Particle Swarm Optimization Research Toolbox Documentation

Version 20110515i

By George Evers

www.georgeevers.org/pso_research_toolbox.htm

Clickable Contents

Introduction 1
 Statement of Purpose 1
 Documentation Overview 1
  Conceptual Outline 1
  The Documentation As a Help Menu 1
  Formats Available 2
 Toolbox Compatibility 2
 Prerequisite Terminology 2
  Decision Variable 2
  Decision Vector 2
  Control Panel 2
  Switches 3
  Settings 3
  Trial 3
  Column (Statistics Pertaining to One Set of Trials) 3
  Table (One or More Columns) 3
I. System Preparation 4
 Setting Virtual Memory Size 4
  Step 1 4
  Steps 2 & 3 5
 Checking Java Version 6
 Setting MATLAB Preferences 6
  Line Wrapping 6
  Command Windows Scroll Buffer 6
  Right-hand Text Limit 7
 Selecting a Font Suitable for Pasting into Word 7
 Packing Memory 7
II. A Guided Walk Through 7
 Installing the PSO Toolbox (i.e. Unzipping the Files) 7
 Opening Files for Editing 7
 Auto-saving Figures 8
  Figure Switches 8

- Figure Position 8

Understanding Automatic Input Validation 9

Understanding User Input Validation 9

Interpreting Graphs 10

- Figure Numbers 10
- Objective Function 10
- Phase Plots 10
- Position vs. Iteration 11

Analyzing Workspace Variables 11

- fg 11
- thresh_for_succ 11
- num_trials_successful 12
- k 12
- iter_success 12
- x 13
- xhist 13
- v 14
- vhist 14
- p 15
- Evaluating Positions 15
- fp 16
- f 16
- phist 16
- l 18
- fhist, lhist, & lbest_neighb_size 18
- g 20
- ghist 20
- Workspace Organization 21
- Summary 21

Knowing Where Data is Saved 22

- Figures 22
- Workspaces 22

Activating and De-activating Graphs 24

- De-activating Phase Plots, Position vs. Iteration Graphs, and Benchmark Graph 24
- Activating Graphs of Personal Bests & Global Best vs. Iteration 24
- Activating Graphs of Function Values vs. Iteration 24

Interpreting Newly Activated Graphs 24

- Function Value vs. Iteration 24
- Personal Bests & Global Best vs. Iteration 24
- Global Best's Function Value vs. Iteration 25

Switching Between Lbest and Gbest PSO's 25

Switching to Static Inertia Weight 25

Changing the Swarm Size 25

Changing the Number of Iterations 25

Activating Swarm Trajectory Graphs 25
Re-using Figures 25
Marking Personal Bests and the Global Best 25
Changing the Objective Function 26
Analyzing Swarm Trajectory 26
Re-using Figures 26
Activating Regrouping PSO (RegPSO) 26
Setting the Format for Auto-Saved Figures 27
Analyzing Swarm Trajectory of RegPSO 27
Analyzing Workspace Variables for RegPSO 28
Switching Termination Criterion from Iterations to Function Evaluations 28
Changing the Number of Trials 29
Changing the Problem Dimensionality 29
Changing the Swarm Size 29
De-activating Unnecessary Histories 29
Disabling Graphs 29
Analyzing the Statistics of Multiple Trials 29
User Input Validation 29
Each Trial's Data 30
Performance Measures 31
Summary 32
III. More Details on Select Topics 32
Specifying Gbest or Lbest PSO as the Core Algorithm 32
Setting the Inertia Weight 33
Setting the Velocity Clamping Percentage 33
Selecting the Objective Function(s) 35
Setting the Center and Range of the Initialization Space 35
Using RegPSO to Escape from Stagnation 36
Specifying the Maximum Number of Iterations or Function Evaluations 38
Maximum per Grouping 38
Maximum over all RegPSO Groupings 38
Maximum over all MPSO Starts 38
Confining Particles to the Search Space 39
Velocity Reset 39
Position Clamping 39
Generating Statistics Automatically 39
Generating Figures 39
Mean Function Value 40
Swarm Trajectory 40
Saving Data Automatically 40
Examining Saved Data 41
Measures of Efficiency: Iterations, Function Evaluations, and Real Time 42
IV. Guide to Conducting Your Own Research 43
Adding Constraints 43

Hu & Eberhart's Approach 43
An Alternative Idea 44
Enhancing Velocity Updates 44
Enhancing Position Updates 45
Varying the Inertia Weight Non-linearly 45
Control_Panel.m 45
gbest_core_loop.m & lbest_core_loop.m 45
Input_Validation.m 46
Display_Settings.m 46
Title_Graphs.m 46
Adding Your Problem to the Toolbox 46
Writing Your Function 46
Listing Your Function in Objectives.m 47
Other Enhancements 48
Benefiting from Updates 49
V. Contribution Standards 49
Following Naming Conventions When Creating New Variables 49
Minimizing Hard Coding via "Control_Panel.m" 49
Validating Inputs via "Input_Validation.m" 50
Steps for Adding an Automatic Correction to "Input_Validation.m" 51
Displaying the Most Important Settings via "Display_Settings.m" 51
Maintaining Workspace Integrity 52
Steps to Avoid Unintentional Usage of Data 52
Testing New Code Independently of the PSO Toolbox 53
Modifying all Relevant Gbest & Lbest Scripts 53
Commenting After "else" 53
Using Short-Circuit Logical Operators 54
Adding New Algorithms to Graph Titles 54
Avoiding Pre-Allocated Zeros 55
Defining New Switches & Programming Variables 56
Applying Changes Throughout the Particle Swarm Research Toolbox 57
VI. Program Flow 57
Open_Files.m 57
Control_Panel.m 57
Objectives.m 57
Input_Validation.m 58
Display_Settings.m 58
RegPSO_main.m 58
Trial_Initializations.m 58
MPSO.m 58
Reg_Methods_0And1.m 58
Reg_Method_1.m 59
Save_data_per_trial.m 59
Load_trial_data_for_stats.m 59

Standard_Output.m 59
Autosave_Workspace_Per_Column.m 59
Graphs.m 60
VII. Disclaimer 60
VIII. Add-ins 60
References 60
Appendix A: Definitions 61
Of Programming/Workspace Variables 61
center_IS 61
center_IS0 61
dim 61
f 62
fg 62
fg_array 62
fghist 62
fghist_current_trial 62
fghist_mean 62
fghist_over_all_trials 63
fhist 63
fhist_current_trial 63
fp 63
fphist 63
fphist_current_trial 64
g 64
ghist 64
ghist_current_trial 64
Internal_i 64
Internal_j 65
iter_mean_per_trial 65
iter_success 65
iter_success_mean 66
iter_tot_all_trials 66
k 66
l 66
lbest_neighb_size 66
lhist 66
lhist_current_trial 66
max_iter_over_all_groupings 67
max_iter_per_grouping 67
max_FEs_over_all_groupings 67
max_FEs_per_grouping 67
MPSO_FE_counter 67
MPSO_k 68
np 68

num_particles_2_evaluate 68
num_trials_successful 68
OnOff_asymmetric_initialization 68
OnOff_func_evals 68
OnOff_NormR_stag_det 69
OnOff_position_clamping 69
OnOff_RegPSO 69
OnOff_SuccessfulUnsuccessful 69
OnOff_Terminate_Upon_Success 69
OnOff_v_reset 70
OPSO_ghost_FEs_per_grouping 70
OPSO_ghost_FEs_RegPSO 70
p 70
phist 70
phist_current_trial 71
position_matrix 71
r1 71
r2 71
Rand_seq_start_point 71
range_IS 71
range_IS0 72
reg_fact 72
RegPSO_grouping_counter 72
RegPSO_k 72
RegPSO_k_after_each_grouping 73
stag_thresh 73
thresh_for_succ 73
true_global_minimum 73
v 74
vhist 74
vhist_current_trial 74
vmax 74
x 74
xhist 75
xhist_current_trial 75
xmax 75
xmin 75
Of Functions 76
ObjFun_Ackley 76
ObjFun_Griewangk 76
ObjFun_Quadric 76
ObjFun_Quartic_Noisy 77
ObjFun_Rastrigin 77
ObjFun_Rosenbrock 77

ObjFun_Rosenbrock_Engelbrecht_VandenBergh 78
ObjFun_Schaffers_f6 78
ObjFun_Schwefel 79
ObjFun_Spherical 80
ObjFun_Weighted_Sphere 80
Of Files 81
Appendix B: Error Correction 81
java.lang.OutOfMemoryError: Java heap space 81
Error using ==> save: Unable to write file… No such file or directory 81
Appendix C: Efficient MATLAB Shortcuts 81
Editing Code 81
Navigating 82
Managing Files 82
Appendix D: Citation of Toolbox or Documentation 82
Examples 82

Introduction

Statement of Purpose

The Particle Swarm Optimization Research Toolbox is currently designed to handle continuous, single-objective optimization problems. Chapter "IV: Guide to Conducting Your Own Research" clarifies how a motivated researcher could add constraints or make other improvements. I will be happy to guide motivated researchers in adding new functionality as done at this MathWorks thread.

Documentation Overview

Conceptual Outline

Preparation

This introduction gives a quick overview of the documentation and toolbox, and the first chapter explains how to optimize your system for improved MATLAB performance.

Usage Guide

The second chapter explains how to use the most popular switches and settings of the toolbox, and the third chapter contains more detailed instructions on various topics – such as how to prevent particles from leaving the initialization space and/or how to use RegPSO to escape premature convergence for continued optimization.

