Homework 6

Particle Swarm Optimization using Schwefel Benchmark Function

Daniel Carpenter

April 2022

Contents

Global Variables

Input variables like the *random seed*, and the *lower and upper bound of solution space*
Please assume these are referenced by following code chunks

```
import math
from random import Random
import numpy as np

# Random seed
seed = 12345
randNumGenerator = Random(seed)

# bounds for Schwefel Function search space
lowerBound = -500
upperBound = 500
```

1 *Question 1:* Particle Swarm Optimization

Find the global optimium of the `Schwefel` function below

$$f(x) = 418.982887272433n - \sum_{i=1}^{n} x_i \sin(\sqrt{|x_i|})$$

Dimensions:	n
Domain:	$-500.0 \leq x_i \leq 500.0$
Global Optimum:	$f(x) \approx 0.0$ at $x = (420.9687, 420.9687, \ldots, 420.9687)$

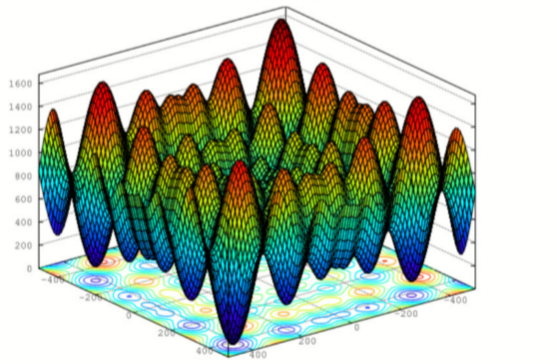


Figure 1: 2D Schwefel Function

The Following Functions Satisfy Problem 1(a)

1.1 Schwefel Function

```
# ========================================================================
# SCHWEFEL FUNCTION
# Schwefel function to evaluate a real-valued solution x
# note: the feasible space is an n-dimensional hypercube centered at the origin with side length = 2 * 
# ========================================================================
def evalFitnessVal(x):
    fitnessValue = 0
    numParticles = len(x)

    # For every particle particle, calculate the fitness value based on Schwefel function
    for particle in range(numParticles):
        fitnessValue = fitnessValue + x[particle]*math.sin(math.sqrt(abs(x[particle])))

    fitnessValue = 418.9829*numParticles - fitnessValue

    return fitnessValue
```

1.2 Functions for the Global and Local Best Logic (*using `Ring Structure`*)

- Local best topology method: `Ring` structure
- The driver function takes in a `string` called `local` or `global`
- This parameter switches between these two methods for the driver
- For example, if I chose `local`, then it would use the `local` best method
- See inline comments for detailed logic

```
# ========================================================================
# GLOBAL MIN VALUE AND POSITION SEARCH FUNCTION
# Returns the 2 element list (each containing a single value) with the global best particle's:
# ---- [0] min value and
# ---- [1] associate position of
# ========================================================================
def getGlobalBest(fitnessValues, positions, swarmSize):
    minValue = np.min(fitnessValues)          # Find the Minimum fitness value of all particles
    minIndex = fitnessValues.index(minValue) # Find the index of the position for the min. fit. value

    minPosition = positions[minIndex][:] # Now get a copy of the particle's position with min index

    # Returns: the global best particle's minimum fitness value and its position
    return [minValue, minPosition]


# ========================================================================
# LOCAL MIN VALUE AND POSITION SEARCH FUNCTION
# Topology: Ring structure with n neighbors  (default 2)
# Returns the 2 element list of lists with the each particle's local best within neighborhood
# ---- [0] min value and
# ---- [1] associate position of
```

```
# Can change numParticlesInNbrhood to consider more or less in particle's neighborhood
# ========================================================================
def getLocalBest(fitnessValues, positions, swarmSize,
                 numParticlesInNbrhood = 2):  # Number of particles to compare to for local best

    lBestFitValue = [] # will hold the best VALUE    of the n surrounding particles, for each particle
    lBestPosition = [] # will hold the best POSITION of the n surrounding particles, for each particle


    # For every particle in the swarm, (starting at n less than index 0)
    for particle in range(-numParticlesInNbrhood, swarmSize - numParticlesInNbrhood):

        # Identify the two neighbors fitness value of this particle,
        # which are the two precedng particles
        personalBestNeighbor1 = fitnessValues[particle]
        personalBestNeighbor2 = fitnessValues[particle + 1]

        # Identify the lowest fitness value of this particle's the two preciding neighbors
        minNeighValue = min(personalBestNeighbor1, personalBestNeighbor2)

        # Store the index of the particle
        minNeighIndex = fitnessValues.index(minNeighValue)

        # Store the particle's best neighbors fitness value and position
        lBestFitValue.append(fitnessValues[minNeighIndex])
        lBestPosition.append(positions[minNeighIndex])

    # Returns a list of particles and the min of their n best fit. valued neighbors
    return[lBestFitValue, lBestPosition]


# If you needed to index the list just returned for global or local best
VALUE_IDX    = 0
POSITION_IDX = 1
```

