R-code-for-calculating-ODFs-and-example-runs_15June2024.Rmd

Alan E. Berger

in https://github.com/AlanBerger/R-code-for-calculating-orientation-distribution-functions-for-rodlike-particles

Herein, **ODF** is the abbreviation for ***orientation distribution function***

The file
compute_approx_ODF_27Jan2024.R
in the GitHub repository noted above contains R code for calculating by iteration ODFs that are local minima of the free energy function modeling orientation distribution functions of rodlike particles, using trapezoidal numerical integration as in

Judith Herzfeld, Alan E. Berger and John W. Wingate, A highly convergent algorithm for computing the orientation distribution functions of rodlike particles, Macromolecules 1984, v17, 1718-1723, https://pubs.acs.org/doi/pdf/10.1021/ma00139a014 [**HBW 1984**]

while handling more general free energy functions.

The README file in this repository contains detailed information on each argument of the R functions compute_approx_ODF and calculate_ODF that are in the file: compute_approx_ODF_27Jan2024.R

The computational runs below produce plots and demonstrate how to use the compute_approx_ODF and calculate_ODF functions to calculate ODFs and their properties.

The R code in compute_approx_ODF_27Jan2024.R should not need to be modified; one should be able to do desired runs just by varying the arguments in the function calls, including the function W(gamma) in the particle interaction term in the free energy function, and the applied field function V(theta) (the default for V is V = 0, so V does not need to be specified when a computational run with no applied field is to be done).

Note clicking on web links in pdf files as displayed in GitHub does not work, but will work if you download the pdf file to your computer.

Note also:

R. F. Kayser Jr. and H. J. Raveche', Bifurcation in Onsager's model of the isotropic-nematic transition, Physical Review A 1978, v17, 2067-2072, https://journals.aps.org/pra/abstract/10.1103/PhysRevA.17.2067 [**KR 1978**]

A. E. Berger, Conditions under which a natural iterative method for calculating the orientation distribution of rodlike particles decreases the free energy at each step. May 29, 2024 [**Berger, submitted**]

The figures and their legends generated in the code below are not necessarily identical to those in [Berger, submitted]. The intent is to give representative examples of calculating ODFs that are local minima of the free energy F, using the functions in the file compute_approx_ODF_27Jan2024.R

For plots of planar and axial quantities vs. B for the **hard core reference system** ($W(\gamma) = \sin(\gamma)$ and $V(\theta) = 0$), we have already run and saved results for axial and planar ODFs vs. B into tab delimited text files. Will read each data file back in as a data frame (df) to do plots of a desired quantity vs. B.

Plot the axial and planar ODFs for the hard core reference system for B = 14

```
#### Plot the axial and planar ODFs for the hard core reference system for B = 14

rm(list = ls())  # clear out any "left over" variables (R objects) (usually a good idea)
##### note rm in R is not the Unix rm that permanently removes files

# calculate the axial and planar ODFs for B = 14
dirstr <- "C:/berger/orientationdistr/ODF_R_fcns"
# location of code for calculating ODFs on my computer

filename <- "compute_approx_ODF_27Jan2024.R"
# the name of the file on my computer containing the code for compute_approx_ODF

full.file.name <- paste(dirstr, "/", filename, sep = "")  # the full path

source(full.file.name)  # "compile" the program

# W(gamma) is the particle interaction kernel function
W <- function(g) sin(g)  # the hard core reference system

# V(theta) is the applied field, the default is  V = 0   in the
# function       compute_approx_ODF
# so there is no need to specify V when calling compute_approx_ODF
# if want zero applied field

N.theta.intervals <- 512
N.phi.intervals <- 2048     # since phi integrals are only done once for a given
#                                        W and V, can take N.phi.intervals "large"
#                                         without requiring too much computing time

calculate_ODF <- compute_approx_ODF(N.theta.intervals, N.phi.intervals, W)
```

```
## [1] "run of   compute_approx_ODF   January 27, 2024 version"
```

```
# returns the function
#    calculate_ODF(initial.f, B, tau.conv, max.num.iter, min.iterations)
# and calculates various quantities that only need to be computed once
# for given values of      N.theta.intervals, N.phi.intervals, W, and V

# See the extensive comments in the README file and in the file:
#            compute_approx_ODF_27Jan2024.R
# for how to use the R functions:  compute_approx_ODF  and   calculate_ODF
# (including details on what their arguments are and how the functions work).


B <- 14
tau.conv <- 1.0e-6  # the bound for the convergence criterion (see [HBW 1984])
max.num.iter <- 3000  # maximum number of iterations allowed
min.iterations <- 10  # require having done  min.iterations  iterations before
#                                        start testing for convergence

# get initial ODFs for calculating axial and planar shaped ODFs

# Get the vector of theta values used for numerical integration.
```

```r
# These are the locations where the ODF is being calculated.

delta.theta <- pi / N.theta.intervals  # length of theta subintervals
num.theta.points <- N.theta.intervals - 1   # number of theta points inside (0, theta)
theta.indices <- 1:num.theta.points
theta.vec <- theta.indices * delta.theta

# Specify starting (initial) values for the ODF f
# (here, this is a vector of values at the grid points theta.vec).
# The function   calculate_ODF    will "adjust" the initial values
# so they are positive (any values < 0.0001 are set to 0.0001), in order to
# avoid issues with the log function.
# Then   calculate_ODF   will scale the initial values, meaning
# all the values are multiplied by the same positive constant) such
# that the integral of the starting ODF
#       (1/2) integral over [0, pi] of f(theta) sin(theta) d theta
# will equal 1   (as calculated by trapezoidal numerical integration)
#
planar.initial.f <- sin(theta.vec) * sin(theta.vec)  # planar shaped - peak at theta = pi / 2
axial.initial.f <- cos(theta.vec) * cos(theta.vec)  # axial shaped - peaks at theta = 0 and pi

axial.results <- calculate_ODF(axial.initial.f, B, tau.conv, max.num.iter, min.iterations)
```

```
## [1] "run of    calculate_ODF    January 27, 2023  version"
```

```r
# calculate_ODF carries out iterations starting with the initial ODF (its first argument)
# and continuing to iterate until convergence is achieved or the maximum number of
# iterations is done

axial.f.for.plotting <- axial.results[[2]]    # y values for plot
thetas.for.plotting  <- axial.results[[3]]   # x values for plot

# next get the values for plotting the calculated free energy
# as a function of iteration number:

axial.free.energy.for.each.iteration <- axial.results[[5]]
axial.iterations.for.plotting <- axial.results[[6]]  # starts at 0 and ends at
#            length(axial.free.energy.for.each.iteration) - 1
# The first value of the free energy is that for the initial ODF (considered as the ODF
# at iteration 0); at each iteration the value of the free energy for the ODF obtained
# in the previous iteratation step is calculated (based on the quantities readily
# available at that point of the computation).

axial.vector.of.values <- axial.results[[7]]
# values of interest such as -entropy and the order parameter
names.for.vector.of.values <- axial.results[[8]]

planar.results <- calculate_ODF(planar.initial.f, B, tau.conv, max.num.iter, min.iterations)
```

```
## [1] "run of    calculate_ODF    January 27, 2023  version"
```

```r
planar.f.for.plotting <- planar.results[[2]]

# for plotting the calculated free energy as a function of iteration number
planar.free.energy.for.each.iteration <- planar.results[[5]]
planar.iterations.for.plotting <- planar.results[[6]]

planar.vector.of.values <- planar.results[[7]]

planar.line.color <- "darkseagreen"
axial.line.color <- "blueviolet"

# plot the ODFs


plot(thetas.for.plotting, axial.f.for.plotting, lwd = 4, ,
        type = "l", col = axial.line.color, xaxt = "n", xlab = " ", ylab = 'ODF value')
# request tic marks and labels for the x-axis:
axis(1, at = c(0, pi/4,  pi/2, 3*pi/4,  pi),
         labels = c("0", expression(paste(pi,"/4")), expression(paste(pi,"/2")),
                           expression(paste("3",pi,"/4")),  expression(pi)))
title(main = NULL, sub = expression(theta), cex.sub = 1.4)

lines(thetas.for.plotting, planar.f.for.plotting, lty = "solid", lwd = 4,
        col = planar.line.color, type = "l")

text(1.6, 6, "planar", col = planar.line.color, cex = 1.5)
text(0.8, 30.0, "axial", col = axial.line.color, cex = 1.5)
```