Contribution Guide

The fourth chapter outlines how to use the toolbox to solve unique problems or to make novel improvements to the algorithms implementable by the toolbox. The fifth chapter outlines a structured framework for editing the code to ensure that we all move in the same direction. The sixth chapter outlines program flow for contributors, though end users need only concern themselves with the control panel.

Acknowledgements

The seventh chapter is a disclaimer, and the eighth chapter credits contributors and provides links to their webpages.

Appendices

Appendix A defines variables and functions used by the toolbox. Appendix B explains how to prevent certain errors from occurring. Appendix C contains useful MATLAB shortcuts. And Appendix D contains examples of how to cite the toolbox and/or documentation.

The Documentation As a Help Menu

The documentation will hopefully answer most questions about the toolbox and how to use it for novel research. The table of contents is clickable by topic and can be thought of as an expanded help index. Ctrl + F functionality can be used to search the contents for keywords as would be done within MATLAB's help menu. If the desired topic is not addressed or its discussion is found to be insufficient, please email george [at] georgeevers.org for further clarification; however, please take the time to check this documentation for the answers to your questions.

Formats Available

This documentation is distributed with the toolbox in .pdf format since the free Adobe Reader optimizes aesthetics better than word processors, but .doc format is available for those who prefer to highlight while reading. Oracle, the new owner of Sun Microsystems and hence also of the Java for which Sun fought Microsoft to retain ownership, also makes Open Office – a free alternative to Microsoft and its business practices that runs on Windows, Mac, and Linux. While more basic in functionality and not yet integrated with Outlook, it is surprisingly capable and low on bugs for freeware: I highly recommend it to anyone on a budget.

Toolbox Compatibility

The Particle Swarm Optimization Research Toolbox has been verified to be compatible with MATLAB 2007a, 2009a, 2010a and with Windows XP and 7. It is also thought to be compatible with other recent MATLAB versions since all reported error messages have been addressed.

Prerequisite Terminology

Definitions in this chapter are not meant to be comprehensive but to assign basic meaning to terms used herein.

Decision Variable

A decision variable is one of "n" variables to be optimized by the decision maker (i.e. the Particle Swarm Optimization Research Toolbox user). The number of decision variables to be optimized determines the dimensionality of the optimization problem – i.e. the dimensionality of the hypercube to be searched by the swarm in pursuit of either a global or local minimizer.

Decision Vector

A decision vector is the set of decision variables to be optimized by the decision maker. In PSO literature, this is an n-dimensional position vector since each position assumed by a particle represents one candidate solution to the optimization problem.

Control Panel

The Particle Swarm Optimization Research Toolbox evolved over the course of thesis while combating the premature convergence problem of particle swarm optimization (PSO). Rather than hard coding changes to test new ideas, variables were created whose values are set within in the control panel. It is essential that a research toolbox give the user a considerable degree of flexibility by which to test new ideas, which is what the extensive control panel strives to do.

Switches

Rather than maintaining different codes for different variations of the PSO algorithm, *switches* were created in the control panel by which to activate and de-activate various functionalities. Switches are logical variables occupying one byte of memory rather than the usual eight bytes, and they check quickly and efficiently during program execution since no computation is required. For example, "if *OnOff_lbest*" evaluates to "true" when switch *OnOff_lbest* in the control panel has been activated by setting it to "logical(1)," but it evaluates to "false" when the user has deactivated the switch by setting it to "logical(0)"; this particular check is used within the toolbox to determine whether to implement Lbest or Gbest PSO.

Settings

Aside from switches, which are simply active or inactive, the control panel also contains *settings* which store numerical values. Some settings store basic PSO specifications such as swarm size, inertia weight, acceleration coefficients, and velocity clamping percentage. Others control features such as the number of trials to be executed, the numerical output format, and the size of figures generated.

The most important parts of the control panel with which to become familiar initially are:

(i) The switches in subsections "PSO ALGORITHM SELECTION" and "PSO HISTORIES TO BE MAINTAINED" of section "(1) BASIC SWITCHES & PSO ALGORITHM SELECTION,"

(ii) The basic PSO settings of section "(2) PARTICLE SWARM ALGORITHM SETTINGS," and

(iii) The graphing switches in subsection "REGPSO & PSO GRAPHING SWITCHES" of section "(5) REGPSO & PSO GRAPHING SWITCHES AND SETTINGS".

These essentials will be introduced in chapter "II. A Guided Walk Through". The more familiar you become with the toolbox, the more settings you will master.

Trial

Each simulation is referred to as a "trial". When *num_trials* is set to 1 atop section "(2) PARTICLE SWARM ALGORITHM SETTINGS" of the control panel, one trial will be executed according to the settings specified.

Column (Statistics Pertaining to One Set of Trials)

When *num_trials* is set greater than 1, each trial's best function value is automatically analyzed by the Particle Swarm Optimization Research Toolbox to calculate and display the median, mean, minimum, maximum, and standard deviation. Each trial within this set of trials will use precisely the same settings, but each will use a uniquely repeatable sequence of pseudo-random numbers. The more trials one conducts, the less effect any particular trial's randomness will have on the statistics, and the better the indication will be of the algorithm's overall effectiveness.

Table (One or More Columns)

The table feature is only available in the full version of the Particle Swarm Optimization Research Toolbox, so this discussion can be skipped by users of the free demo version.

A table containing multiple columns is generated when a value such as the swarm size, inertia weight, velocity clamping percentage, or regrouping factor (i.e. of RegPSO [1]) is automatically incremented per column. Displayed atop each table are the basic PSO parameters held constant across columns. Displayed atop each column is the value of the parameter being incremented. And displayed within each column are the statistics generated for the parameter combination specific to that set of trials.

Most basic PSO parameters can be incremented across columns within a table, while the objective and acceleration coefficients can be incremented across the tables themselves. This programming direction was chosen to automatically increment the inertia weight for specific acceleration coefficients within a table before automatically incrementing the acceleration coefficients between tables in pursuit of a high-quality, general-purpose PSO parameter combination, such as those presented in Chapter III of thesis [2].

A "table" can generally be thought of as a set of "columns," where each column contains the statistics produced by a particular parameter combination. For example, a table pertaining to acceleration coefficients $c_1 = 2.0, c_2 = 2.0$ might contain the statistics produced by inertia weight $\omega = 0.71$, in which case the next column would contain the statistics produced by $\omega = 0.72$, and so forth. When only the objective function is incremented, each "table" will contain exactly one "column" of statistics since no parameter would be incremented within each table.

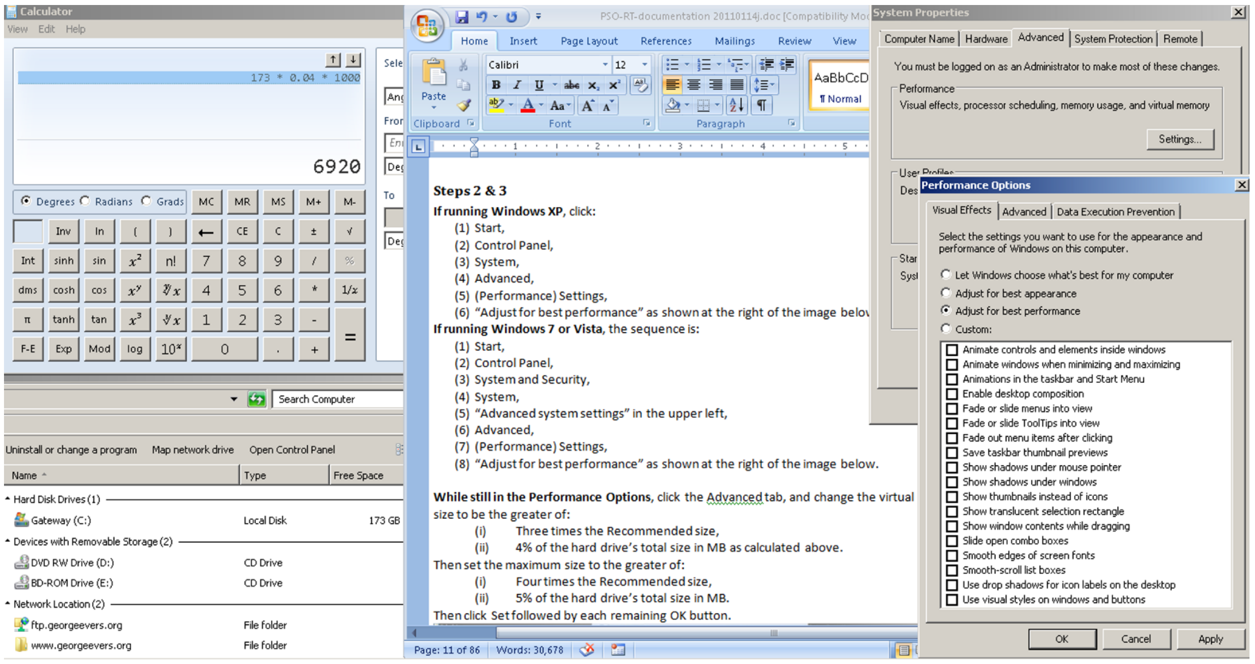
I. System Preparation

Setting Virtual Memory Size

Since MATLAB is more memory-intensive than the typical Windows program, customizing the following Windows options can improve performance.

Step 1

(i) Click Start followed by My Computer,
(ii) Press Alt + {v, d} to view details,
(iii) Leave My Computer open with the Total Size of your hard drive visible as done at the bottom left of the figure below,
(iv) Press Windows + R to access the Run dialog box,
(v) Type "calc" and press Enter,
(vi) Multiply your hard drive's Total Size by 40 (i.e. by 0.04 to find 4%, then by 1,000 to convert from GB to MB) as done at the top left of the figure below.