1.3 Function to Initialize the Swarm

- Allows you to easily change the swarm size, dimensions, an the local or global best method

```
# ================================================================================
# STEP 1 - SWARM INITIALIZATION / EVALUATION
# Randomly initialize a swarm instance
# Set the partical's best to it's starting position
# ================================================================================
def initializeSwarm(swarmSize, numDimensions, functionToGetBest):

    # In the current time period, position[particle] and velocity[particle] of each particle i,
    # Each particle contains n-dimensional list of the coordinate position & velocity
    position = [[] for _ in range(swarmSize)] # X[particle]: position (2D: x, y) of particle i
    velocity = [[] for _ in range(swarmSize)] # V[particle]: velocity (2D: x, y) of particle i

    # Lists containing info related to each particle in swarm
    pCurrFitValue = []  # X[particle] The current position of particle i


    # For each particle and dimension, randomly initialize the...
    for particle in range(swarmSize):
        for theDimension in range(numDimensions):

            # Position: give random value between lower/upper bounds (-500, 500 for schwefel)
            position[particle].append(randNumGenerator.uniform(lowerBound, upperBound))

            # Velocity: give random value between -1 and 1   --- maybe these are good bounds?  maybe no
            velocity[particle].append(randNumGenerator.uniform(-1, 1))

        # 1.1 - Evaluate fitness value
        pCurrFitValue.append(evalFitnessVal(position[:][particle]))  # evaluate the current position's

    # 1.2 - Log the individual and global bests
    pBestPosition = position[:]       # initialize pBestPosition to the starting position
    pBestFitValue = pCurrFitValue[:]  # initialize pBestPosition to the starting position's value

    # 1.3 - Log the Global or local best (depends on chosen method) fitness value and position
    glBestFitValue, glBestPosition = functionToGetBest(pBestFitValue[:], pBestPosition[:], swarmSize)

    return [position, velocity, pCurrFitValue,
            pBestPosition, pBestFitValue,
            glBestFitValue, glBestPosition]
```

1.4 Function to Update `Velocity` and `Position`

- Uses the velocity calcultion to update the position.
- Function is dynamic and can use the `local` or `global` best methods
- Also handles:
  - *Infeasibility of* `Velocity`: pushes the position back into the lower or upper bound solution space (see logic below)
  - *Infeasibility of* `Position`: Randomly moves above the solution space until the position has found a feasible solution

```
# ================================================================================
# UPDATE VELOCITY AND POSITION
# ================================================================================
def updateVelocityAndPosition(intertiaWeight, velocity, position, phi1, phi2, pBestPosition, glBestPosi
                              swarmSize, numDimensions):
# Velocity ---------------------------------------------------------------------

    ## random weights of r for random velocity adjustment
    r1, r2 = randNumGenerator.random(), randNumGenerator.random()

    ## Calculations of updating velocity, separated by
    ## intertia + cognitive + social (for simplicity)
    vInertia   = np.multiply(intertiaWeight, velocity[:])                          # Interia   componen
    vCognitive = np.multiply(phi1*r1, np.subtract( pBestPosition[:], position[:])) # Cognitive componen
    vSocial    = np.multiply(phi2*r2, np.subtract(glBestPosition[:], position[:])) # Social    componen

    ## Update the new velocity to the summation of intertia, cognitive, and social
    newVelocity =  vInertia[:] + vCognitive[:] + vSocial[:]

    ## Limit the velocity between the upper and lower bound limits
    for particle in range(swarmSize):
        for theDimension in range(numDimensions):

            # If the new velocity of particle i is > the limit, then reduce to the limit
            if newVelocity[particle][theDimension] > upperBound:
                newVelocity[particle][theDimension] = upperBound

            # If the new velocity of particle i is < the limit, then increase to the limit
            if newVelocity[particle][theDimension] < lowerBound:
                newVelocity[particle][theDimension] = lowerBound

    # Position ---------------------------------------------------------------

    ## Update new position based on the updated velocity
    newPosition = position[:] + newVelocity[:]

    ## Make sure that the position is within the bounds -----------------------
    for particle in range(swarmSize):
        for theDimension in range(numDimensions):

            # Check to see if position in bounds
            while (newPosition[particle][theDimension] > upperBound) or (newPosition[particle][theDimens
```

```
                # Stocastically put the position in bounds
                newPosition[particle][theDimension] = randNumGenerator.uniform(lowerBound, upperBound)

    # Convert position and velocity back to list ------------------------------
    newPosition = newPosition.tolist()
    newVelocity = newVelocity.tolist()

    return [newPosition, newVelocity]
```