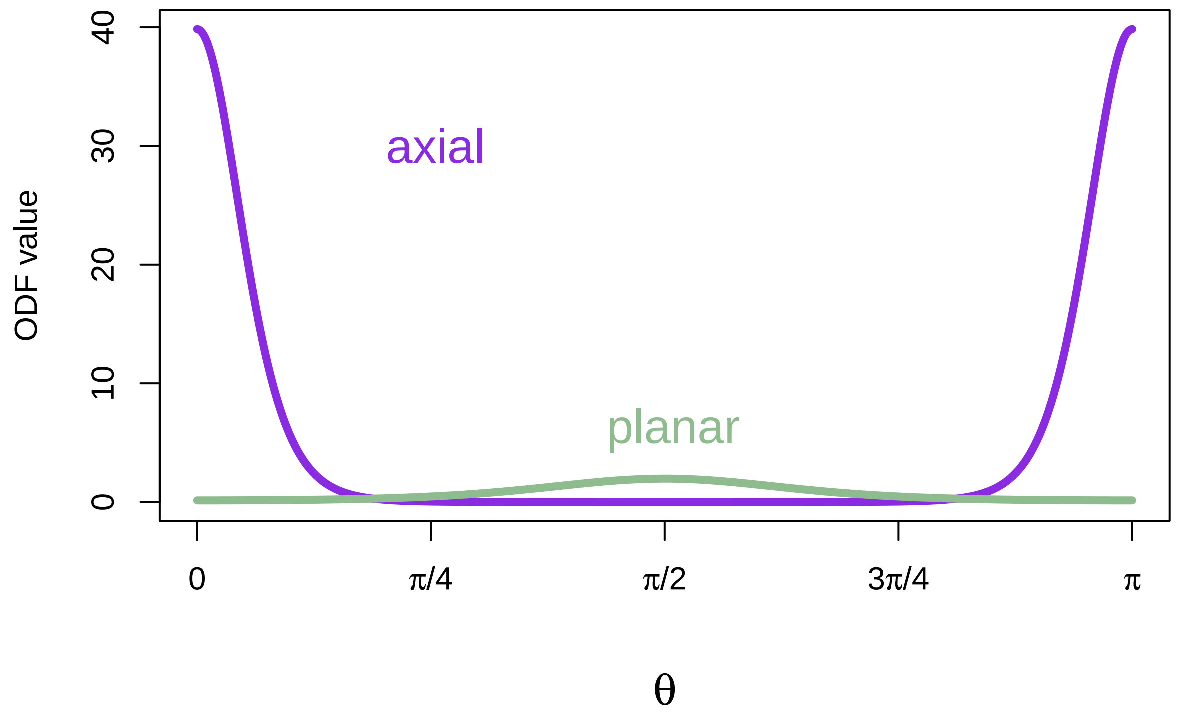


Figure legend: Axial and planar shaped orientation distribution functions for B = 14 that are local minima for the hard core reference system. With the particle interaction term kernel $W(\gamma) = \sin(\gamma)$, particle alignment ($\gamma = 0$ or π) is favored. The planar shaped local minimum ODF balances unfavorable particle interactions (at θ around $\pi/2$) with higher entropy from a more dispersed orientation distribution.

```
## plot the free energy vs. iteration number
# planar.line.color <- "darkseagreen"
# axial.line.color <- "blueviolet"

# axial.free.energy.for.each.iteration <- axial.results[[5]]
# axial.iterations.for.plotting <- axial.results[[6]]  # starts at 0 and ends at
#                   length(axial.free.energy.for.each.iteration) - 1

line.color <- planar.line.color                              #     "cyan3"
plot(planar.iterations.for.plotting,  planar.free.energy.for.each.iteration, lwd = 4, ,
        type = "l", col = line.color, xlab = 'iteration number ', ylab = 'Free Energy',
        ylim = c(4.5, 5.5))

lines(axial.iterations.for.plotting, axial.free.energy.for.each.iteration, lty = "solid",
        lwd = 4, col = axial.line.color, type = "l")
```

```
text(15, 5.3,  "planar", col = planar.line.color, cex = 1.5)
text(7.0, 4.8,  "axial", col = axial.line.color, cex = 1.5)
```

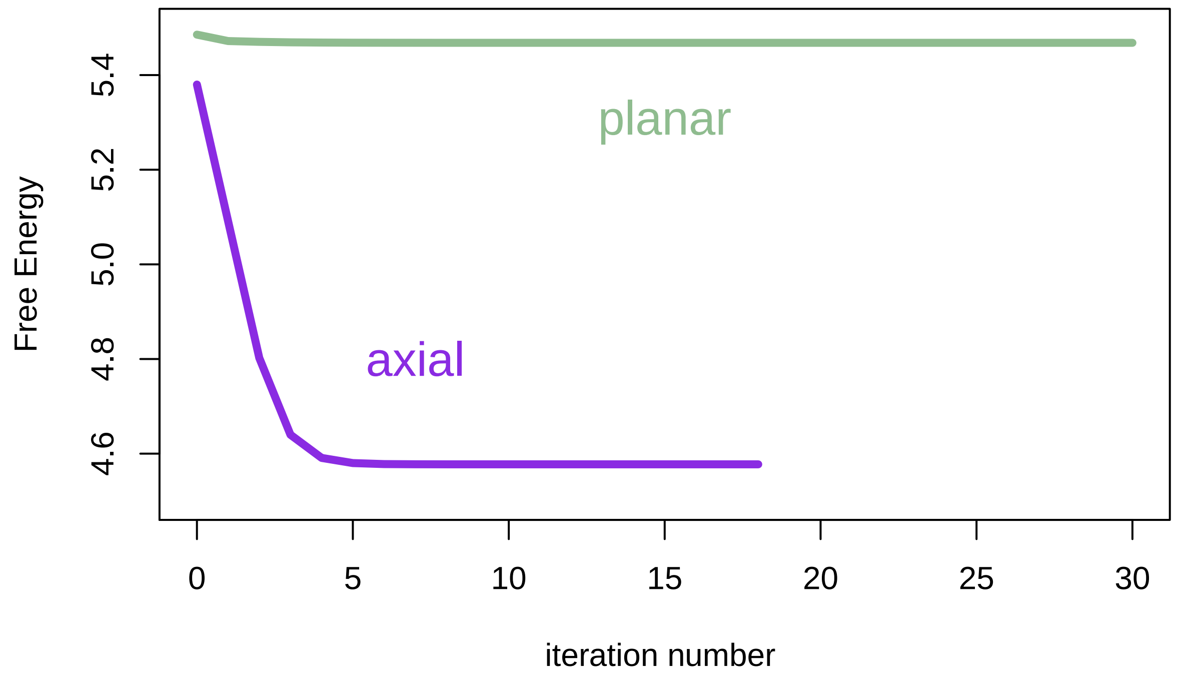


Figure Legend: Free energy vs. iteration number for axial and planar shaped initial ODFs for B = 14 for the hard core reference system. The initial ODF for obtaining the axial solution is a U shaped ODF with peaks at theta = 0 and theta = pi. The initial ODF for obtaining the planar solution is an ODF with a peak at theta = pi / 2 and values near 0 at theta = 0 and theta = pi. In both cases the free energy monotonically decreased at each iteration. This is expected since the free energy function for the hard core reference system satisfies the criteria given in [Berger submitted] for the iteration to decrease the free energy at each step. The ODF at iteration 0 is the one used as the starting value for the calculation.

```r
## Plot the entropy contribution to the free energy, which is actually
# minus the entropy, since high entropy is generally favored, subject to the
# influence of the particle interaction term in the free energy (and the
# applied field, if present)

rm(list = ls())  # clear out any "left over" variables (R objects) (usually a good idea)
##### note rm in R is not the Unix rm that permanently removes files

# columns in data files for varaibles to plot
col_for_free_energy <- 12
col_for_B <- 1
col_for_f_logf <- 9
col_for_P2f <- 10

dirstr <- "C:/berger/orientationdistr/ODF_R_fcns"   # folder with data files and R code
# have previously calculated orientation distribution functions and associated
# values (e.g., <ln f>) for multiple values of B and saved a file containing the results

# axial data
filename <- "results.for.several.B.axial.ODFs.8Feb2024_Ntheta512.tab"  # tab delimited text file
full.file.name <- paste(dirstr, "/", filename, sep = "")  # the full path
axial.data.df <- read.table(full.file.name, header = TRUE, sep = "\t", quote = "\"",
            dec = ".", row.names = NULL, strip.white = TRUE, check.names = FALSE,
            blank.lines.skip = TRUE, stringsAsFactors = FALSE)

# results = 'hide' would suppress printout of results from the
# section of R code when knitr is run
# echo = FALSE would suppress printing the R code in the output

# planar data
filename <- "results.for.several.B.planar.ODFs.8Feb2024_Ntheta512.tab"  # tab delimited text file
full.file.name <- paste(dirstr, "/", filename, sep = "")  # the full path
planar.data.df <- read.table(full.file.name, header = TRUE, sep = "\t", quote = "\"",
            dec = ".", row.names = NULL, strip.white = TRUE, check.names = FALSE,
            blank.lines.skip = TRUE, stringsAsFactors = FALSE)

planar.flogf <- planar.data.df[[col_for_f_logf]]
axial.flogf <- axial.data.df[[col_for_f_logf]]
Bvalues.axial <- axial.data.df[[col_for_B]]
Bvalues.planar <- planar.data.df[[col_for_B]]
# Bmin, Bmax for both axial and planar are 8, 14
# planar flogf ranges from essentially 0 to 0.215 upper bound
# axial  flogf ranges from essentially 0 to 2.45 upper bound

# plot axial and planar <f log f> vs B values


planar.line.color <- "darkseagreen"
plot(Bvalues.planar, planar.flogf , lwd = 4, xlim = c(8, 14), ylim = c(0, 2.5),
        type = "l", col = planar.line.color, xlab = 'B', yaxt = "n",
        ylab = expression(paste("<ln f(",theta,")>")))
axis(2, cex.lab = 1.5, at = c(0, 0.5, 1.0, 1.5, 2.0, 2.5),
         labels = c("0", "0.5", "1.0", "1.5", "2.0", "2.5"))
```

```
axial.line.color <- "blueviolet"
lines(Bvalues.axial, axial.flogf, lty = "solid", lwd = 4, col = axial.line.color, type = "l")

lines(c(10.18592, 10.18592), c(-0.087, -0.045), lty = "solid", lwd = 1.8,
        col = "black", type = "l")

text(11.57, 0.6, "planar initial ODFs", col = planar.line.color, cex = 1.5)
text(10.0, 2.1, "axial initial ODFs", col = axial.line.color, cex = 1.5)
```