Name	Type	Free Space
Hard Disk Drives (1)		
Gateway (C:)	Local Disk	173 GB
Devices with Removable Storage (2)		
DVD RW Drive (D:)	CD Drive	
BD-ROM Drive (E:)	CD Drive	
Network Location (2)		
ftp.georgeevers.org	File folder	
www.georgeevers.org	File folder	

Steps 2 & 3
If running Windows XP, click:
(1) Start,
(2) Control Panel,
(3) System,
(4) Advanced,
(5) (Performance) Settings,
(6) "Adjust for best performance" as shown at the right of the image belov
If running Windows 7 or Vista, the sequence is:
(1) Start,
(2) Control Panel,
(3) System and Security,
(4) System,
(5) "Advanced system settings" in the upper left,
(6) Advanced,
(7) (Performance) Settings,
(8) "Adjust for best performance" as shown at the right of the image below.

While still in the Performance Options, click the Advanced tab, and change the virtual size to be the greater of:
(i) Three times the Recommended size,
(ii) 4% of the hard drive's total size in MB as calculated above.
Then set the maximum size to the greater of:
(i) Four times the Recommended size,
(ii) 5% of the hard drive's total size in MB.
Then click Set followed by each remaining OK button.

System Properties
You must be logged on as an Administrator to make most of these changes.
Performance
Visual effects, processor scheduling, memory usage, and virtual memory

Performance Options
Select the settings you want to use for the appearance and performance of Windows on this computer.
- Let Windows choose what's best for my computer
- Adjust for best appearance
- [✓ Adjust for best performance]
- Custom:
- ☐ Animate controls and elements inside windows
- ☐ Animate windows when minimizing and maximizing
- ☐ Animations in the taskbar and Start Menu
- ☐ Enable desktop composition
- ☐ Fade or slide menus into view
- ☐ Fade or slide ToolTips into view
- ☐ Fade out menu items after clicking
- ☐ Save taskbar thumbnail previews
- ☐ Show shadows under mouse pointer
- ☐ Show shadows under windows
- ☐ Show thumbnails instead of icons
- ☐ Show translucent selection rectangle
- ☐ Show window contents while dragging
- ☐ Slide open combo boxes
- ☐ Smooth edges of screen fonts
- ☐ Smooth-scroll list boxes
- ☐ Use drop shadows for icon labels on the desktop
- ☐ Use visual styles on windows and buttons

Do not close My Computer or the calculator until the virtual memory allocation has been customized below.

Steps 2 & 3

If running Windows XP, click:
(1) Start,
(2) Control Panel,
(3) System,
(4) Advanced,
(5) (Performance) Settings,
(6) "Adjust for best performance" as shown at the right of the image above.
(7) Click "OK", but do not close any other window yet.

If running Windows 7 or Vista, the sequence is:
(1) Start,
(2) Control Panel,
(3) System and Security,
(4) System,
(5) "Advanced system settings" in the upper left,
(6) Advanced,
(7) (Performance) Settings,
(8) "Adjust for best performance" as shown at the right of the image above.
(8) Click "OK", but do not close any other window yet.

While still in the Performance Options, click the Advanced tab, which is highlighted at the right of the figure below, and change the virtual memory size to be the greater of:
(i) Three times the size Recommended near the bottom of the window,

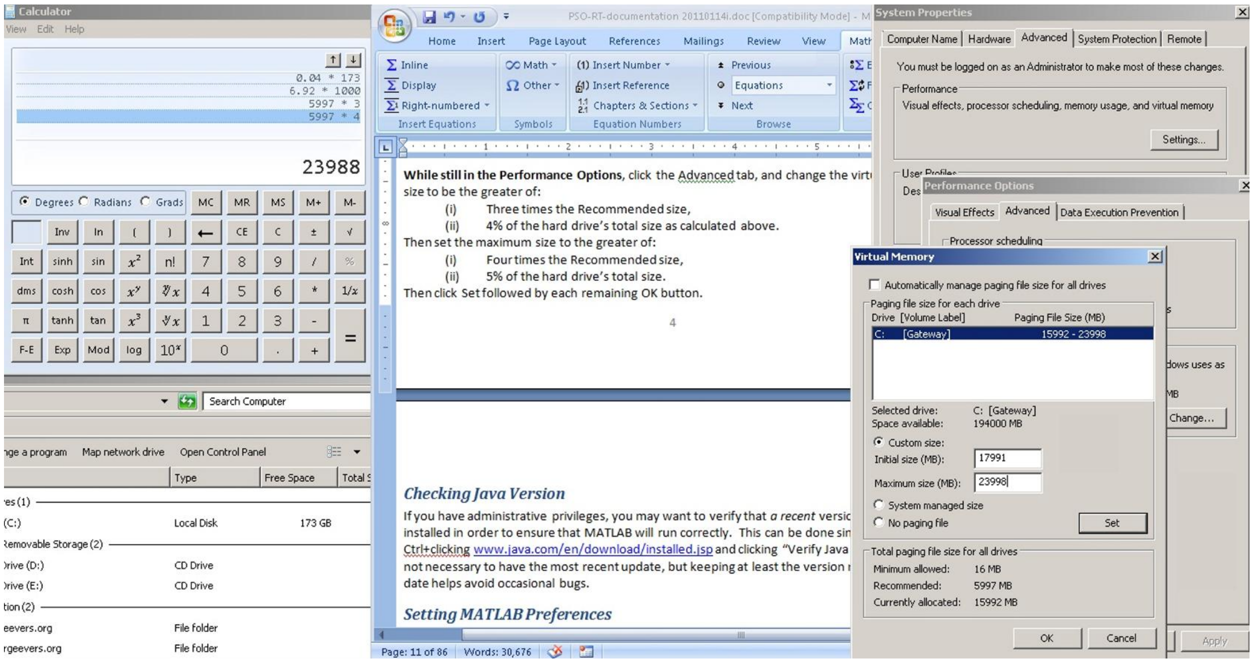
(ii) 4% of the hard drive's total size in MB.

Then set the maximum size to the greater of:

(i) Four times the Recommended size,

(ii) 5% of the hard drive's total size in MB.

Then click Set followed by each remaining OK button.



```
0.04 * 173
6.92 * 1000
5997 * 3
5997 * 4
23988
```

Virtual Memory

[ ] Automatically manage paging file size for all drives

Paging file size for each drive

Drive [Volume Label]	Paging File Size (MB)
C: [Gateway]	15992 - 23998

Selected drive: C: [Gateway]
Space available: 194000 MB

[✓ Custom size:]
Initial size (MB): 17991
Maximum size (MB): 23998

System managed size
No paging file

Total paging file size for all drives
Minimum allowed: 16 MB
Recommended: 5997 MB
Currently allocated: 15992 MB

Checking Java Version

If you have administrative privileges, you may want to verify that *a recent* version of Java is installed to ensure that MATLAB will run correctly. This can be done simply by Ctrl+clicking www.java.com/en/download/installed.jsp and clicking "Verify Java version". It is not necessary to have the most recent update, but keeping at least the version number up-to-date helps avoid occasional bugs.

Setting MATLAB Preferences

Line Wrapping

Messages generated by "Input_validation.m" expect the user to **have "Wrap lines" activated** to automatically fit messages to the user's Command Window. This can be **activated by clicking "File," "Preferences," and "Command Window"** (under the "Font" sub-heading) **then ticking "Wrap lines".**

Command Windows Scroll Buffer

You might also want to change the "Number of lines in command window scroll buffer" to 25,000 to ensure that outputs do not scroll off the screen if tables of statistics are generated.

Right-hand Text Limit

A right-hand text limit can be activated by clicking “**Preferences**” in the File menu, expanding the “**Editor/Debugger**” submenu, clicking “**Display**,” ticking the box to “**Show line**,” and choosing a “Placement” of **75**. Comments typed beyond this column in the Editor will automatically wrap to the next line, which keeps comments narrow enough for pasting into a word processor. The dashed line at column 75 is also a visual reminder to break long lines of code onto separate lines using an ellipsis, which keeps the code itself narrow enough for pasting.

Selecting a Font Suitable for Pasting into Word

When pasting from MATLAB’s Command Window into Word, font **Courier New preserves spacing**, and **MS Mincho does the same** with a narrower font. Both of these are available as standard MATLAB outputs **under the “Fonts” heading** of the “File” menu’s “Preferences”. Setting either of these as the MATLAB font simplifies the process of pasting into Word.

Packing Memory

Typing “pack” at MATLAB’s command prompt defragments the memory assigned to MATLAB and improves response time. Type it whenever MATLAB has been in use for days or many hours and seems to be lagging.

II. A Guided Walk Through

Installing the PSO Toolbox (i.e. Unzipping the Files)

After downloading the most recent version of the Particle Swarm Optimization Research Toolbox from www.georgeevers.org/pso_research_toolbox.htm, right-click the zipped file, and extract all contents to “C:\”. Unzipping to this location instead of to My Documents ensures that workspaces named transparently with the values of their contained variables will not become problematic when combined with the length of the file path. (If you cannot unzip the file, you might first need to install a program such as the free 7-Zip.)

Opening Files for Editing

Browse to the folder that you just unzipped, right click “Open_Files.m,” and left click “Run”. This opens the files most likely to be edited and selects “Control_Panel.m” as the current tab, within which you will specify precisely what simulations to run. **Settings not self-explanatory by name are described by accompanying comments**. In the future, you can comment out irrelevant lines within “Open_Files.m” to focus on particular files to be modified.

The most essential settings are discussed below to introduce you to the Particle Swarm Optimization Research Toolbox.