1.5 Function to Calculate Particle Historical Best Fitness Value and Position

```
# ================================================================================
# Compare current position fitness value to the current best (for each particle)
# ================================================================================
def calculateParticleBests(position, swarmSize, numDimensions,
                           pCurrFitValue, pBestPosition, pBestFitValue):
    # Calculate the fitness of the new positions
    for particle in range(swarmSize):
        for theDimension in range(numDimensions):

            # Get the current fitness value of the new positions
            pCurrFitValue[particle] = evalFitnessVal(position[:][particle])

            # Compare the current positions' value to their person best
            if pCurrFitValue[particle] < pBestFitValue[particle]:

                # If better, then set the best VALUE to the current value (as a copy [:])
                pBestFitValue[particle] = pCurrFitValue[:][particle]

                # If better, then set the best POSITION to the current position  (as a copy [:])
                pBestPosition[particle] = position[:][particle]

    return [pCurrFitValue, pBestPosition, pBestFitValue]
```

1.6 2 Functions to Display Best Solution and Write to CSV

- `writeIteratonsToCSV()` is used to write a CSV with the top `n` iterations
- Using R, I plotted the first 5 iterations of `swarmSize` 5 (see related question below)

```
# ================================================================================
# DISPLAY GLOBAL BEST AND DIMENSIONS FUNCTION
# Function for displaying the global best and its dimensions
# ================================================================================
def displayGlobalBest(glBestFitValue, glBestPosition, numDimensions, printDims):
    # Print the global optima
    print('\nGlobal Best Value:\t % 0.4f' % glBestFitValue)


    # Print each dimension (if toggled)
    if printDims:
        print('For each [dimension], Global Best Position:')

        # Print the position of each dimension in markdown table format
        print('\n',
              '| Dimension | Global Best |', sep ='')
        print('|-----------|-------------|')
        for theDimension in range(numDimensions):
            print('|' + str(theDimension).rjust(10, ' '),
                  '|' + '{:.4f}'.format(glBestPosition[theDimension]).rjust(12, ' ') + ' |'
                  )


# ================================================================================
# WRITE TOP n SWARM ITERATIONS TO A CSV FUNCTION
# Basic function for writing to file
# ================================================================================
def writeIteratonsToCSV(numDimensions, # Number of dimensions in the swarm
                        filename,
                        method,
                        velocityIterations,
                        positionIterations,
                        gBestPositionIterations,
                        swarmSize):

    iterBreak     = 1 # Display every n iterations
    maxIterToView = 5 # Top iteration to display

    # Write to file if the dimensions are 2D and using Global best.
    # (only global since list structure for each particle, not single value)
    if numDimensions == 2 and method == 'global':

        # Print the first 5 best positions of the swarm, while highlighting global best
        f = open(filename + '.csv', 'w')  # Open CSV for writing

        # Column names
        f.write('iteration,particle,position1,position2,globalBest1,globalBest2,velocity1,velocity2\n')

        # Write the data to a CSV for plotting and summary table
```

```
        for iteration in range(maxIterToView):
            thisIter = ''
            for particle in range(swarmSize):

                if iteration % iterBreak == 0:

                    # Get the Iteration, Particle, position1/2 and velocity 1/2
                    theVelocities = velocityIterations[iteration][particle]
                    thePositions  = positionIterations[iteration][particle]
                    globalBest    = gBestPositionIterations[iteration]

                    # Convert the positions to flat text
                    velDims    = ','.join(['{:.4f}'.format(positionVal) for positionVal in theVelocitie
                    posDims    = ','.join(['{:.4f}'.format(positionVal) for positionVal in thePositions
                    globalDims = ','.join(['{:.4f}'.format(positionVal) for positionVal in globalBest])

                    # Write tp the file
                    f.write(str(iteration+1) + ',' + str(particle) + ',' + str(posDims) + ',' +
                            str(globalDims) + ',' + str(velDims) + '\n')

        f.close()  # close the file for saving output
```