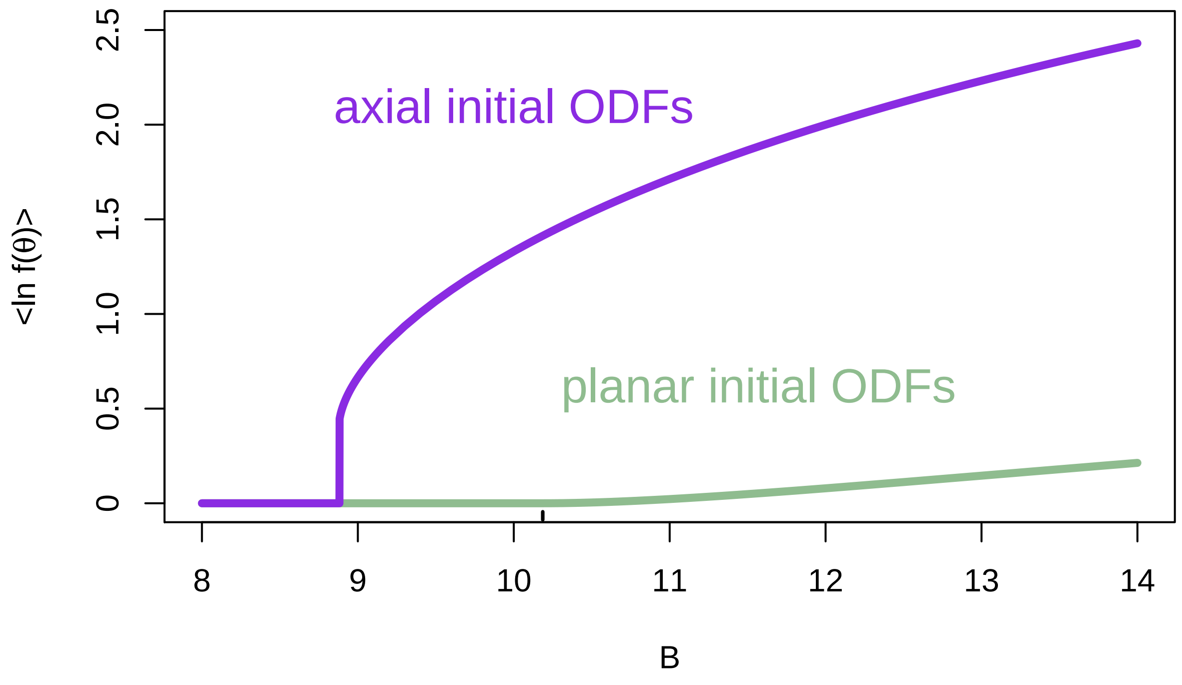


Figure Legend: $<\ln f(\theta)>$ vs. B for converged ODFs for the hard core reference system, resulting from starting the iteration with axial and with planar shaped ODFs. $<\ln f(\theta)>$ is the entropy contribution $\frac{1}{2}\int_0^\pi f(\theta)\ln f(\theta)\sin\theta\, d\theta$ to the free energy. Note the jump in this plot for the axial ODFs around B $= 8.883$ which corresponds to the location ρ_b of Kayser and Ravaché [12] (their $\rho^* = 4$ corresponds to the bifurcation location B $= 32/\pi$ at which planar ODFs that are local minima bifurcate from the isotropic ODF, giving a conversion factor of $8/\pi$). A short vertical line is drawn above the x-axis at this value of B (10.18592).

```r
## Plot the order parameter <P2(cos theta)>
rm(list = ls())  # remove any "left over" R objects  (usually a good idea)
##### note rm in R is not the Unix rm that permanently removes files

# columns in data files for varaibles to plot
col_for_free_energy <- 12
col_for_B <- 1
col_for_f_logf <- 9
col_for_P2f <- 10

dirstr <- "C:/berger/orientationdistr/ODF_R_fcns"   # folder with data files and R code
# have previously calculated orientation distribution functions and associated
# values (e.g., <ln f>) for multiple values of B and saved a file containing the results

# axial data
filename <- "results.for.several.B.axial.ODFs.8Feb2024_Ntheta512.tab" # tab delimited text file
full.file.name <- paste(dirstr, "/", filename, sep = "")  # the full path
axial.data.df <- read.table(full.file.name, header = TRUE, sep = "\t", quote = "\"",
            dec = ".", row.names = NULL, strip.white = TRUE, check.names = FALSE,
            blank.lines.skip = TRUE, stringsAsFactors = FALSE)

# results = 'hide' would suppress printout of results from the
# section of R code when knitr is run
# echo = FALSE would suppress printing the R code in the output

# planar data
filename <- "results.for.several.B.planar.ODFs.8Feb2024_Ntheta512.tab"   # tab delimited text file
full.file.name <- paste(dirstr, "/", filename, sep = "")  # the full path
planar.data.df <- read.table(full.file.name, header = TRUE, sep = "\t", quote = "\"",
            dec = ".", row.names = NULL, strip.white = TRUE, check.names = FALSE,
            blank.lines.skip = TRUE, stringsAsFactors = FALSE)

planar.P2f <- planar.data.df[[col_for_P2f ]]
axial.P2f <- axial.data.df[[col_for_P2f ]]
Bvalues.axial <- axial.data.df[[col_for_B]]
Bvalues.planar <- planar.data.df[[col_for_B]]
# Bmin, Bmax for both axial and planar are 8, 14
# planar P2f ranges from essentially  -0.264 to 0  upper bound
# axial  P2f ranges from essentially 0 to 0.904 upper bound

# plot axial and planar <P2f> vs B values
# note B values for axial and planar runs could be different

planar.line.color <- "darkseagreen"
plot(Bvalues.planar, planar.P2f , lwd = 4, xlim = c(8, 14), ylim = c(-0.40, 1.00),
        type = "l", col = planar.line.color, xlab = 'B', yaxt = "n",
        ylab = expression(paste("<P2(cos ",theta,")>")))
axis(2, cex.lab = 1.5, at = c(-0.4, -0.2, 0, 0.2, 0.4, 0.6, 0.8, 1.0),
         labels = c("-0.4", "-0.2", "0", "0.2", "0.4", "0.6", "0.8", "1.0"))

axial.line.color <- "blueviolet"
lines(Bvalues.axial, axial.P2f, lty = "solid", lwd = 4, col = axial.line.color, type = "l")
```

```
text(10.8, 0.2, "planar initial ODFs", col = planar.line.color, cex = 1.5)
text(11.0, 0.6, "axial initial ODFs", col = axial.line.color, cex = 1.5)
```

