.pi*float(n0)/float(Nt)
         for nx in range(Nx):
            #kx = 2.*np.pi*(float(nx) + 0.5)
            kx = 2.*np.pi*float(nx)/float(Nx)
            for ny in range(Nx):
               #ky = 2.*np.pi*(float(ny) + 0.5)
               ky = 2.*np.pi*float(ny)/float(Nx)
               for nz in range(Nx):
                  #kz = 2.*np.pi*(float(nz) + 0.5)
                  kz = 2.*np.pi*float(nz)/float(Nx)
                  D = complex(2.*float(d+1) + m*m - 2.*np.cos(kx)-
                      2.*np.cos(ky) -
                      2.*np.cos(kz)-2.*np.cosh(mu)*np.cos(w),-2.0*np.sinh(mu)*np.sin(w))
                  dDdmu =
                      complex(-2.0*np.cos(w)*np.sinh(mu),-2.0*np.sin(w)*np.cosh(mu))
                  density += (D.conjugate()*dDdmu)/(abs(D)*abs(D))
                  count += 1
      #print "%s diagonals in Dkk"%count
      return density/(float(Nt)*float(np.power(Nx,d)))
```

```
   else:
      print "invalid dimension, D = %s"%d
      sys.exit()

d = 1
Nx = 4
Nt = 4
m = 1.
mu_min = 0.0
mu_max = 1.8
dmu = 0.05

f =
     open("params_lambda_0_D"+str(d)+"_Nx"+str(Nx)+"_Nt"+str(Nt)+".txt","w")
f.write("{:<5}".format('#mu')+'\t\t')
f.write("{:<10}".format('Re[phisq]')+'\t\t')
f.write("{:<10}".format('Im[phisq]')+'\t\t')
f.write("{:<10}".format('Re[density]')+'\t\t')
f.write("{:<10}".format('Im[density]')+'\n')
mu = mu_min
while mu <= mu_max:
   phisq = 0.
   phisq = noninteracting_phisq(d,m,Nx,Nt,mu)
   density = 0.
   density = noninteracting_density(d,m,Nx,Nt,mu)
   f.write("{:<3.2f}".format(float(mu))+'\t\t')
   f.write("{:<10.5e}".format(float(phisq.real))+'\t\t')
   f.write("{:<10.5e}".format(float(phisq.imag))+'\t\t')
   f.write("{:<10.5e}".format(float(density.real))+'\t\t')
   f.write("{:<10.5e}".format(float(density.imag))+'\n')
   mu += dmu
f.close()
```