1.7 Driver Funtion to Compile and Loop through a PSO Problem

```
# ================================================================================
# SWARM OPTIMIZATION FUNCTION
# Parameters:
# ---- numDimensions:   The number of dimension in the schwefel function
# ---- swarmSize:       The number of particles within the swarm
# ---- intertiaWeight:  The weight assigned to the intertia component of the velocity eq.
# ---- phi1:            Cognitive weight of the velocity equation. Note phi1 + phi2 <= 4
# ---- phi2:            Social    weight of the velocity equation. Note phi1 + phi2 <= 4
# ---- totalIterations: The total number of iterations before stopping (Stopping criterion)
# ---- method:          Can use 'local' or 'global' best methods.
# ----                  'local' uses Ring Structure with 2 neighbors by default
# ---- filename:        Name of CSV file to export to working directory.
# ----                  If using 2D and 'global' best method, will export CSV.
# ----                  Reason for exporting is to read in data to R for plotting iterations
# ---- printDims:       Print the dimensions' value or not
# ================================================================================

def swarmOptimizationSchwefel( # -------------------- Defaults --------------------
    numDimensions   = 2,       # number of dimensions of problem
    swarmSize       = 5,       # number of particles in swar
    phi1            = 2,       # Cognitive weight
    phi2            = 2,       # Social weight
    intertiaWeight  = 0.1,     # Constant Inertia weighting value
    totalIterations = 100,     # Stopping criteria = the total number of iterations
    method          = 'local', # 'local' or 'global' best function name
    filename        = 'output',# Name of the ouput csv file to write first 5 iterations,
    printDims       = False     # Print the dimension's value or not
    ):

    # Initialize to global best function by default
    functionToGetBest = getGlobalBest

    # If not using the global best, then switch to the local best method
    if method != 'global':
        functionToGetBest = getLocalBest


    # --------------------------------------------------------------------------
    # INITIALIZE POSITION AND VELOCITY, and INITIAL BESTS
    # the swarm will be represented as a list of positions, velocities, values,
    # pBestPosition, and pBestPosition values
    # Note: position[0] and velocity[0] provides the position and velocity of particle 0;
    # position[1] and velocity[1] provides the position and velocity of particle 1; and so on.
    # --------------------------------------------------------------------------


    # Step 1: Initialize swarm and get the particles' and global best (and current position)
    position, velocity, pCurrFitValue, pBestPosition, pBestFitValue, glBestFitValue, glBestPosition = in

    # Create empty lists for holding the swarm iterations
    positionIterations      = [] # Each particle's velocity
```

```
velocityIterations      = [] # Each particle's position
gBestPositionIterations = [] # The current Global Best Position


# ------------------------------------------------------------------------------
# Main Loop
# ------------------------------------------------------------------------------
for iteration in range(totalIterations):

    # Step 0: Keep track of each iterations/dimension for velocity, position, and current global be
    velocityIterations.append(velocity)
    positionIterations.append(position)
    gBestPositionIterations.append(glBestPosition)

    # Step 2: Update the velocity and position
    velocity, position = updateVelocityAndPosition(intertiaWeight, velocity, position,
                                                   phi1, phi2, pBestPosition, glBestPosition,
                                                   swarmSize, numDimensions)

    # Step 3: Recalculate the particle and global bests
    pCurrFitValue, pBestPosition, pBestFitValue = calculateParticleBests(position,
                                                                         swarmSize, numDimensions,
                                                                         pCurrFitValue,
                                                                         pBestPosition,
                                                                         pBestFitValue)

    # Step 4: Get the Global or local best (depends on chosen method) fitness value and position
    glBestFitValue, glBestPosition = functionToGetBest(pBestFitValue[:], pBestPosition[:], swarmSiz


# ------------------------------------------------------------------------------
# Global Best
# ------------------------------------------------------------------------------

# Finally, if using the local best method, get the absolute best from the local bests
if method == 'local':
    gBestFitValue, gBestPosition = getGlobalBest(glBestFitValue, glBestPosition, swarmSize)

else: # if not local best, then change the gl best is the global best
    gBestFitValue, gBestPosition = glBestFitValue, glBestPosition


# ------------------------------------------------------------------------------
# Print and Export
# ------------------------------------------------------------------------------

# Print the global (or local best) and each dimensions' position
displayGlobalBest(gBestFitValue, gBestPosition, numDimensions, printDims)

# If 2D, then write to a csv for plotting in R
writeIteratonsToCSV(numDimensions, filename, method, velocityIterations,
                    positionIterations, gBestPositionIterations, swarmSize)
```

```
    return gBestFitValue # return the best fit
```

1.8 Part 1 (b) & (c) - Global Best Method

1.8.1 1(b) 2D Global Swarm

Create a swarm of size 5 and solve the 2D Schwefel problem

```
# 1(b) Create a swarm of size 5 and solve the 2D Schwefel problem. ------------
swarmOptimizationSchwefel(
    numDimensions   = 2,        # number of dimensions of problem
    swarmSize       = 5,        # number of particles in swar
    phi1            = 2,        # Cognitive weight
    phi2            = 2,        # Social weight
    intertiaWeight  = 0.1,      # Constant Inertia weighting value
    totalIterations = 100000,   # Stopping criteria = the total number of iterations
    method          = 'global', # 'local' or 'global' best function name
    filename        = 'output', # Name of the ouput csv file to write first 5 iterations
    printDims       = True      # Print the dimension's value or not
    )
```

```
##
## Global Best Value:     0.0005
## For each [dimension], Global Best Position:
##
## | Dimension | Global Best |
## |-----------|-------------|
## |         0 |    420.9120 |
## |         1 |    420.9941 |
## 0.0005125608778371316
```

1.8.2 1(b) i. - iii. Iteration Table & Plot

i. - ii. Record in table the first 5 positions and velocities of each particle. Highlight Global Best