Auto-saving Figures

Figure Switches

Saving figures to MATLAB's *.fig format is recommended since this facilitates future modifications to titles, legends, font sizes, etc. before publication; whereas, figures saved to other formats are more difficult to modify. When switch *OnOff_autosave_figs* is active in subsection "REGPSO & PSO GRAPHING SETTINGS" of the control panel's section "(5) REGPSO & PSO GRAPHING SWITCHES AND SETTINGS," the Particle Swarm Optimization Research Toolbox automatically saves all figures as *.fig files. This switch is active by default.

Also active by default is *OnOff_autosave_figures_to_another_format*, which auto-saves figures to the additional format specified in setting *GraphParams_autosave_format.*

Figure Position

When saving to a format other than .fig, images of the highest quality are obtained by maximizing figures before saving them. The *Figure_Position* setting was created for this purpose, though it could instead be used to specify any non-maximized location on the screen for figure generation.

For now, let's set figures to full-screen viewing in the control panel according to the following steps pasted from subsection "REGPSO & PSO GRAPHING SETTINGS" of section "(5) REGPSO & PSO GRAPHING SWITCHES AND SETTINGS". You can jump directly to the setting by pressing Ctrl + F within the control panel and typing or pasting "*Figure_Position*".

```
    Figure_Position = [1 76 1920 1058];
        %To set the ideal figure position for automatically
                %displaying figures to your screen:
        %(1) Type "figure(1)" at MATLAB's command prompt.
        %(2) Resize the figure to assume the size and location
                %on the screen desired for automatically generated
                %figures: for best display, this is the maximized size.
        %(3) At the command prompt, type or paste "get(figure(1), 'Position')"
                %If you maximized the figure in step 2, this method accounts
                %for the space occupied by all Taskbars such as those used by
                %Windows and Office (i.e. same as "get(0,'ScreenSize')"
                 %except space is reserved for all menu bars for optimal display).
        %(4) The result is the 1X4 vector to be used as
                 %"Figure_Position" above.
```

Once you have completed the four steps above, let's run a basis simulation by which to understand the effects of changes to be made in the walk through. With "Control_Panel.m" open and active, please press F5 to begin an example simulation; this is the quickest way to begin program execution.

Understanding Automatic Input Validation

The Particle Swarm Optimization Research Toolbox validates input consistency automatically; when the user's intention is inferable, it corrects conflicting switches and settings. The first paragraph displayed to the "Automatic Input Validation" section of the Command Window warns that switch *OnOff_graph_fg_mean* has been activated in the control panel, which indicates that the mean function value should be graphed versus iteration number; however, we contradictorily specified that only one trial should be conducted so that there is no mean function value across trials. The toolbox cannot definitively infer whether to conduct multiple trials and graph the mean function value or conduct one trial and graph its function value. Consequently, the toolbox: (i) notifies us of the discrepancy, (ii) de-activates *OnOff_graph_fg_mean* automatically to allow program execution according to the current settings, (iii) notifies us of the action taken, and (iv) recommends an alternate solution in case the automatic correction was not the desired reparation.

In this example, since conflicting settings were not severe enough to prevent data generation, a message warned of the potential problem but allowed program execution to continue. When a conflict is severe enough to interfere with intended program execution and the user's intention cannot reliably be inferred to fix the problem automatically, a beep will be generated along with an error message guiding you to correct the conflicting switches or settings.

The second paragraph displayed to the "Automatic Input Validation" section of the Command Window clarifies that since phase plots and position versus iteration graphs are active, the history of positions has automatically been activated from which to construct those graphs. Automatic changes such as these make the toolbox easier to use by intelligently fixing problems prior to program execution.

No changes need to be made at this point; these messages were generated as examples.

Understanding User Input Validation

Once "Input_Validation.m" has made obvious corrections and mentioned any problems thought to remain, "Display_Settings.m" displays settings for user validation. It can be seen from the "User Input Validation" section that:

- Lbest PSO will execute 35 iterations.
- No mechanism such as position clamping or velocity reset will be employed by which to restrain particles to remain within the initialization space (i.e. they are free to search for better solutions outside the space in which they are initialized, which might be useful for some application problems where the initialization space is nothing more than an educated guess as to where the global minimizer might lie rather than being based on physical limitations or a priori information).
- Velocities will be clamped to 50% of the range of the search space, which sets vmax = xmax since the initialization space is centered at the Euclidean origin [i.e. vmax = 0.5*range = 0.5*(*xmax* – -*xmax*) = 0.5*(2**xmax*) = *xmax*].
- All histories to be maintained are displayed for verification.

- One trial will be conducted.
- The swarm will consist of 5 particles.
- The inertia weight will vary linearly from 0.9 in the first velocity update to 0.4 in the final velocity update.
- The cognitive and social acceleration coefficients are each 1.49618.
- The two-dimensional Ackley benchmark has been selected to allow the user to verify that the graph produced by the Particle Swarm Optimization Research Toolbox is correct since the "ackleyfcn" function of MATLAB contains a couple of bugs.
- Symmetric rather than asymmetric initialization will be employed.
- If a trial produces a function value less than or equal to the threshold required for success, 5×10^{-5}, it will be considered successful.
- The search will not terminate upon satisfying the threshold required for success but will continue for the full 35 iterations.

Please type "y" to confirm that the displayed settings are correct, and let's look at some of the graphs that can be generated by the Particle Swarm Optimization Research Toolbox. Please be patient while the graphs are generated.

Interpreting Graphs

Graphs are highly important since one can more easily understand what can be visualized. MATLAB's ability to effortlessly generate powerful graphs gives it a tremendous advantage over lower-level programming languages such as C++.

Figure Numbers

To prevent the possibility of overlapping graphs should all graph types be activated simultaneously, the Particle Swarm Optimization Research Toolbox reserves a range of figure numbers for each graph type. Clicking on the MATLAB icon shows the different ranges of figure numbers used for the selected graph types.

Objective Function

Figure 10,000,000 shows the function value versus the two dimensions being optimized for the Ackley objective function. MATLAB does an excellent job of generating 3-D figures such as this.

Objective functions can only be graphed in 3-D (i.e. function value vs. two dimensions to be optimized) due both to geometric constraints and the difficulty of visualizing higher dimensions even within the human mind. Since objective behavior can change in higher dimensions, beware that graphs in low-dimensionality may not be representative of objective behavior in higher dimensions unless one is attentive to the formulas themselves and what can be inferred from them.

Phase Plots

Each of Figures 1,001 – 1,005 shows a phase plot for one particle. These overlay a 2-D contour map of the objective function to show how particles' movements are affected by the terrain of the search space. The first movement takes each particle from its position at *k* = 0 to its position at *k* = 1, where programming variable *k* counts the updates.

Phase plots shed light on how particles behave in 2-D space, which may be useful for improving performance in more problematic, higher-dimensional spaces. The next graph type becomes useful in higher dimensions.

Position vs. Iteration

Each of Figures 1 – 5 shows each dimension of each particle's position vector versus iteration number. Position vs. iteration graphs are similar to phase plots in that they show a particle's dimensions changing with time. Both graph types show how each particle moved iteratively from its starting position to its final position; however, position vs. iteration graphs can be constructed in higher dimensions since a color is assigned to each of *dim* dimensions instead of plotting dimension 2 versus dimension 1.

Theoretically, position versus iteration graphs can handle any number of dimensions; yet practically, graphing more than seven dimensions simultaneously could become confusing due to overlapping lines.

Analyzing Workspace Variables

In this section, you will learn how to analyze data produced by the Particle Swarm Optimization Research Toolbox. After understanding this section, you will be able to analyze the results of the optimization process – such as each iteration's position vectors, personal bests, local bests/global best, and the corresponding function values. You will also be able to analyze the velocity vectors of any iteration. While this data can be analyzed graphically, some users will need to access workspace variables directly to determine precise values or to write their own toolbox add-ins. If you are only concerned with the graphical features of the toolbox, you may skip this section and jump directly to "Knowing Where Data is Stored".

fg

The global best's function value is iteratively updated and stored to workspace variable *fg*. The best function value, "*fg* = 0.10052," is displayed in the "Results" section of the Command Window. The same nomenclature, *fg*, is used regardless of whether Gbest or Lbest PSO is implemented (i.e. even though the global best is not iteratively calculated by Lbest PSO unless switch *OnOff_ghist* has been activated to maintain a history of each iteration's best position). You may type *fg* at the command prompt if you would like to verify that the value is correctly displayed.

thresh_for_succ

Please type or paste *thresh_for_succ* at the command prompt. You should see:

```
thresh_for_succ =
  5.0000e-005
```

This is the value of the threshold for success as set in "Objectives.m" for the objective function and problem dimensionality selected. To be considered successful, a trial must reduce the objective function's value, *fg*, to this threshold value or better.

num_trials_successful

Notice the lines:

```
Number of Successful Trials: 0
Number of Unsuccessful Trials: 1.
```

Since *fg* was not minimized to the threshold for success, the trial was considered unsuccessful. This is no reason for concern, however, since the trial was run only for 35 iterations using a small swarm size for demonstration purposes.

You may type or paste *num_trials_successful* at the command prompt any time to want to determine how many trials were successful, though this will generally be displayed for you.

k

This counts the number of updates. Typing *k* at the command prompt shows:

```
k =
   34
```

Positions and velocities were updated 34 times since 35 "iterations" were specified in the control panel via setting *max_iter_per_grouping* (i.e. initialization of the swarm + 34 rounds of updates = 35 "iterations").

iter_success

Notice at the end of the last line displayed:

```
iter_success = NaN.
```

Workspace variable *iter_success* stores the iteration number at which the objective function was successfully minimized to *thresh_for_succ*. When multiple trials are conducted, *iter_success* is a 1-by-*num_trials* vector storing the iteration of success for each trial.