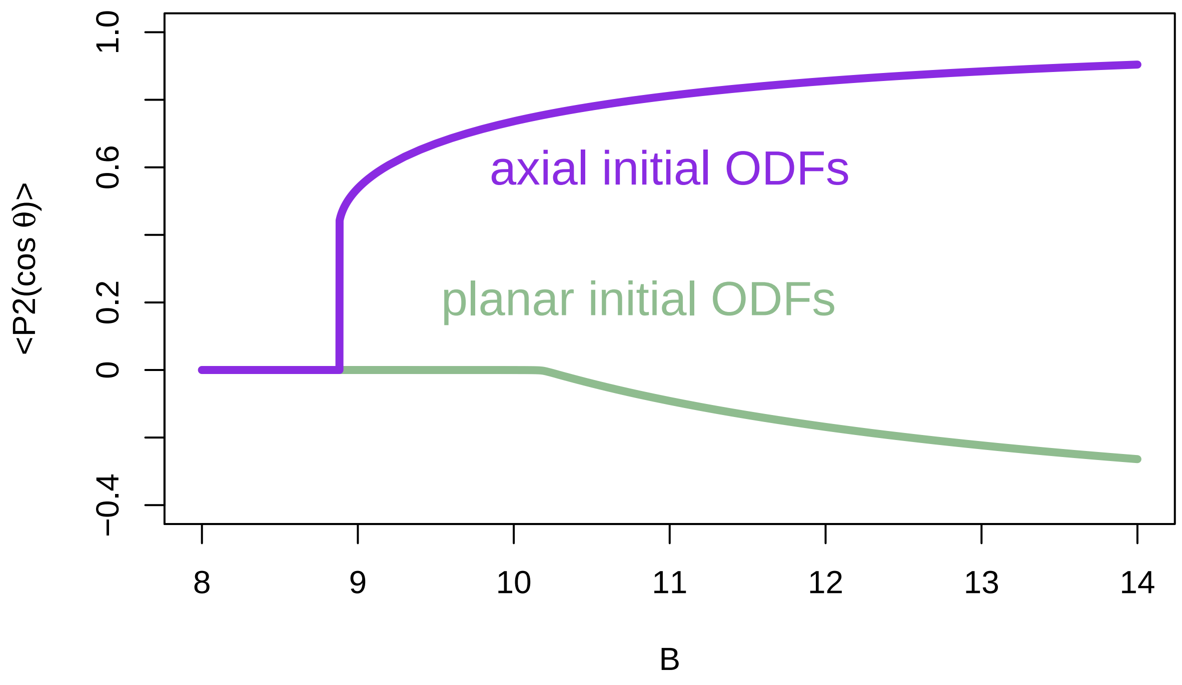


Figure Legend: The order parameter $<P_2(\cos\theta)>$ vs. B for converged ODFs for the hard core reference system, resulting from starting the iteration with axial and with planar shaped ODFs. $<P_2(\cos\theta>$ is $\frac{1}{2}\int_0^\pi f(\theta)P_2(\cos\theta)\sin\theta\,d\theta$. Note the jump in this plot for the axial ODFs around B = 8.883, and the change in slope in the “planar curve" at the bifurcation point $B = 32/\pi$.

```
## Now plot the free energy

# echo = FALSE would suppress code printout

# planar free energy ranges from essentially 3 to 5.5 upper bound
# axial   free energy ranges from essentially 3 to 4.6 upper bound

planar.F <- planar.data.df[[col_for_free_energy]]
axial.F <- axial.data.df[[col_for_free_energy]]
Bvalues.axial <- axial.data.df[[col_for_B]]
Bvalues.planar <- planar.data.df[[col_for_B]]

#
bindices.axial <- which(Bvalues.axial >= 8.88299)    # get >= 8.883 while avoiding rounding error
Baxial <- Bvalues.axial[bindices.axial]
Faxial <- axial.F[bindices.axial]

bindices.planar <- which(Bvalues.planar >= 10.1859)    # get >= 32 / pi = 10.18592 while
#                                                                                     avoiding rounding
Bplanar <- Bvalues.planar[bindices.planar]
Fplanar <- planar.F[bindices.planar]
#
#
#
#
#

planar.line.color <- "darkseagreen"
plot(Bplanar , Fplanar , lwd = 4, xlim = c(8, 14), ylim = c(3.0, 5.5),
        type = "l", col = planar.line.color, xlab = 'B', yaxt = "n",
        ylab = "free energy")
axis(2, cex.lab = 1.5)


axial.line.color <- "blueviolet"
lines(Baxial, Faxial, lty = "solid", lwd = 4, col = axial.line.color, type = "l")

text(11.0, 5.0, "planar", col = planar.line.color, cex = 1.5)
text(12.5, 3.7, "axial", col = axial.line.color, cex = 1.5)

## plot the isotropic free energy
Bisotropic <- Bvalues.axial
Fisotropic <-  Bisotropic * pi / 8    #   (B / 2) * w_0 = B * pi / 8    since    w_0 = pi / 4
lines(Bisotropic, Fisotropic, lty = "dotted", lwd = 4, col = "black", type = "l")
text(8.4, 4.5, pos = 4, "isotropic", col = "black", cex = 1.5)
```

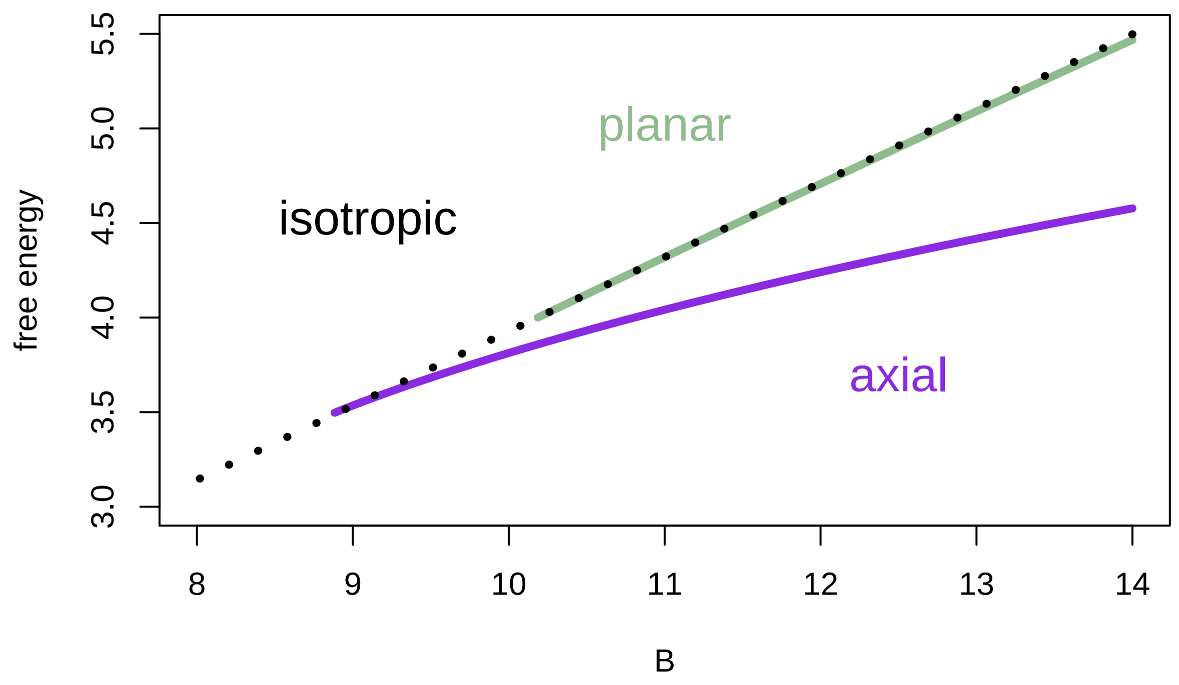


Figure Legend: Free energy F vs. B for axial, planar and isotropic ODFs for the hard core reference system. The free energy for the axial ODF becomes visibly smaller than that for the isotropic ODF as B gets larger. F for the planar ODF and the isotropic ODF are quite close within the range of B values in the plot. F for the planar ODF becomes distinguishably smaller than that for the isotropic ODF toward the right end of the plot. The isotropic ODF is no longer a local minimum of F for B larger than the bifurcation point $32/\pi$. Note the solutions found by the iterative method studied in [HBW 1984] and [Berger, submitted] are local minima of the free energy. The "unstable" bifurcation branch of ODFs (not consisting of local minima of F [KR 1978]) that leads to the axial ODFs that are local minima will not be found using this iteration procedure.

```
## run a different W
## W(gamma) equal sin(gamma) - 8 * P6(cos(gamma))  # P6 is the 6th Legendre polynomial
# starting with an axial shaped initial ODF results in an axial shaped local minimum ODF
# starting with a planar shaped initial ODF results in a local minimum ODF
# that has 2 peaks near where -P6 has maxima

 rm(list = ls())  # clear out any "left over" variables (R objects) (usually a good idea)
 ##### note rm in R is not the Unix rm that permanently removes files

 # calculate the axial and planar ODFs for B = 12
 dirstr <- "C:/berger/orientationdistr/ODF_R_fcns"
```

```
# location of code for calculating ODFs on my computer

filename <- "compute_approx_ODF_27Jan2024.R"
# the name of the file on my computer containing the code for compute_approx_ODF

full.file.name <- paste(dirstr, "/", filename, sep = "")  # the full path

source(full.file.name)  # "compile" the program


# give appropriate values for the arguments of the   compute_approx_ODF   function

N.theta.intervals <- 512  # values of the approximate ODF that this code
#                                                calculates correspond to equally spaced
#                                                theta points inside (0, pi)
N.phi.intervals <- 2048

# 6th Legendre polynomial P6(x)  # x will be cos(gamma)
  P6 <- function(x) (231*x^6 - 315*x^4 + 105*x^2 - 5) / 16


W <- function(gamma) {
   x <- cos(gamma)
   return(sin(gamma) - 8 * (231*x^6 - 315*x^4 + 105*x^2 - 5) / 16)
}

# define initial ODFs
delta.theta <- pi / N.theta.intervals  # length of theta subintervals
num.theta.points <- N.theta.intervals - 1  # theta points interior to (0, pi)
theta.indices <- 1:num.theta.points
theta.vec <- theta.indices * delta.theta
# theta points for values of the calculated discrete ODF
tau.conv <- 1.0e-6  # the bound for the convergence criterion
max.num.iter <- 2000  # maximum number of iterations allowed
min.iterations <- 10  # require having done  min.iterations  iterations before
#                                        start testing for convergence

planar.initial.f <- sin(theta.vec) * sin(theta.vec)  # planar shaped - peak at theta = pi / 2

axial.initial.f <- cos(theta.vec) * cos(theta.vec)  # axial shaped - peaks at theta = 0 and pi
# a convenient U shaped (axial shaped) function on [0, pi]

calculate_ODF <- compute_approx_ODF(N.theta.intervals, N.phi.intervals, W)
```