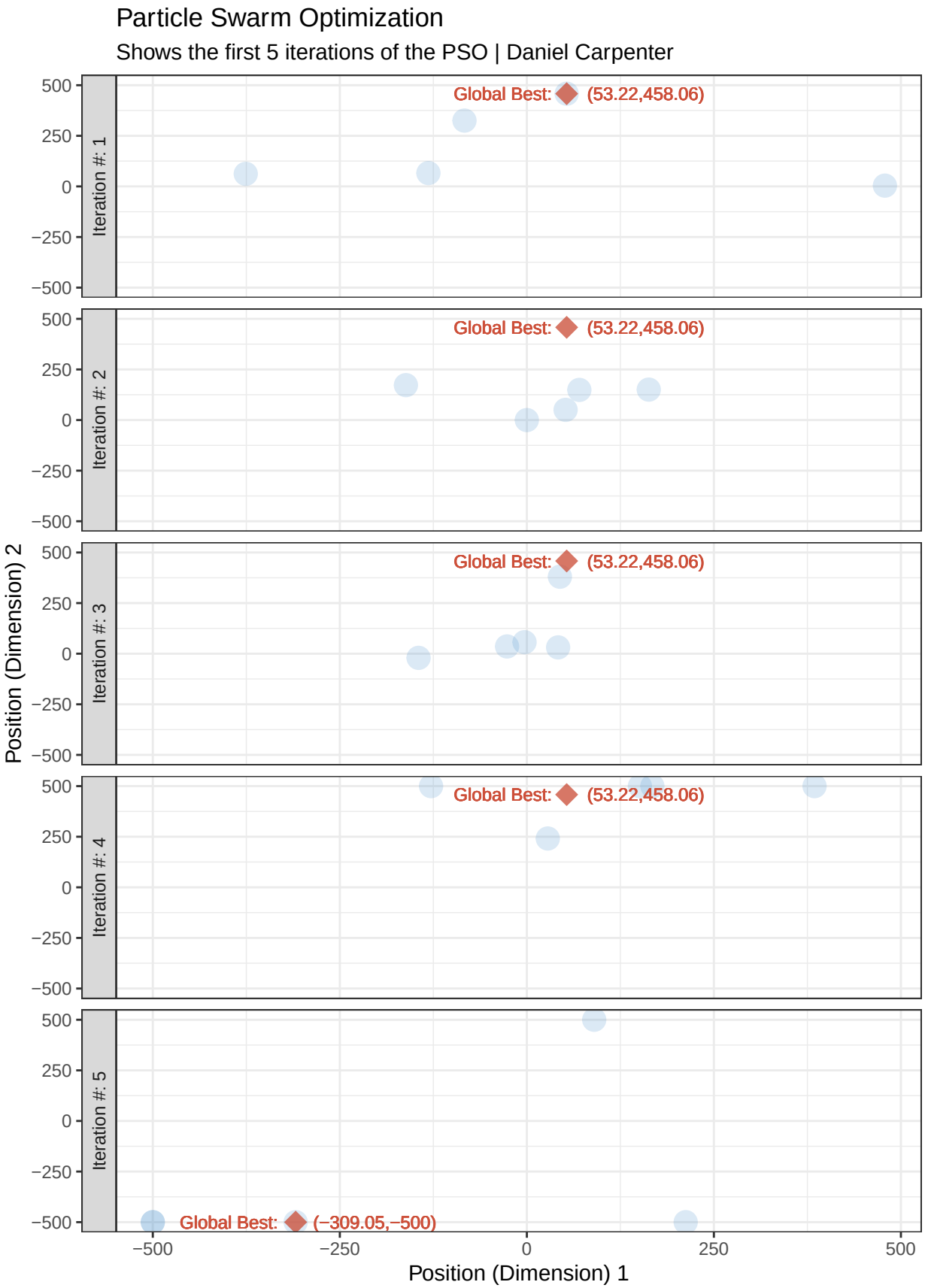
```
library(tidyverse)
library(ggplot2)
library(readr)  # Read CSV
library(knitr)  # table printing


# Pull in the data from the swarm optimization (written in
# main loop)
swarmData <- read_csv("output.csv")


# Table holding the 5 iterations for the 5 particles
# Position and Velocity 1 & 2, as well indicator of global
# best
swarmDataTable <- swarmData %>%
    mutate(isGlobalBest = if_else(position1 == globalBest1 &
        position2 == globalBest2, "New Global Best!", "")) %>%

select(particle, iteration, starts_with("pos"), starts_with("vel"),
    isGlobalBest)

# Print the table
kable(swarmDataTable)
```