NaN was stored to *iter_success* because the objective function was not minimized to *thresh_for_succ* at any point during the trial. NaN's denote unsuccessful trials since 0 would literally mean that at least one particle satisfied the threshold for success at initialization so that no updates were required to achieve success – a possible scenario should the user select either a large value for *thresh_for_succ* or a very large swarm size. Accidental averaging of a 0 instead of a NAN when calculating the mean number of iterations required for success would produce an erroneous numerical value; whereas, accidental averaging of a NAN would produce a NAN, which would clearly indicate to the developer a programming mistake. Safeguards such as this improve the reliability of calculations performed by the Particle Swarm Optimization Research Toolbox.

x

Position matrix, *x,* is updated iteratively according to the position update equation of PSO. Each row of *x* stores one particle's position vector: i.e. "x(1, :)" holds the first particle's position vector, "x(2, :)" holds the second particle's position vector, and so forth. Typing "*x*" at the command prompt displays particles' final positions as shown below.

```
x =
 -1.7193e-002  1.9425e+000
 -7.1203e-002  3.0715e-002
  2.5068e-001 -1.4472e-002
 -5.3025e-001 -2.2441e-001
 -3.5571e-002 -9.2143e-001
```

The first particle came to rest at position vector [-0.017193, 1.9425], the second particle came to rest at [-0.071203, 0.030715], and so forth.

xhist

This is the history of all positions occupied during the search. Particles' starting positions are randomly generated during the first iteration when update counter *k* = 0; these starting positions are contained within the first *dim* columns of the history (i.e. in this case columns 1 and 2 since *dim* = 2). Typing "whos xhist" at the command prompt shows that the history's size is 5x70: this is because each of the 5 particles recorded its 2-D position vector at each of the 35 iterations. The last *dim* columns (i.e. columns 69 and 70 in this case) contain the final position matrix at iteration 35, which is equivalent to the final value of *x* pasted above.

The codes in Table 1 can be used to access the position matrix of any iteration number or update number from history *xhist* according to the pattern shown in the final row. If you plan to write any scripts to access data, you will want to familiarize yourself with the patterns in the tables below. At this point, you may want to paste some codes from Table 1 at the command prompt and analyze the position matrices of the corresponding iterations.

Table 1: Extracting the Position Matrix of Any Iteration from "*xhist*"

Update Number ("*k*")	**Iteration Number ("*k* + 1")**	**Code for Extracting Position Matrix from History**
0	1	xhist(:, 1:dim)
1	2	xhist(:, (dim + 1):(2*dim))
2	3	xhist(:, (2*dim + 1):(3*dim))
3	4	xhist(:, (3*dim + 1):(4*dim))
4	5	xhist(:, (4*dim + 1):(5*dim))
k	k + 1	xhist(:, (k*dim + 1):((k + 1)*dim))

Note that *xhist* is only maintained when switch *OnOff_xhist* is activated at the end of the control panel's first section. When a variable's history is not activated, only the final value of the iteratively updated workspace variable will be available for analysis. For long trials, de-activating unnecessary histories improves execution speed and reduces the size of automatically saved workspaces. Simulations can always be repeated to generate any necessary histories if an exact copy of "Control_Panel.m" is saved along with the version of the toolbox used to generate the data.

v

This matrix is updated iteratively to hold the swarm's velocity vectors. At the conclusion of each trial, it contains the swarm's final velocity vectors. To examine particles' final velocities, please type *v* at the command prompt.

```
v =
-1.5168e-002 -9.5737e-001
-2.0359e-002 -3.2566e-003
 2.7746e-001 -6.4505e-002
-1.2088e+000  8.9889e-001
 1.7978e-003 -1.0605e-003
```

The first particle concluded the search with velocity vector [-0.015168, -0.95737], the second with velocity vector [-0.020359, -0.0032566], and so forth as seen above.

vhist

Velocities tend to decrease over the course of the search. The history of velocities, *vhist*, is useful for examining exactly how they change. Particles' starting velocities are randomly generated during the first iteration when update counter *k* = 0; these are contained within the first *dim* columns of the history (i.e. in this case columns 1 and 2 since *dim* = 2). The last *dim* columns (i.e. columns 69 and 70 in this case) contain the final velocity matrix at iteration 35, which is equivalent to the final value of *v* pasted above.

The codes in Table 2 can be used to access the velocity matrix of any iteration number or update number from history *vhist* according to the pattern that becomes obvious in the table. At this point, you may want to paste some codes from the third column of Table 2 at the command prompt and analyze the velocity matrices of the corresponding iterations.

Table 2: Extracting the Velocity of Any Iteration from "*vhist*"

Update Number ("*k*")	**Iteration Number ("*k* + 1")**	**Code for Extracting Velocity Matrix from History**
0	1	vhist(:, 1:dim)
1	2	vhist(:, (dim + 1):(2*dim))
2	3	vhist(:, (2*dim + 1):(3*dim))
3	4	vhist(:, (3*dim + 1):(4*dim))
4	5	vhist(:, (4*dim + 1):(5*dim))
k	k + 1	vhist(:, (k*dim + 1):((k + 1)*dim))

Like other histories, *vhist* is only maintained when its switch, *OnOff_vhist*, is active at the end of the control panel's first section.

p

This matrix is updated iteratively to hold each particle's personal best. To examine the swarm's final personal bests, please type *p* at the command prompt.

```
p =
 1.5066e-002 -1.2579e+000
-2.0065e-002  1.9715e-002
-2.6780e-002  5.0033e-002
-5.3025e-001 -2.2441e-001
-3.5571e-002 -9.2143e-001
```

Each row of *p* stores one particle's personal best: "p(1, :)" = [0.015066, -1.2579] is the personal best of the first particle, "p(2, :)" = [-0.020065, 0.019715] is the personal best of the second particle, and so forth.

Evaluating Positions

To see Ackley's function value for each of these personal bests, please paste

```
ObjFun_Ackley(p, np)
```

at the command prompt. This passes into the Ackley function the matrix of personal bests and requests that the first *np* function values be passed back; since *np* is the number of particles in the swarm as set in the control panel, all 5 positions will be evaluated.

```
ObjFun_Ackley(p, np)
ans =
4.3731e+000
1.0052e-001
2.4433e-001
3.6193e+000
2.6344e+000
```

The second row contains the lowest function value of 1.0052e-001, which corresponds to the second particle's personal best of [-0.020065, 0.019715] shown in *p* above. The best personal best is also the global best as will be verified later.

Notice in the phase plot of Figure 1002 that the second particle, which eventually produced the best function value, landed in the general vicinity of the global minimizer after only two movements. Following other particles' movements along their phase plots reveals that they did not have such a high-quality personal best until sometime after their tenth movement. This gave the second particle an advantageous head start, which helped it outperform the other particles over the mere 35 iterations conducted.

Notice by comparing Figures 1 – 5 that the particles producing the best function values (i.e. particles 2 and 3) had approximated the global minimizer relatively well and reduced their oscillations on both dimensions by iteration 35, while the other three particles were either still oscillating (e.g. Figures 1 & 4) or moving slowly toward the global minimizer (e.g. Figure 5). Hopefully, the graphs generated by the Particle Swarm Optimization Research Toolbox will help us understand and improve the particle swarm algorithm.

fp

The function values of the personal bests are iteratively stored to workspace variable *fp*. At the conclusion of a trial, it stores the function values produced by each particle's personal best. Typing *fp* at the command prompt shows its contents.

```
fp =
4.3731e+000
1.0052e-001
2.4433e-001
3.6193e+000
2.6344e+000
```

Notice that this column vector of function values is identical to that generated by ObjFun_Ackley(*p*, *np*) above. It is good to know how to evaluate a position matrix or vector, but function values can generally be analyzed directly by typing the proper variable name at the command prompt.

f

While *fp* was the column vector storing the function value at each particle's personal best, *f* is the column vector storing the function value of each particle's current or final position. Please type *f* at the command prompt to see the function value at each particle's final resting position.

```
f =
4.8983e+000
3.7257e-001
1.7742e+000
3.6193e+000
2.6344e+000
```

When the search concluded at iteration 35, the fourth and fifth particles had the same function values stored in *f* as in *fp*. This means that they found new personal bests in the final iteration as will be confirmed below.

phist

To examine the history of all personal bests, please type *phist* at the command prompt. Then type *p* and notice that the last two columns of *phist* are identical to *p* since the latter is horizontally concatenated to the former after each iteration.

As mentioned when comparing *f* to *fp*, the personal bests of the final two particles were updated in the final iteration since their new positions were the best they had found over the course of the search. This can now be verified by comparing the matrix of personal bests at iteration 34 with the final matrix of personal bests at iteration 35. To examine the personal bests of iteration 34, please paste "phist(:, (33*dim + 1):(34*dim))" at the command prompt. Then type *p* to compare with the personal bests of final iteration 35.

```
phist(:, (33*dim + 1):(34*dim))
ans =
 1.5066e-002 -1.2579e+000
-2.0065e-002  1.9715e-002
-2.6780e-002  5.0033e-002
 6.7859e-001 -1.1233e+000
-3.7369e-002 -9.2037e-001

p =
 1.5066e-002 -1.2579e+000
-2.0065e-002  1.9715e-002
-2.6780e-002  5.0033e-002
-5.3025e-001 -2.2441e-001
-3.5571e-002 -9.2143e-001
```

The first three particles kept the same personal bests because their final positions did not produce better function values in *f* than did their existing personal bests in *fp*; but the final two particles updated their personal bests since the new function values they produced in *f* were better than those already contained in *fp*. After updating the personal bests, *fp* was updated with the corresponding function values.