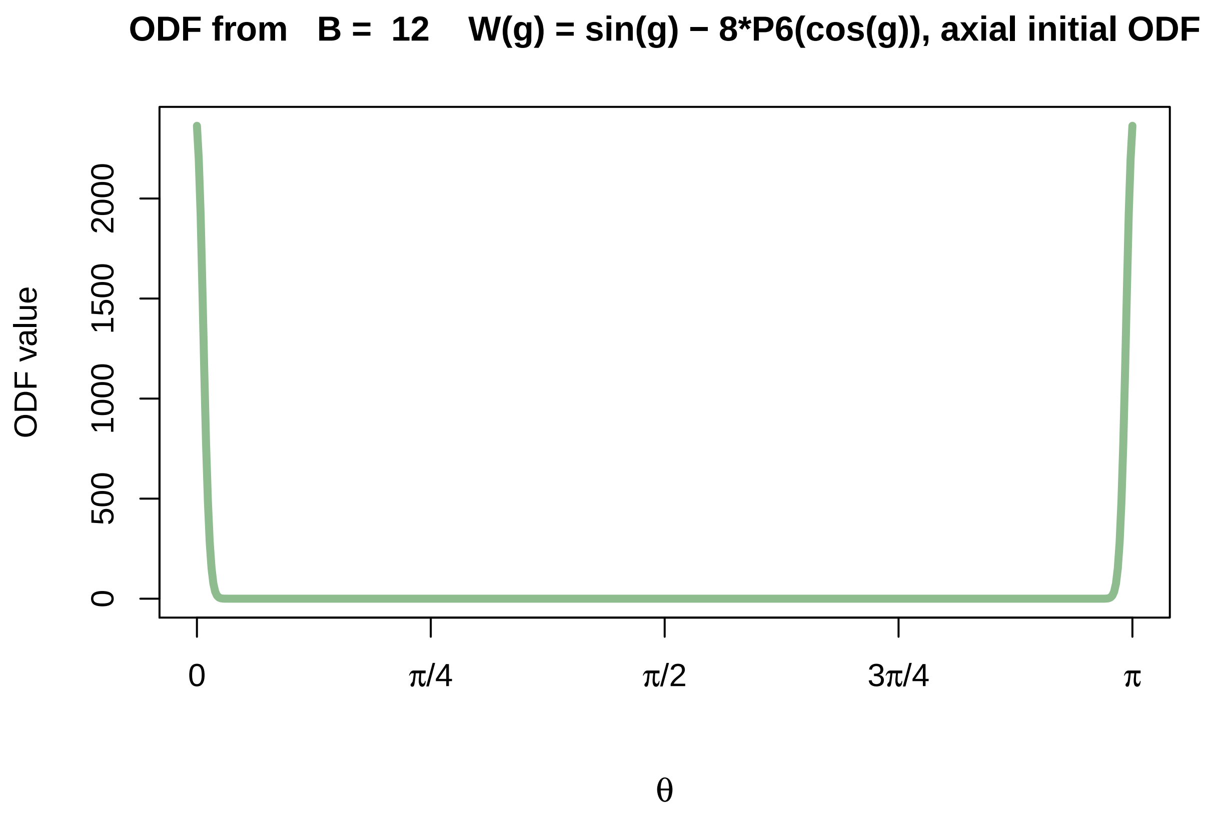
```
## [1] "run of   compute_approx_ODF   January 27, 2024 version"
```

```
B <- 12  # a choice of B to display an ODF with a different shape
results <- calculate_ODF(axial.initial.f, B, tau.conv, max.num.iter, min.iterations)
```

```
## [1] "run of    calculate_ODF    January 27, 2023  version"
```

```r
axial.f.for.plotting <- results[[2]]     # y values for plot
thetas.for.plotting  <- results[[3]]    # x values for plot


line.color <- "darkseagreen"
plot(thetas.for.plotting, axial.f.for.plotting, lwd = 4, ,
        type = "l", col = line.color, xaxt = "n", xlab = ' ', ylab = 'ODF value')
# request tic marks and labels for the x-axis:
axis(1, at = c(0, pi/4,  pi/2, 3*pi/4,  pi),
         labels = c("0", expression(paste(pi,"/4")), expression(paste(pi,"/2")),
                           expression(paste("3",pi,"/4")),  expression(pi)))
title(main = paste("ODF from   B = ", B, "   W(g) = sin(g) - 8*P6(cos(g)), axial initial ODF"),
cex.main = 1.1)
title(main = NULL,
      sub = expression(theta), cex.sub = 1.1)
```



```r
# the axial initial ODF results in the iterative method converging to an axial shaped ODF

# run this choice of W(gamma) with a planar shaped initial ODF
# this yields a different shaped ODF

results <- calculate_ODF(planar.initial.f, B, tau.conv, max.num.iter, min.iterations)
```

```
## [1] "run of    calculate_ODF    January 27, 2023  version"
```

```r
planar.f.for.plotting <- results[[2]]    # y values for plot
thetas.for.plotting  <- results[[3]]   # x values for plot


line.color <- "darkseagreen"
plot(thetas.for.plotting, planar.f.for.plotting, lwd = 4, ,
        type = "l", col = line.color, xaxt = "n", xlab = ' ', ylab = 'ODF value')
# request tic marks and labels for the x-axis:
axis(1, at = c(0, pi/4,  pi/2, 3*pi/4,  pi),
         labels = c("0", expression(paste(pi,"/4")), expression(paste(pi,"/2")),
                           expression(paste("3",pi,"/4")),  expression(pi)))
## title(main = paste("different shape ODF    B = ", B, "   W(g) = sin(g) - 8*P6(cos(g))" ),
## cex.main = 1.1)
title(main = NULL, sub = expression(theta), cex.sub = 1.4)
```

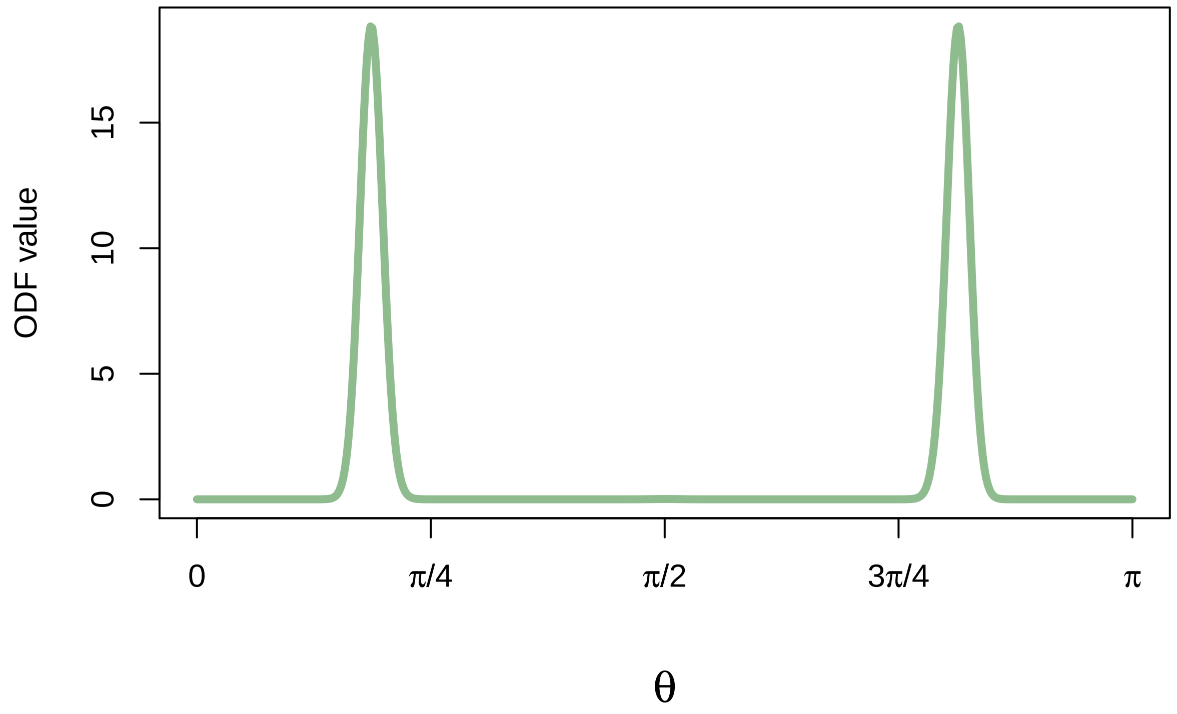


Figure Legend: A different shaped ODF resulting from $W(\gamma) = \sin(\gamma) - 8\,P_6(\cos\gamma))$, B = 12, and a planar shaped initial ODF.

```
# show how to print values of various quantities for the converged ODF
 names.for.vector.of.values <- results[["names.for.vector.of.values"]]
 vector.of.values <- results[["vector.of.values"]]
 df.for.values <- data.frame(names.for.vector.of.values,  vector.of.values,
                                                         stringsAsFactors = FALSE)
 colnames(df.for.values) <- c("variable name", "value")
 df.for.values
```

```
##                   variable name         value
## 1                             B  1.200000e+01
## 2           num_theta_intervals  5.120000e+02
## 3             num.phi.intervals  2.048000e+03
## 4                          f(0)  9.313608e-23
## 5                          fmid  1.165966e-02
## 6                           fpi  9.313608e-23
## 7                             L  3.543001e-02
## 8           pi/4 - <<W(gamma)>>  1.445550e+00
## 9                <ln(f(theta))>  2.424155e+00
## 10             <P2(cos(theta))>  5.363273e-01
## 11             <P4(cos(theta))> -1.206562e-01
## 12                  free.energy -1.536758e+00
## 13               num_iterations  1.100000e+01
## 14 num_iter_F_did_NOT_decrease  0.000000e+00
```

Do a Common Tangent plot for $\mathbf{W}(\gamma) = \sin(\gamma)$, $V(\theta) = 0$

Have already run and saved results for axial and planar ODFs vs B into tab delimited text files. The code for these is in this Repository.