particle	iteration	position1	position2	velocity1	velocity2	isGlobalBest
0	1	-83.3801	325.2065	-0.9797	-0.4027	
1	1	-131.5883	66.0082	-0.6127	-0.6766	
2	1	-375.7331	62.0785	-0.1341	-0.6513	
3	1	53.2211	458.0648	-0.2902	-0.8174	New Global Best!
4	1	478.6400	3.9354	-0.1758	-0.7037	
0	2	51.8027	50.4383	-31.5775	375.6448	
1	2	70.1557	148.8914	-61.4326	214.8996	
2	2	162.9646	150.3870	-212.7685	212.4655	
3	2	-0.0290	-0.0817	53.1921	457.9830	
4	2	-161.6524	172.4728	316.9876	176.4082	
0	3	-3.0910	56.7402	48.7117	107.1785	
1	3	-144.7555	-20.5850	-74.5997	128.3064	
2	3	-26.4395	35.7205	136.5252	186.1075	
3	3	44.2005	380.3201	44.1715	380.2383	
4	3	41.8069	31.0758	-119.8455	203.5486	
0	4	151.1079	500.0000	148.0169	478.7985	
1	4	384.5587	500.0000	239.8032	479.4150	
2	4	167.7929	500.0000	141.3534	244.5300	
3	4	27.9970	241.2485	72.1975	-496.5454	
4	4	-128.0547	500.0000	-86.2478	440.2385	
0	5	-309.0507	-500.0000	-157.9428	0.0000	New Global Best!
1	5	-500.0000	-500.0000	-115.4413	0.0000	
2	5	-500.0000	-500.0000	-332.2071	0.0000	

particle	iteration	position1	position2	velocity1	velocity2	isGlobalBest
3	5	90.0355	500.0000	118.0325	-321.1262	
4	5	212.3308	-500.0000	84.2761	0.0000	

iii Plot the first 5 positions of each of the 5 particles

```
require(ggplot2)

# Plot the first 5 iterations with each particle's position
iterPlot <- swarmData %>%
  ggplot() +

  # Points for the Swarm
  geom_point(aes(x = swarmData$position1,
             y = swarmData$position2),
             alpha = 0.2, color = 'steelblue3', size  = 5) +

  # # Red Point for the global best
  geom_point(aes(x = swarmData$globalBest1,
                 y = swarmData$globalBest2),
             alpha = 0.25,
             color = 'tomato3', size  = 5, shape = 18) +

  # # Label for the global best
  geom_text(label = paste0('     Global Best:       (',
                           round(swarmData$globalBest1, 2), ',',
                           round(swarmData$globalBest2, 2),')'),
            aes(x = swarmData$globalBest1,
                y = swarmData$globalBest2),
            color = 'tomato3',
            # alpha = 0.9,
            size = 3) +

  # Facet by Iteration
  facet_grid(rows = vars(paste0('Iteration #: ', iteration)),
             switch = 'y') +
  labs(title = 'Particle Swarm Optimization',
       subtitle = paste0('Shows the first ', max(swarmData$iteration),
                         ' iterations of the PSO | Daniel Carpenter'),
       x = 'Position (Dimension) 1',
       y = 'Position (Dimension) 2') +

  theme_bw() # a theme

print(iterPlot)
```


Particle Swarm Optimization
Shows the first 5 iterations of the PSO | Daniel Carpenter
Global Best: (53.22,458.06)
Global Best: (53.22,458.06)
Global Best: (53.22,458.06)
Global Best: (53.22,458.06)
Global Best: (−309.05,−500)
Iteration #: 1
Iteration #: 2
Iteration #: 3
Iteration #: 4
Iteration #: 5
500
250
0
−250
−500
Position (Dimension) 2
Position (Dimension) 1