Like the other histories analyzed thus far, *phist* has 70 columns: 2 dimensions for each of 35 iterations. After any number of iterations, the length of *phist* will be "(*k* + 1)**dim*" as can be inferred from Table 3 below. This also applies to *xhist*, *vhist*, and *lhist* – all of which iteratively concatenate a matrix of width *dim*.

At this point, you may wish to type or paste from the third column of Table 3 to the command prompt to analyze the matrices of the corresponding iterations.

Table 3: Extracting the Personal Bests of Any Iteration from "*phist*"

Update Number ("*k*")	**Iteration Number ("*k* + 1")**	**Code for Extracting Personal Bests from History**
0	1	phist(:, 1:dim)
1	2	phist(:, (dim + 1):(2*dim))
2	3	phist(:, (2*dim + 1):(3*dim))
3	4	phist(:, (3*dim + 1):(4*dim))
4	5	phist(:, (4*dim + 1):(5*dim))
k	k + 1	phist(:, (k*dim + 1):((k + 1)*dim))

Remember that *phist* is only maintained when switch *OnOff_phist* is activated at the end of the control panel's first section.

l

Local bests are iteratively stored to workspace variable *l*. At the conclusion of a trial, *l* stores the best position found by each particle's neighborhood over the course of the search.

```
l =
-2.0065e-002  1.9715e-002
-2.0065e-002  1.9715e-002
-2.0065e-002  1.9715e-002
-2.6780e-002  5.0033e-002
-3.5571e-002 -9.2143e-001
```

As discussed under subheading "Evaluating Positions," the second particle's personal best outperformed those of the rest of the swarm; consequently, the first and third particles take the second particle's personal best to be their local best since neighborhood size 2 was specified in the control panel's first section via setting *lbest_neighb_size*. Since the third particle had the second-best of all personal bests, its position became the local best of the neighboring fourth particle. The fifth particle took its own personal best as its local best since it produced the third-best of all personal bests, which outperformed the neighboring first and fourth particles, as can be verified by typing *fp*, *p*, and *l* at the command prompt and examining their contents.

Matrix *l* is only created when Lbest PSO is executed (i.e. when switch *OnOff_lbest* is active in the control panel's first section), whereas global best *g* can be maintained in conjunction with Lbest PSO by activating switch *OnOff_ghist* to monitor how the best position changed over the course of the search. Of course, the global best is not accessed by the update equations of Lbest PSO but is made available merely for analysis of how the best position changes in Gbest PSO versus Lbest PSO. You may verify that matrix *g* contains in each row the best of all personal bests by typing *g* at the command prompt.

fhist, lhist, & lbest_neighb_size

To see how the local bests of iteration 1 derived from the positions of the same iteration, please type or paste "fhist(:, 1)" at the command prompt, which shows each particle's function value at iteration 1. Then type or paste "xhist(:, 1:dim)" to see the positions corresponding to the function values. Lastly, paste "lhist(:, 1:dim)" to see the local bests. The results are pasted below for your convenience.

```
fhist(:, 1)
ans =
1.5709e+001
1.8227e+001
2.1294e+001
2.1297e+001
1.5522e+001
```

```
xhist(:, 1:dim)
ans =
 2.9288e+000  8.7536e+000
 1.2911e+001 -3.7448e+000
 6.1658e+000  2.3506e+001
 2.6930e+000  2.7820e+001
-4.5807e+000 -6.9935e+000

lhist(:, 1:dim)
ans =
 -4.5807e+000 -6.9935e+000
  2.9288e+000  8.7536e+000
  1.2911e+001 -3.7448e+000
 -4.5807e+000 -6.9935e+000
 -4.5807e+000 -6.9935e+000
```

Notice that when the swarm was initialized at iteration 1, the best function value of 15.522 belonged to the fifth particle, which had position vector [-4.5807, -6.9935]. Since the neighborhood size is set to 2 in the first section of the control panel via setting *lbest_neighb_size*, the fifth particle and its two neighbors took that position vector as their local bests. The second-best function value of 15.709 belonged to the first particle, which had position vector [2.9288, 8.7536]; the neighboring second particle took that position as its local best. The second particle produced the third-best function value of 18.227 at position vector [12.911, -3.7448], which became the local best of the neighboring third particle. Those three positions comprised matrix *l* of local bests at iteration 1.

Tables 4 shows how to extract from history *fhist* the column vector, *f*, of function values corresponding to any iteration number.

Table 4: Extracting the Function Values of Any Iteration from "*fhist*"

Update Number ("*k*")	**Iteration Number ("*k* + 1")**	**Code for Extracting Function Values from History**
0	1	fhist(:, 1)
1	2	fhist(:, 2)
2	3	fhist(:, 3)
3	4	fhist(:, 4)
k	k + 1	fhist(:, k + 1)

Table 5 shows how to extract from history *lhist* the matrix of local bests corresponding to any update number or iteration number.

Table 5: Extracting the Local Bests of Any Iteration from "*lhist*"

Update Number ("*k*")	**Iteration Number ("*k* + 1")**	**Code for Extracting Local Bests from History**
0	1	lhist(:, 1:dim)
1	2	lhist(:, (dim + 1):(2*dim))
2	3	lhist(:, (2*dim + 1):(3*dim))
3	4	lhist(:, (3*dim + 1):(4*dim))
4	5	lhist(:, (4*dim + 1):(5*dim))
k	k + 1	lhist(:, (k*dim + 1):((k + 1)*dim))

As with all histories, *lhist* and *fhist* will only be maintained when switches *OnOff_lhist* and *OnOff_fhist* respectively are active at the end of the control panel's first section.

g

Matrix *g* iteratively replicates the global best across all of its rows to accommodate matrix subtraction in the velocity update equation of Gbest PSO. The global best is not usually maintained when Lbest PSO is employed (i.e. when switch *OnOff_lbest* is active); it was maintained for this demonstration only because switch *OnOff_ghist* is active in the control panel, which activates a history of each iteration's global best for analysis.

Typing *g* at the command prompt shows its contents.

```
g =
-2.0065e-002  1.9715e-002
 -2.0065e-002  1.9715e-002
 -2.0065e-002  1.9715e-002
 -2.0065e-002  1.9715e-002
 -2.0065e-002  1.9715e-002
```

It was seen earlier that the second particle's personal best of [-0.020065, 0.019715] was the best of the swarm at iteration 35. Consequently, it was the global best stored to each row of *g* when the search concluded.

ghist

When switch *OnOff_ghist* is activated, the global best is iteratively stored to the row of *ghist* equal to the iteration number. Typing *ghist* at the command prompt will show the full history of global bests: the top row is the global best from initialization of the swarm at iteration 1, and the last row stores the same position shown in *g* above. As the search progressed, both dimensions of the global best approached zero since Ackley's global minimizer is the null vector.

At this point, you may wish to type or paste some rows of the third column of Table 3 at the command prompt to familiarize yourself with how to analyze the global bests of specific iterations.

Table 6: Extracting the Global Best of Any Iteration from "*ghist*"

Update Number ("*k*")	Iteration Number ("*k* + 1")	Code for Extracting Global Bests from History
0	1	ghist(1, :)
1	2	ghist(2, :)
2	3	ghist(3, :)
3	4	ghist(4, :)
4	5	ghist(5, :)
k	k + 1	ghist(k + 1, :)

As with other histories, *ghist* is only maintained when switch *OnOff_ghist* is active at the end of the control panel's first section.

Workspace Organization

Please type "whos" at the command prompt. If the workspace does not display well, try widening or maximizing the Command Window. Typing "whos" lists each variable in the workspace along with its characteristics. Since capitalized variables are shown first, variables of relative importance are lower case by design to cause relatively unimportant variables to float to the top of the list out of sight. For example, switches are named "OnOff…," which organizes them together in the workspace and pushes them off the screen when "whos" is typed.

Typing "who" at the command prompt shows all variables in the workspace without displaying their characteristics.

Knowing these commands will help you find and analyze important data.

Summary

In this section you learnt that the following workspace variables are updated iteratively:

- *x* stores positions,
- *f* stores the function values produced by *x*,
- *p* stores personal bests,
- *fp* stores the function values produced by *p*,
- *g* stores the global best,
- *fg* stores the function value produced by *g*,
- *v* stores velocities, and
- *l* stores local bests.

Furthermore, histories of these variables can be activated or de-activated at the end of the control panel's first section.

Tables 1 – 6 show how to extract the data of particular iteration numbers from activated histories. Since de-activating histories improves execution speed, the copy of "Control_Panel.m" used to generate any particular data can be saved and used to regenerate that data along with the corresponding histories at a later time simply by activating the appropriate switches. This is generally preferable since time is more of the essence than excess data.

Knowing Where Data is Saved

When the proper switches are active as they are by default, figures and workspaces will automatically be saved to the "Data" folder of the current directory. You can browse to this folder by:

(i) clicking "Start" at the bottom left of Windows,
(ii) clicking on "My Documents" in Windows XP or "Documents" in Windows 7,
(iii) typing "MATLAB" to locate the proper folder and pressing Enter,
(iv) browsing to the "Data" folder in the current directory (i.e. the "Current Directory" is shown above MATLAB's Command Window in what resembles the address bar of a browser), and
(v) opening the folder with the most recent date and time in its name.