Will read each data file back in as a data frame (df) and do plots. The max and min of the B values for both the axial and planar runs read in here are 14 and 8. For the common tangent plot, will extend (by hand with an easy calculation) relevant data to include B between 8 and 7 (the only minimum of the free energy in this range of B is the isotropic ODF).

```
## Plot    (B min(F) - cB) vs B  where min(F) is the minimum of the free energy for each B
## and c is a suitable constant chosen to have the "action" (variation over the y-axis values)
## fit in the plot

rm(list = ls())  # clear out any "left over" variables (R objects) (usually a good idea)
##### note rm in R is not the Unix rm that permanently removes files

# columns in data files for varaibles to plot
col_for_free_energy <- 12
col_for_B <- 1
col_for_f_logf <- 9
col_for_P2f <- 10

dirstr <- "C:/berger/orientationdistr/ODF_R_fcns"
# folder on my computer with data files and R code

# axial data
  filename <- "results.for.several.B.axial.ODFs.8Feb2024_Ntheta512.tab"   # tab delimited text file
```

```
## filename <-       "results.for.several.B.axial.ODFs.13Feb2024_Ntheta1024.tab"
##           # use N.theta.intervals = 1024 for the common tangent Figure in the 2024 paper and
##           #  do runs for B between 7 and 14

full.file.name <- paste(dirstr, "/", filename, sep = "")  # the full path
axial.data.df <- read.table(full.file.name, header = TRUE, sep = "\t", quote = "\"",
            dec = ".", row.names = NULL, strip.white = TRUE, check.names = FALSE,
            blank.lines.skip = TRUE, stringsAsFactors = FALSE)


# results = 'hide' would suppress printout of results from the
# section of R code when knitr is run
# echo = FALSE would suppress printing the R code in the output


# planar data    # the planar ODF is never a global minimum of the free energy
#                     for the hard core reference system, so we don't need planar ODFs
# for the Common Tangent plot

axial.flogf <- axial.data.df[[col_for_f_logf]]
axial.free.energy <- axial.data.df[[col_for_free_energy]]
Bvalues.axial <- axial.data.df[[col_for_B]]
#Bvalues.planar <- planar.data.df[[col_for_B]]
# Bmin, Bmax for both axial and planar are 8, 14


minF.line.color <- "coral1"
isotropic.line.color <- "blueviolet"

# Note beyond the the bifurcation point $B = 32 / \pi$ = 10.18592 where the planar ODF arises,
# the isotropic ODF is no longer a local minimum.  The free energy of the planar solution
# (which exists for $B > 32 / \pi$) is always below that of the isotropic solution.


# Bmin, Bmax for both axial and planar are 8, 14  for the Feb 8, 2024 data


# for the common tangent graph, we will also want values of the
# free energy for the isotropic solution
# for B between 7.0 and 8.0 (the axial solution does not arise until about B = 8.883)

B_8_to_7 <- seq(from = 8.0, to = 7.0, by = -0.05)
isotropic.free.energy_8_to_7  <-  (B_8_to_7 / 2) * pi / 4
BFisotropic_8_to_7  <-  B_8_to_7 * isotropic.free.energy_8_to_7




# isotropic free energy is (B / 2) * pi / 4
B.to.plot.isotropic <- Bvalues.axial    # goes from 14 down to 8
isotropic.free.energy <- (B.to.plot.isotropic / 2) * pi / 4
BFisotropic <- B.to.plot.isotropic * isotropic.free.energy

BFaxial <- Bvalues.axial * axial.free.energy
```

```r
B.minF <- pmin(BFisotropic, BFaxial)  # parallel minimum (min for each vector index)

# now include the data for B between 8 and 7

Bvalues.axial <- c(Bvalues.axial, B_8_to_7)
B.minF <- c(B.minF, BFisotropic_8_to_7)

# translate by a constant slope line to get the variation to be within a small
# range of y-axis values

slope1 <- 6.4387

yk <- 25.20


yvec <- slope1 * Bvalues.axial

BminF.transl <- B.minF - yvec

# first have R set up axis ranges

plot(Bvalues.axial, BminF.transl, lty = "solid", lwd = 2 , type = "l", col = minF.line.color,
     xlab = "B", ,  ylab = 'B(F - C)', cex.lab = 1.3, ylim = c(-26.5, -25.8))
# lty = "blank" would skip drawing the curve

ylevel <-  -0.083  -8*slope1 + yk
lines(c(8, 12), c(ylevel, ylevel), lty = "solid", lwd = 1,
       col = isotropic.line.color, type = "l")
```

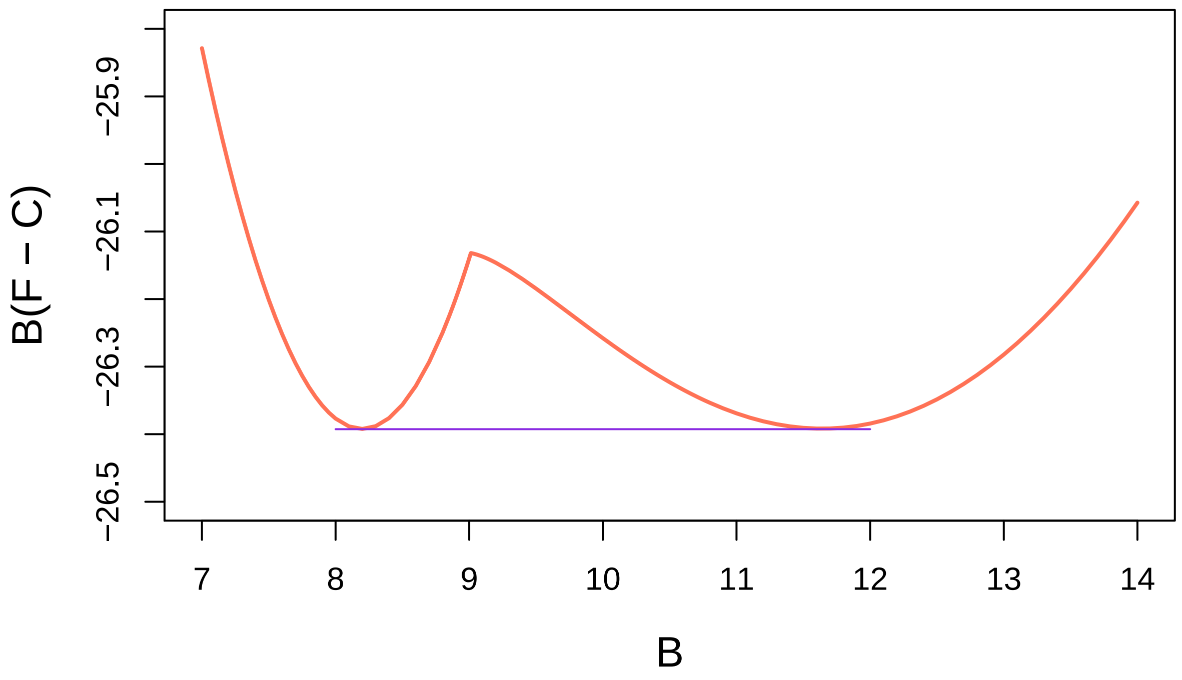


Figure Legend: The common tangent construction in which the free energy per particle, F, is multiplied by B to give an energy per unit volume, with the zero of energy per particle adjusted to give a horizontal common tangent by choosing $C = 6.4387$.

do plot for ODFs not symmetric about pi over 2 from W and V not symmetric about pi/2

Run W(gamma) <- - (7 * pi / 32) * P3(cos(gamma)) and V = -cos(theta)