1.8.3 1(c) 200D Global Swarm w/Various Param

Create a swarm to best and solve the 200D Schwefel problem

- Please note I did 50 iterations as stopping criteria simply for the sake a running to completion
- My computer cannot handle such large loads. It is likely that higher number of iterations would lead to closer to the known optima
- You can see that the 2D performs very well, as it does with increased dimensions (just not 200).

```
# 1(c) create a swarm to best and solve the 200D Schwefel problem. ------------
# Try different swarm sizes, inertial weights, and values for phi1 and phi2

# 1(c, i) Base 200D
s1 = swarmOptimizationSchwefel(
      numDimensions   = 200,      # number of dimensions of problem
      swarmSize       = 5,        # number of particles in swar
      phi1            = 2,        # Cognitive weight
      phi2            = 2,        # Social weight
      intertiaWeight  = 0.1,      # Constant Inertia weighting value
      totalIterations = 50,       # Stopping criteria = the total number of iterations
      method          = 'global', # 'local' or 'global' best function name
      printDims       = False     # Print the dimension's value or not
      )

# 1(c, ii.) 200D, changes: Swarm from 5 to 10
```

```
##
## Global Best Value:     76347.0722
```

```
s2 = swarmOptimizationSchwefel(
    numDimensions   = 200,      # number of dimensions of problem
    swarmSize       = 10,       # number of particles in swar
    phi1            = 2,        # Cognitive weight
    phi2            = 2,        # Social weight
    intertiaWeight  = 0.1,      # Constant Inertia weighting value
    totalIterations = 50,       # Stopping criteria = the total number of iterations
    method          = 'global', # 'local' or 'global' best function name
    printDims       = False     # Print the dimension's value or not
    )

# 1(c, iii.) 200D, changes: phi1: 1, phi2: 3
```

```
##
## Global Best Value:     75781.8518
```

```
s3 = swarmOptimizationSchwefel(
    numDimensions   = 200,      # number of dimensions of problem
    swarmSize       = 10,       # number of particles in swar
    phi1            = 1,        # Cognitive weight
    phi2            = 3,        # Social weight
    intertiaWeight  = 0.1,      # Constant Inertia weighting value
    totalIterations = 50,       # Stopping criteria = the total number of iterations
```

```
    method          = 'global', # 'local' or 'global' best function name
    printDims       = False     # Print the dimension's value or not
    )

# 1(c, iv.) 200D, changes: interia weight =  0.9
```

```
##
## Global Best Value:     75998.1002
```

```
s4 = swarmOptimizationSchwefel(
    numDimensions   = 200,      # number of dimensions of problem
    swarmSize       = 10,       # number of particles in swar
    phi1            = 1,        # Cognitive weight
    phi2            = 3,        # Social weight
    intertiaWeight  = 0.9,      # Constant Inertia weighting value
    totalIterations = 50,       # Stopping criteria = the total number of iterations
    method          = 'global', # 'local' or 'global' best function name
    printDims       = False     # Print the dimension's value or not
    )

# 1(c, v.) 200D, changes: phi1: 3, phi2: 1
```

```
##
## Global Best Value:     77573.9291
```

```
s5 = swarmOptimizationSchwefel(
    numDimensions   = 200,      # number of dimensions of problem
    swarmSize       = 10,       # number of particles in swar
    phi1            = 3,        # Cognitive weight
    phi2            = 1,        # Social weight
    intertiaWeight  = 0.9,      # Constant Inertia weighting value
    totalIterations = 50,       # Stopping criteria = the total number of iterations
    method          = 'global', # 'local' or 'global' best function name
    printDims       = False     # Print the dimension's value or not
    )

# Compile the results
```

```
##
## Global Best Value:     74347.6093
```

```
swarmResults = [s1, s2, s3, s4, s5]

print('\n------- Question 1(c) Results of Swarm Runs -------')
```

```
##
## ------- Question 1(c) Results of Swarm Runs -------
```

```
for result in range(0, len(swarmResults)):
    print('Result', result, ':\t', swarmResults[result])
```

```
## Result 0 :    76347.07221466661
## Result 1 :    75781.85183383343
## Result 2 :    75998.1002495657
## Result 3 :    77573.92907934781
## Result 4 :    74347.60932545959
```

1.9 Part 1 (d) & (e) - Local Best Method

1.9.1 Reference: Local Swarm Calculation and Logic

Uses a ***`Ring Topology`*** *for Local Best Method*

- Note this function already defined but placed here for ease of referencing

```
# ================================================================================
# LOCAL MIN VALUE AND POSITION SEARCH FUNCTION
# Topology: Ring structure with n neighbors  (default 2)
# Returns the 2 element list of lists with the each particle's local best within neighborhood
# ---- [0] min value and
# ---- [1] associate position of
# Can change numParticlesInNbrhood to consider more or less in particle's neighborhood
# ================================================================================
def getLocalBest(fitnessValues, positions, swarmSize,
                 numParticlesInNbrhood = 2):  # Number of particles to compare to for local best

    lBestFitValue = [] # will hold the best VALUE    of the n surrounding particles, for each particle
    lBestPosition = [] # will hold the best POSITION of the n surrounding particles, for each particle


    # For every particle in the swarm, (starting at n less than index 0)
    for particle in range(-numParticlesInNbrhood, swarmSize - numParticlesInNbrhood):

        # Identify the two neighbors fitness value of this particle,
        # which are the two precedng particles
        personalBestNeighbor1 = fitnessValues[particle]
        personalBestNeighbor2 = fitnessValues[particle + 1]

        # Identify the lowest fitness value of this particle's the two preciding neighbors
        minNeighValue = min(personalBestNeighbor1, personalBestNeighbor2)

        # Store the index of the particle
        minNeighIndex = fitnessValues.index(minNeighValue)

        # Store the particle's best neighbors fitness value and position
        lBestFitValue.append(fitnessValues[minNeighIndex])
        lBestPosition.append(positions[minNeighIndex])

    # Returns a list of particles and the min of their n best fit. valued neighbors
    return[lBestFitValue, lBestPosition]
```