Figures

The "Figures" folder is where graphs are automatically saved. Each figure was saved using both the *.fig format and *.png formats. This is because (i) switch *OnOff_autosave_figs* is active in the "REGPSO & PSO GRAPHING SETTINGS" subsection of the control panel, and (ii) switch *OnOff_autosave_figures_to_another_format* is active along with "*GraphParams_autosave_format* = 'png'". The purpose of saving to dual formats is that if an *.png file is not to your liking, you can open the corresponding *.fig file in MATLAB and manually modify it. Furthermore, slideshows of recognized file types can easily be generated by right-clicking the first file in the sequence and selecting "Preview".

Workspaces

Just as figures are saved to the "Figures" folder, snapshots of the workspace are saved to the "WS" folder. The name of each saved workspace begins with "RM#," where # represents the regrouping method employed. When standard PSO without regrouping is used, saved workspaces begin with "RM0". When RegPSO is activated to liberate the swarm from the state of premature convergence, saved workspace names begin with "RM1". "RM2" is reserved for future usage. Please browse to folder "WS".

RM#Tr#… (Trial Data)

File "RM0Tr1…" contains the workspace at the end of trial 1. When multiple trials are conducted, the Particle Swarm Optimization Research Toolbox will also save workspaces "RM0Tr2…," "RM0Tr3…," etc., for each trial conducted. Any time multiple trials are conducted, it is recommended to have switch *OnOff_Autosave_Workspace_Per_Trial* activated atop the control panel's first section to use RAM efficiently by saving the workspace at the end of each trial, which both improves execution speed and reduces the probability of memory overflow. It is recommended to leave this switch active as it is by default.

RM#C… (Column Data)

When a set of trials completes, matrices of data are constructed from vectors of trial data and saved to "RM#C…". For example, the graph of mean function value per iteration requires that a matrix of function values be constructed from each trial's vector of function values and averaged column-wise. Statistical analyses are also computed from combined trial data. Column data is saved after each column generated to free up RAM, which significantly improves the speed of execution while significantly reducing the probability of memory overflow. Pertinent column data is reconstructed before saving table data.

RM#T… (Table Data)

Table features are only available in the full version of the Particle Swarm Optimization Research Toolbox. Tables are most useful when incrementing a PSO parameter across columns while holding other settings static to understand the effect of the parameter being incremented. After each table of statistics generated, the workspace is saved to "RM#T…". For the free demo version of the toolbox, this workspace will not appear.

Since only one trial was conducted for only one objective with no value incremented per column, all three workspaces are in this case essentially the same.

Actual Windows

In early versions of the toolbox, it was necessary to load a saved workspace to analyze the settings that produced its data. I created transparent filenames to look into the workspace through the filename, which allowed me to locate the file of interest without loading multiple workspaces. For example, the following are encoded into the filename after "RM0C".

- "Ack" – the first three letters of the objective name
- "d2" – the problem dimensionality
- "np5" – the number of particles in the swarm
- "v_cl1" – whether or not velocity clamping is active
- "gr1" – the number of groupings performed: relevant for RegPSO
- "k35" – the number of iterations performed
- "n1" – the number of trials performed
- "ca1.4962" – the value of the cognitive acceleration coefficient
- "cb1.4962" – the value of the cognitive acceleration coefficient
- "vm%0.5" – the percentage of the range to which velocities are clamped
- "w_i0.9" – the initial value of the inertia weight
- "w_f0.4" – the final value of the inertia weight
- "0.10052" – the best function value produced by the trial
- "23Aug-10-20.59.27" – the date and time the workspace was created, which guarantees uniqueness of filenames to avoid accidental overwriting of data.

When naming workspaces of individual trials, I was mostly interested in transparently seeing each trial's final function value. As the Particle Swarm Optimization Research Toolbox generated trials automatically, I watched the folder to which data was being saved to monitor the function value at the end of each trial's workspace name. Transparently displaying data in the filename was preferable to cluttering the command window with outputs since data was sometimes generated for tens of thousands of trials while automatically incrementing parameters in search of high-quality PSO parameters, such as those published in Chapter III of thesis [2]. When the function value was not being minimized well, I closed the application and fixed the bug rather than wasting time. Function value "0.10052" is visible before the date at the end of workspace "RM0Tr1…"; unfortunately, to make room for it, I had to abbreviate other portions of the filename.

Activating and De-activating Graphs

De-activating Phase Plots, Position vs. Iteration Graphs, and Benchmark Graph

In subsection "REGPSO & PSO GRAPHING SWITCHES" of the control panel's section "(5) REGPSO & PSO GRAPHING SWITCHES AND SETTINGS," please de-activate switches *OnOff_phase_plot*, *OnOff_graph_x,* and *OnOff_graph_ObjFun_f_vs_2D* by setting them to "logical(0)". You may activate and de-activate switches by typing "true" and "false" instead of "logical(1)" and "logical(0)," but switching between 1's and 0's seems easier.

Activating Graphs of Personal Bests & Global Best vs. Iteration

While in subsection "REGPSO & PSO GRAPHING SWITCHES," please activate switches *OnOff_graph_g* and *OnOff_graph_p*, which show how the global best and personal bests respectively change over time.

Activating Graphs of Function Values vs. Iteration

To examine the function values of the global best and of each individual particle, please activate switches *OnOff_graph_fg* and *OnOff_graph_f* respectively.

Now press "F5" to generate different graphs for what will otherwise be the same trial analyzed above; the changes made within the control panel will be saved automatically.

Interpreting Newly Activated Graphs

Function Value vs. Iteration

The graphs of function value versus iteration in Figures 100,001 – 100,005 show how each particle sometimes steps in a poor direction that adversely affects its function value, but the overall trend lines show progress. The decreasing trend lines are largely due to each particle's iterative acceleration toward the global best.

Personal Bests & Global Best vs. Iteration

The graphs of personal bests vs. iteration in Figures 101 – 105, and of the global best vs. iteration in Figure 1000, show the bests gradually honing in on global minimizer [0, 0]. Unlike phase plots, position vs. iteration graphs can be generated in multiple dimensions since they graph each dimension versus the iteration number instead of graphing dimension 2 versus dimension 1; however, even position vs. iteration graphs can become difficult to interpret for dimensionalities greater than 7 due to the number of lines graphed per figure.

Global Best's Function Value vs. Iteration

Unlike each particle's function value in Figures 100,001 – 100,005, the global best's function value graphed in Figures 1,000,000 and 1,000,001 never deteriorates from one iteration to the next since the global best is the swarm's memory of the best position discovered. Only when the swarm is restarted using Van den Bergh's multi-start approach (i.e. when *OnOff_MPSO* is active) can the global best's function value increase from one iteration to the next, since with each restart the global best is re-initialized rather than remembered as with RegPSO.

Switching Between Lbest and Gbest PSO's

In section "(1) BASIC SWITCHES & PSO ALGORITHM SELECTION" under subsection "PSO ALGORITHM SELECTION," de-activate switch *OnOff_lbest* to activate Gbest PSO. It may be easiest to press Ctrl + Home, Ctrl + F, type *OnOff_lbest*, and press Enter to locate the setting.

Switching to Static Inertia Weight

While in subsection "PSO ALGORITHM SELECTION," de-activate switch *OnOff_w_linear* to switch from a linearly varying inertia weight to a static weight. The static value, *w,* is set in section "(2) PARTICLE SWARM ALGORITHM SETTINGS," though it does not need to be changed at this point.

Changing the Swarm Size

In section "(2) PARTICLE SWARM ALGORITHM SETTINGS," change the number of particles, *np*, from 5 to 3. Again, it may be quicker to press Ctrl + F and type "*np".*

Changing the Number of Iterations

Near the end of section "(2) PARTICLE SWARM ALGORITHM SETTINGS," increase the number of iterations from 35 to 100 via setting *max_iter_per_grouping.*

Activating Swarm Trajectory Graphs

Press Ctrl+F and type "(5)" to move directly to section "(5) REGPSO & PSO GRAPHING SWITCHES AND SETTINGS". In subsection "REGPSO & PSO GRAPHING SWITCHES," activate the first three switches and de-activate the remaining seven. Notice that the comments describe the switches.

Re-using Figures

Near the beginning of the next subsection, "REGPSO & PSO GRAPHING SETTINGS," activate switch *OnOff_reuse_figures*, which closes each figure before generating another to avoid the memory overflow or system drag that could otherwise result from generating many figures.

Marking Personal Bests and the Global Best

Toward the end of subsection "REGPSO & PSO GRAPHING SETTINGS," notice that switches *OnOff_mark_personal_bests* and *OnOff_mark_global_best_always* are active by default. These activate black tags on the swarm trajectory maps denoting where the global best and personal bests are located, which facilitates understanding of particles' movements. Leave these switches active.

Changing the Objective Function

Let's also change the objective from Ackley to Rastrigin: press Ctrl + End to move to the end of the control panel and change *objective_id* from 1 to 5. There is a more detailed discussion of objective selection in the next chapter.

Now, press F5 to begin execution, and confirm that the displayed settings are correct. Please be patient while swarm trajectory graphs are generated.

Analyzing Swarm Trajectory

The small swarm of three particles *prematurely converged* near the local minimizer slightly below [0, 1]. Since switch *OnOff_zoom_on_final_graph* is active in subsection "REGPSO & PSO GRAPHING SETTINGS" of section "(5) REGPSO & PSO GRAPHING SWITCHES AND SETTINGS," the final graph zoomed in on the final swarm state. The third particle approximated the local minimizer better than the other two particles. Notice in the final figure generated to folder "Figures" that all values displayed on the horizontal axis are to be divided by 10^3, which implies that better accuracy was obtained on the vertical axis than on the horizontal.