Calculate and plot the ODFs

```
rm(list = ls())  # clear out any "left over" variables (R objects) (usually a good idea)
##### note rm in R is not the Unix rm that permanently removes files

dirstr <- "C:/berger/orientationdistr/ODF_R_fcns"
# location of code for calculating ODFs on my computer

filename <- "compute_approx_ODF_27Jan2024.R"
# the name of the file on my computer containing the code for compute_approx_ODF

full.file.name <- paste(dirstr, "/", filename, sep = "")  # the full path

source(full.file.name)  # "compile" the program
```

```
# Some Legendre polynomials

P2 <- function(x)  (3 * x^2 - 1) / 2
P4 <- function(x)  (35*x^4 - 30*x^2 + 3) / 8
P6 <- function(x)  (231*x^6 - 315*x^4 + 105*x^2 - 5) / 16
P3 <- function(x)  (5*x^3 - 3*x) / 2


# W(gamma) is the particle interaction kernel function
# W <- function(g) sin(g)  is the W for the hard core reference system

## here  set  W  to be   - (7 * pi / 32) * P3(cos(gamma))

W <- function(g) {
   x <- cos(g)
    value <-     - (7 * pi / 32) * P3(x)
   return(value)
}

# give an applied field
c2 <-    0.1
V <- function(theta)    -c2*cos(theta)



N.theta.intervals <- 512
N.phi.intervals <- 2048     # since phi integrals are only done once for a given
#                                        W and V, can take N.phi.intervals "large"
#                                         without requiring too much computing time

calculate_ODF <- compute_approx_ODF(N.theta.intervals, N.phi.intervals, W, V)
```

```
## [1] "run of   compute_approx_ODF   January 27, 2024 version"
```

```
# returns the function
#    calculate_ODF(initial.f, B, tau.conv, max.num.iter, min.iterations)
# and calculates various quantities that only need to be computed once
# for given values of      N.theta.intervals, N.phi.intervals, W, and V



tau.conv <- 1.0e-6  # the bound for the convergence criterion (see [HBW 1984])
max.num.iter <- 3000  # maximum number of iterations allowed
min.iterations <- 10  # require having done  min.iterations  iterations before
#                                        start testing for convergence

# get initial ODFs for calculating axial and planar shaped ODFs

# Get the vector of theta values used for numerical integration.
# These are the locations where the ODF is being calculated.

delta.theta <- pi / N.theta.intervals  # length of theta subintervals
num.theta.points <- N.theta.intervals - 1   # number of theta points inside (0, theta)
```

```
theta.indices <- 1:num.theta.points
theta.vec <- theta.indices * delta.theta

# Specify starting (initial) values for the ODF f
# (here, this is a vector of values at the grid points theta.vec).

# The function   calculate_ODF   will "adjust" the initial values
# so they are positive (any values < 0.0001 are set to 0.0001).
# Then   calculate_ODF   will scale the initial values, meaning

#
planar.initial.f <- sin(theta.vec) * sin(theta.vec)  # planar shaped - peak at theta = pi / 2
axial.initial.f <- cos(theta.vec) * cos(theta.vec)  # axial shaped - peaks at theta = 0 and pi
P4.initial.f <-    -P4(cos(theta.vec))  # gets adjusted to be an admissible positive ODF
#                                                   within the  calculate_ODF  function

# break symmetry
index1 <- as.integer(N.theta.intervals / 2)
index2 <- as.integer(num.theta.points)
axial.initial.f[index1:index2] <- 0.

second.axial.initial.f <- rev(axial.initial.f)

# for plots
second.axial.line.color <- "darkseagreen"
axial.line.color <- "blueviolet"
planar.line.color <- "darkseagreen"
P4.line.color <- "deeppink1"

B <- 12.0

axial.results <- calculate_ODF(axial.initial.f, B, tau.conv, max.num.iter, min.iterations)
```

```
## [1] "run of    calculate_ODF    January 27, 2023  version"
```

```
# calculate_ODF carries out iterations starting with the initial ODF (its first argument)
# and continuing to iterate until convergence is achieved or the maximum number of
# iterations is done

axial.f.for.plotting <- axial.results[[2]]    # y values for plot
thetas.for.plotting  <- axial.results[[3]]   # x values for plot


axial.vector.of.values <- axial.results[[7]]   # values from the ODF at the end of the iteration
names.for.vector.of.values <- axial.results[[8]] # use this as names for data frame rows or columns

# display these values using a data frame
df.for.values <- data.frame(names.for.vector.of.values,  axial.vector.of.values,
                              stringsAsFactors = FALSE)
colnames(df.for.values) <- c("variable name", "value")
## df.for.values    # don't do the print

plot(thetas.for.plotting, axial.f.for.plotting, lwd = 4, ,
```

```r
        type = "l", col = axial.line.color, xaxt = "n", xlab = " ", ylab = 'ODF value')
# request tic marks and labels for the x-axis:
axis(1, at = c(0, pi/4,  pi/2, 3*pi/4,  pi),
         labels = c("0", expression(paste(pi,"/4")), expression(paste(pi,"/2")),
                          expression(paste("3",pi,"/4")),  expression(pi)))
title(main = NULL, sub = expression(theta), cex.sub = 1.4)


# Now calculate and plot the ODFs resulting from the second axial initial f


B <- 12.0

second.axial.results <- calculate_ODF(second.axial.initial.f, B, tau.conv,
                                                          max.num.iter, min.iterations)
```

```
## [1] "run of    calculate_ODF    January 27, 2023  version"
```

```r
# calculate_ODF carries out iterations starting with the initial ODF (its first argument)
# and continuing to iterate until convergence is achieved or the maximum number of
# iterations is done

second.axial.f.for.plotting <- second.axial.results[[2]]    # y values for plot

second.axial.vector.of.values <- second.axial.results[[7]]
# values from the ODF at the end of the iteration

names.for.vector.of.values <- second.axial.results[[8]]
# use this as names for data frame rows or columns

# display these values using a data frame
df.for.values <- data.frame(names.for.vector.of.values,  second.axial.vector.of.values,
                              stringsAsFactors = FALSE)
colnames(df.for.values) <- c("variable name", "value")
## df.for.values   # don't do the print


lines(thetas.for.plotting, second.axial.f.for.plotting, lty = "dashed", lwd = 4,
        col = second.axial.line.color, type = "l")
```

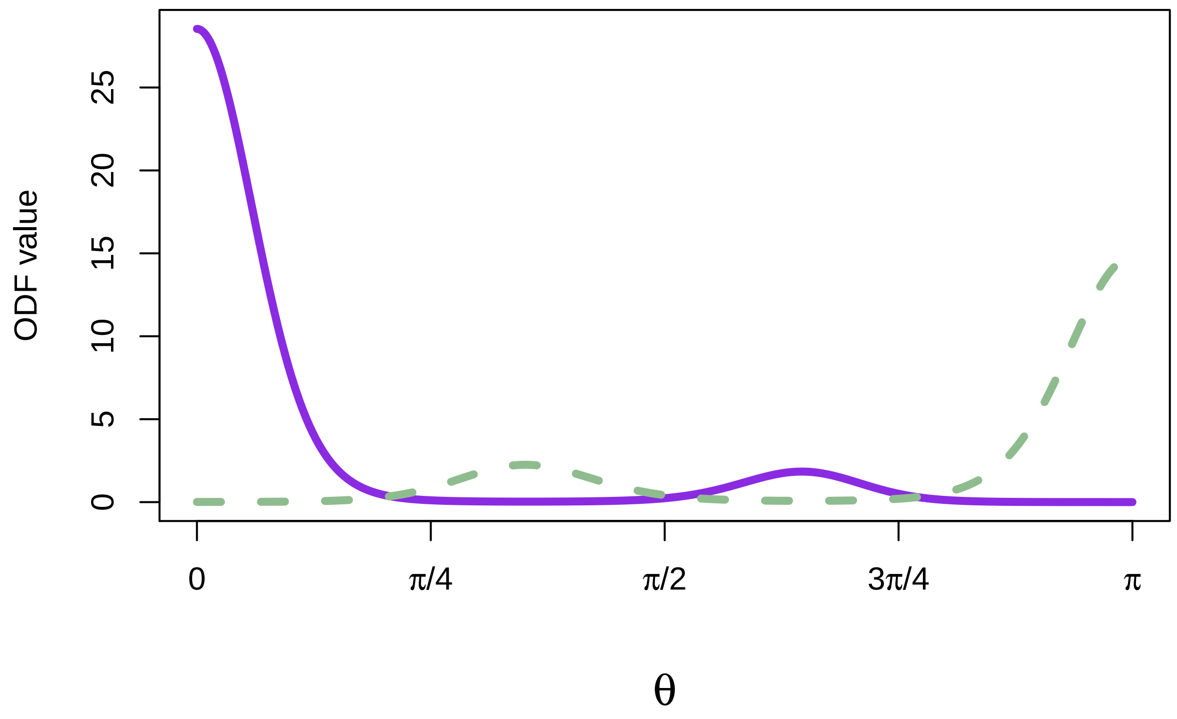


Figure legend: ODF local minima with W and the applied field V not symmetric about $\pi/2$. Setting $W(\gamma) = -(7\pi/32)P_3(\cos\gamma)$, $V(\theta) = -0.1\cos(\theta)$, and $B = 12$ results in the pair of local minima displayed here. This applied field results in lower free energy from orientations around $\theta = 0$ and increased free energy from orientations near $\theta = \pi$.

do plot for W(gamma) equal sin(gamma) plus P4(cos gamma), V = 0, B equal 10.8

This W does NOT satisfy the conditions given in [Berger submitted] for the iteration to decrease the free energy at each step, and the iteration in this case does not converge.