1.9.2 1(d) 2D Local Swarm

Implement a PSO algorithm that uses the `local best` *with* `ring` *topology in place of the global best. See aboce logic and explanation*

```
# 1(d) Create a swarm of size 5 and solve the 2D Schwefel problem. ------------
# Implement a PSO algorithm that uses the "local best" in place of the
# global best and explain which topology you are using.

# Topology: Ring structure with 2 neighbors
swarmOptimizationSchwefel(
    numDimensions   = 2,        # number of dimensions of problem
    swarmSize       = 5,        # number of particles in swar
    phi1            = 2,        # Cognitive weight
    phi2            = 2,        # Social weight
    intertiaWeight  = 0.1,      # Constant Inertia weighting value
    totalIterations = 100000,   # Stopping criteria = the total number of iterations
    method          = 'local',  # 'local' or 'global' best function name
    filename        = 'output', # Name of the ouput csv file to write first 5 iterations
    printDims       = True      # Print the dimension's value or not
    )
```

```
##
## Global Best Value:     0.0004
## For each [dimension], Global Best Position:
##
## | Dimension | Global Best |
## |-----------|-------------|
## |         0 |    420.9179 |
## |         1 |    420.9590 |
## 0.0003635313285030861
```

1.9.3 1(e) 200D Local Swarm w/Various Params.

Solve the 200D Schwefel problem as best as possible

```
# 1(e) create a swarm to best and solve the 200D Schwefel problem. ------------
# Same as 1(c) parameters, just using 'local' now
# Topology: Ring structure with 2 neighbors

# 1(e, i) Base 200D
s1e = swarmOptimizationSchwefel(
      numDimensions   = 200,      # number of dimensions of problem
      swarmSize       = 5,        # number of particles in swar
      phi1            = 2,        # Cognitive weight
      phi2            = 2,        # Social weight
      intertiaWeight  = 0.1,      # Constant Inertia weighting value
      totalIterations = 50,       # Stopping criteria = the total number of iterations
      method          = 'local',  # 'local' or 'global' best function name
      printDims       = False     # Print the dimension's value or not
      )

# 1(e, ii.) 200D, changes: Swarm from 5 to 10
```

```
##
## Global Best Value:     76065.0451
```

```
s2e = swarmOptimizationSchwefel(
    numDimensions   = 200,      # number of dimensions of problem
    swarmSize       = 10,       # number of particles in swar
    phi1            = 2,        # Cognitive weight
    phi2            = 2,        # Social weight
    intertiaWeight  = 0.1,      # Constant Inertia weighting value
    totalIterations = 50,       # Stopping criteria = the total number of iterations
    method          = 'local',  # 'local' or 'global' best function name
    printDims       = False     # Print the dimension's value or not
    )

# 1(e, iii.) 200D, changes: phi1: 1, phi2: 3
```

```
##
## Global Best Value:     77721.3454
```

```
s3e = swarmOptimizationSchwefel(
    numDimensions   = 200,      # number of dimensions of problem
    swarmSize       = 10,       # number of particles in swar
    phi1            = 1,        # Cognitive weight
    phi2            = 3,        # Social weight
    intertiaWeight  = 0.1,      # Constant Inertia weighting value
    totalIterations = 50,       # Stopping criteria = the total number of iterations
    method          = 'local',  # 'local' or 'global' best function name
    printDims       = False     # Print the dimension's value or not
    )

# 1(e, iv.) 200D, changes: interia weight =  0.9
```

```
##
## Global Best Value:     75777.2461
```

```
s4e = swarmOptimizationSchwefel(
    numDimensions   = 200,      # number of dimensions of problem
    swarmSize       = 10,       # number of particles in swar
    phi1            = 1,        # Cognitive weight
    phi2            = 3,        # Social weight
    intertiaWeight  = 0.9,      # Constant Inertia weighting value
    totalIterations = 50,       # Stopping criteria = the total number of iterations
    method          = 'local',  # 'local' or 'global' best function name
    printDims       = False     # Print the dimension's value or not
    )

# 1(e, v.) 200D, changes: phi1: 3, phi2: 1
```

```
##
## Global Best Value:     75704.8377
```

```
s5e = swarmOptimizationSchwefel(
    numDimensions   = 200,      # number of dimensions of problem
    swarmSize       = 10,       # number of particles in swar
    phi1            = 3,        # Cognitive weight
    phi2            = 1,        # Social weight
    intertiaWeight  = 0.9,      # Constant Inertia weighting value
    totalIterations = 50,       # Stopping criteria = the total number of iterations
    method          = 'local',  # 'local' or 'global' best function name
    printDims       = False     # Print the dimension's value or not
    )
```

```
##
## Global Best Value:     75700.0155
```

```
swarmResultsE = [s1e, s2e, s3e, s4e, s5e]

print('\n------- Question 1(e) Results of Swarm Runs -------')
```

```
##
## ------- Question 1(e) Results of Swarm Runs -------
```

```
for result in range(0, len(swarmResultsE)):
    print('Result', result, ':\t', swarmResultsE[result])
```

```
## Result 0 :    76065.04505657653
## Result 1 :    77721.34535963654
## Result 2 :    75777.24606450333
## Result 3 :    75704.83774699294
## Result 4 :    75700.0155348131
```