Calculate and plot the ODFs

```r
rm(list = ls())  # clear out any "left over" variables (R objects) (usually a good idea)
##### note rm in R is not the Unix rm that permanently removes files

dirstr <- "C:/berger/orientationdistr/ODF_R_fcns"
# location of code for calculating ODFs on my computer

filename <- "compute_approx_ODF_27Jan2024.R"
# the name of the file on my computer containing the code for compute_approx_ODF
```

```
full.file.name <- paste(dirstr, "/", filename, sep = "")  # the full path

source(full.file.name)  # "compile" the program

# Some Legendre polynomials

P2 <- function(x)  (3 * x^2 - 1) / 2
P4 <- function(x)  (35*x^4 - 30*x^2 + 3) / 8
P6 <- function(x)  (231*x^6 - 315*x^4 + 105*x^2 - 5) / 16
P3 <- function(x)  (5*x^3 - 3*x) / 2
# the arguments of the compute_approx_ODF function are:

# W(gamma) is the particle interaction kernel function
# W <- function(g) sin(g)  is the W for the hard core reference system


W <- function(g) sin(g) + P4(cos(g))


# V(theta) is the applied field, the default is  V = 0   in the
# function       compute_approx_ODF
# so there is no need to specify V when calling compute_approx_ODF
# if want zero applied field

N.theta.intervals <- 512
N.phi.intervals <- 2048     # since phi integrals are only done once for a given
#                                        W and V, can take N.phi.intervals "large"
#                                         without requiring too much computing time

calculate_ODF <- compute_approx_ODF(N.theta.intervals, N.phi.intervals, W)
```

```
## [1] "run of   compute_approx_ODF   January 27, 2024 version"
```

```
# returns the function
#    calculate_ODF(initial.f, B, tau.conv, max.num.iter, min.iterations)
# and calculates various quantities that only need to be computed once
# for given values of      N.theta.intervals, N.phi.intervals, W, and V

# See the extensive comments in the file:   compute_approx_ODF_27Jan2024.R
# for how to use the R functions:  compute_approx_ODF  and   calculate_ODF
# (including details on what their arguments are and how the functions work).


tau.conv <- 1.0e-6  # the bound for the convergence criterion (see [HBW 1984])
max.num.iter <- 32  # maximum number of iterations allowed
min.iterations <- 10  # require having done  min.iterations  iterations before
#                                        start testing for convergence

# get initial ODFs for calculating axial and planar shaped ODFs

# Get the vector of theta values used for numerical integration.
# These are the locations where the ODF is being calculated.
```

```r
delta.theta <- pi / N.theta.intervals  # length of theta subintervals
num.theta.points <- N.theta.intervals - 1   # number of theta points inside (0, theta)
theta.indices <- 1:num.theta.points
theta.vec <- theta.indices * delta.theta

# Specify starting (initial) values for the ODF f
# (here, this is a vector of values at the grid points theta.vec).

# The function   calculate_ODF   will "adjust" the initial values
# so they are positive (any values < 0.0001 are set to 0.0001).
# Then   calculate_ODF   will scale the initial values, meaning
# all the values are multiplied by the same positive constant) such
# that the integral of the starting ODF f
#       (1/2) integral over [0, pi] of f(theta) sin(theta) d theta
# will equal 1   (as calculated by trapezoidal numerical integration)
#
planar.initial.f <- sin(theta.vec) * sin(theta.vec)  # planar shaped - peak at theta = pi / 2
axial.initial.f <- cos(theta.vec) * cos(theta.vec)  # axial shaped - peaks at theta = 0 and pi
P4.initial.f <-    -P4(cos(theta.vec))  # gets adjusted to be an admissible positive ODF
#                                                     within the  calculate_ODF  function


# for plots
second.axial.line.color <- "darkseagreen"
axial.line.color <- "blueviolet"
planar.line.color <- "darkseagreen"
P4.line.color <- "deeppink1"

B <- 10.8

axial.results <- calculate_ODF(axial.initial.f, B, tau.conv, max.num.iter, min.iterations)
```

```
## [1] "run of    calculate_ODF    January 27, 2023  version"
## iter =  2   F did not decrease   only printing for first 12 instances
## iter =  3   F did not decrease   only printing for first 12 instances
## iter =  4   F did not decrease   only printing for first 12 instances
## iter =  5   F did not decrease   only printing for first 12 instances
## iter =  6   F did not decrease   only printing for first 12 instances
## iter =  7   F did not decrease   only printing for first 12 instances
## iter =  8   F did not decrease   only printing for first 12 instances
## iter =  10   F did not decrease   only printing for first 12 instances
## iter =  12   F did not decrease   only printing for first 12 instances
## iter =  14   F did not decrease   only printing for first 12 instances
## iter =  16   F did not decrease   only printing for first 12 instances
## iter =  18   F did not decrease   only printing for first 12 instances
## [1] "WARNING iteration did not converge"
```

```r
# calculate_ODF carries out iterations starting with the initial ODF (its first argument)
# and continuing to iterate until convergence is achieved or the maximum number of
# iterations is done

axial.f.for.plotting <- axial.results[[2]]    # y values for plot
thetas.for.plotting  <- axial.results[[3]]   # x values for plot
```

```
# values for plotting the calculated free energy as a function of iteration number:

axial.free.energy.for.each.iteration <- axial.results[[5]]
axial.iterations.for.plotting <- axial.results[[6]]  # starts at 0 and ends at
#             length(axial.free.energy.for.each.iteration) - 1
# The first value of the free energy is that for the initial ODF (considered as the ODF
# at iteration 0); at each iteration the value of the free energy for the ODF obtained
# in the previous iteratation step is calculated (based on the quantities readily
# available at that point of the computation).

line.color <- "darkviolet"
plot(axial.iterations.for.plotting[1:31], axial.free.energy.for.each.iteration[1:31], lwd = 4,
        type = "l", col = line.color, xlab = 'iteration number ', ylab = 'Free Energy')
```

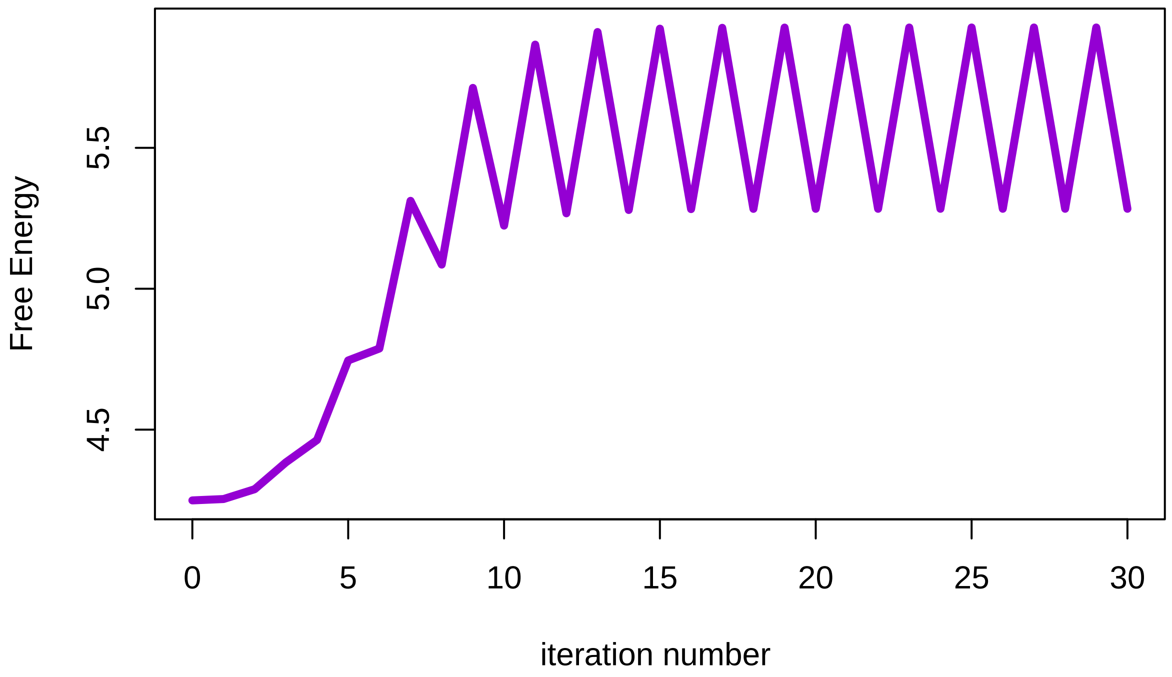


Figure legend: The free energy vs. iteration number for $W(\gamma) = \sin(\gamma) + P_4(\cos\gamma)$, $V = 0$ and $B = 10.8$ starting from an axial ODF. This W does not satisfy the conditions [Berger submitted] for the iteration to decrease the free energy at each step since the coefficient w_4 in the expansion of $W(\gamma)$ in terms of Legendre polynomials $P_m(\cos\gamma)$ equals $1 - 9\pi/256$ which is > 0, and the iteration fails to converge. The oscillation in the free energy that is established by iteration 30 was seen to continue through 3000 iterations.