Re-using Figures

Switch *OnOff_reuse_figures* causes the same figure number to be reused for all graphs, which prevents memory overflow when multiple graphs are generated. This allows quick generation of high-quality figures regardless of the number of figures generated. This switch was designed for use with Swarm Trajectory graphs, which take swarm snapshots at regular intervals and overlay them on high-quality, colored contour maps, each of which occupies RAM. When used in conjunction with switch *OnOff_autosave_figs,* each figure is automatically saved before generating a new one.

The downside is that figures must be opened to be analyzed; however, this is not much of an inconvenience if you know how to view them as thumbnails. To do this, open the "Data" folder and the subfolder named with the most recent date and time, which will be at the end of the list if viewing by name or date. Next, open the "Figures" folder. On Windows XP, press Alt + {V, H} to view the .png files as thumbnails. On Windows 7, click the drop-down arrow under the folder's search field, and select the desired thumbnail size. If your operating system is working properly, you should now see automatically saved .png files serving as visual previews of the .fig files to their left. After previewing an image, you can use the right and left arrow keys to see how the swarm progressed over the course of the search.

Activating Regrouping PSO (RegPSO)

Regrouping PSO (RegPSO) regroups the swarm when premature convergence is detected or when the maximum number of iterations or function evaluations per grouping is reached; this liberates the swarm from the state of premature convergence, thus enabling continued exploration. Let's activate RegPSO via switch *OnOff_RegPSO* in the "PSO ALGORITHM SELECTION" subsection of the control panel's section "(1) BASIC SWITCHES & PSO ALGORITHM SELECTION". Remember, you can always locate a switch or setting quickly by pressing Ctrl + Home, Ctrl + F, typing its name, and pressing Enter.

Near the end of section "(2) PARTICLE SWARM ALGORITHM SETTINGS," change the maximum number of iterations per grouping from 100 to 150 via setting *max_iter_per_grouping*.

Near the end of section "(3) REGPSO ALGORITHM SWITCHES & SETTINGS," notice that the maximum number of iterations over all groupings is 250 for setting *max_iter_over_all_groupings.* This specifies the maximum number of iterations across all groupings of a RegPSO trial. The value does not need to be changed at the moment.

Setting the Format for Auto-Saved Figures

Specify the format to which figures will be saved via setting *GraphParams_autosave_format* in subsection "REGPSO & PSO GRAPHING SETTINGS" of section "(5) REGPSO & PSO GRAPHING SWITCHES AND SETTINGS". Let's change to bitmap format by replacing "png" with "bmp".

Please press F5 to begin execution, and verify that the displayed settings are correct.

Analyzing Swarm Trajectory of RegPSO

In the "Data" folder of the current directory, browse to the folder with the most recent date and time in its name: if viewing by name or date modified, this will be at the end of the list. Double click the "Figures" folder, and double click the bitmap corresponding to iteration 138; the maroon rectangle in the figure depicts the space within which the swarm regrouped about the global best to continue searching for the true global minimizer while remembering the global best of the previous grouping. Notice the efficiency of regrouping within this space versus restarting on the original search space of range 10.24 per dimension.

Now double click the figure corresponding to iteration 178. After regrouping, the first particle found a new personal best in the well that contains the global minimizer. Though this personal best is not yet as high-quality as the global best in the well above it due to its distance from the center of the well, it will draw the particle back to the well as if by curiosity.

Please double click the figure corresponding to iteration 188. The personal bests of the first and third particles are now both within the well containing the global minimizer, and so is the global best. Also, the second particle has discovered a new personal best left of the well containing the global minimizer, which demonstrates the exploratory benefit of regrouping the swarm.

Please double click the bitmap corresponding to iteration number 198. The global best and all personal bests are now located within the well containing the global minimizer. With only three particles, RegPSO has successfully approximated the true global minimizer in less than

200 iterations. The solution refinement phase is now underway with some exploratory momenta. The remaining figures show the swarm refining its approximation of the global minimizer.

Analyzing Workspace Variables for RegPSO

Please type "whos" at the command prompt, and widen the Command Window as necessary to display results aesthetically. Notice that for each history maintained per grouping (e.g. *fhist, ghist, phist, vhist,* and *xhist*), "current_trial" histories were also maintained (i.e. *fhist_current_trial, ghist_current_trial, phist_current_trial, vhist_current_trial,* and *xhist_current_trial*); each of these "current_trial" histories horizontally concatenates each grouping's history across all of a trial's groupings.

After saving each grouping's data to a temporary workspace, that grouping's histories are cleared from the RAM. Since "current_trial" histories are reconstructed at trial's end from these saved workspaces, RAM is used more efficiently than were each trial's "current_trial" histories maintained across groupings.

Per grouping histories such as *fhist, ghist, phist, vhist*, and *xhist* are cleared from RAM after being saved to the hard drive at the end of each grouping; consequently, at trial's end, they store only the histories of the final grouping. Please type or paste "whos ghist ghist_current_trial" at the command prompt to compare the sizes of *ghist* and *ghist_current_trial*; notice that the latter stores the global best of all iterations over both groupings of RegPSO, whereas *ghist* stores only the global bests of the final grouping. Both histories contain the global best of the final iteration in the final row: [2.2955e-005, -1.3974e-004]. To learn more about the details of current trial histories or other workspace variables, please see their descriptions in "Appendix A: Definitions".

Please type *fg_array* at the command prompt to see the function value produced by each grouping's global best. This shows that RegPSO improved solution quality in the second grouping by liberating the swarm from the state of premature convergence suffered in the first grouping. <u>Regrouping is more effective than simply restarting the search as evidenced by the tremendously improved solution quality of the second grouping, which stems from the efficiently sized regrouping space.</u>

Switching Termination Criterion from Iterations to Function Evaluations

Within the control panel, please press Ctrl + Home, Ctrl + F, type "func_evals," and press Enter to quickly find switch "*OnOff_func_evals*" in the "TERMINATION CRITERIA FOR REGPSO & PSO ALGORITHMS" subsection of section "(1) BASIC SWITCHES & PSO ALGORITHM SELECTION". Please activate this to switch the termination criterion from a maximum number of iterations to a maximum number of function evaluations.

Press Ctrl + F again, type "max_FE," press Enter until reaching *max_FEs_per_grouping*, and set its value to "200000" (i.e. 200,000 without a comma). Press Ctrl + F again and press Enter until reaching setting *max_FEs_over_all_groupings*, and set this to “800000” (i.e. 800,000 without a comma).

Changing the Number of Trials

Please press Ctrl + Home, Ctrl + F, type *num_trials*, and press Enter. Change this from 1 to 5 to generate statistics over 5 trials. In practice, you would conduct considerably more trials for statistical reliability, but this is merely for demonstration.

Changing the Problem Dimensionality

While in section 2, also change the dimensionality from 2 to 30 via setting *dim* to test the RegPSO algorithm in a more difficult, higher-dimensional space.

Changing the Swarm Size

Since the problem to be solved is now more difficult, increase the swarm size, *np*, from 3 to 20 on the line below *dim* in the same “(2) PARTICLE SWARM ALGORITHM SETTINGS” section of the control panel.

De-activating Unnecessary Histories

Please de-activate each “*OnOff_...hist*” switch at the end of the control panel’s first section to de-activate histories. Simply press Ctrl + F, type “*OnOff_fghist*,” and set each such switch to logical(0).

Disabling Graphs

Press Ctrl + F in the control panel, type “(5)”, and press Enter twice to move directly to graphing section `“(5) REGPSO & PSO GRAPHING SWITCHES AND SETTINGS”`. Then de-activate switch *OnOff_graphs* by changing its value from logical(1) to logical(0).

Analyzing the Statistics of Multiple Trials

User Input Validation

Please press F5 now to begin execution, which will automatically save changes made to the control panel. Execution will take a few minutes since 5 long trials are being conducted. In the mean time, notice that the settings displayed in the User Input Validation section of MATLAB’s Command Window reflect the changes made in the control panel:

- RegPSO is still applying its regrouping mechanism to Gbest PSO.
- A maximum of 200,000 function evaluations have been allotted to each grouping, and regrouping will continue until 800,000 function evaluations are reached unless *true_global_minimum* is achieved without error before then (i.e. function values less than 10^{-323} cannot be differentiated from the true global minimum of 0 by MATLAB).
- For all groupings, premature convergence will be considered to have occurred when the Euclidean swarm radius shrinks to 0.011% of the grouping’s original swarm radius, which will trigger automatic regrouping. Once premature convergence is detected, particles will be re-initialized within a regrouping space defined per dimension to be directly

proportional to the degree of uncertainty inferred per dimension from the swarm state at premature convergence.

- The regrouping factor is 1.2 divided by the stagnation threshold, *stag_thresh*; this inverse proportionality ensures that the closer particles converge before regrouping, the larger the regrouping factor will be, which ensures that the regrouping space is sufficiently large to facilitate an ongoing search for the global minimizer.
- A static weight is now displayed where initial and final values were displayed in conjunction with linear variation.

Each Trial's Data

Each trial's data is displayed to the Command Window before generating the next trial's data so that any erroneous setting or poorly performing parameter combination will become obvious as soon as possible. When performance suffers, MATLAB can be closed by clicking the X at the top right of the program, and "Control_Panel.m" can be double-clicked to change the settings and try again. While the data is being generated, the following will explain the data to be displayed after each of the five RegPSO trials:

- *fg* (i.e. the function value of the global best) will indicate how effectively the RegPSO algorithm minimized the thirty-dimensional Rastrigin benchmark.
- Like update counter *k*, *RegPSO_k* initializes to 0 during swarm initialization; however, instead of re-initializing with each regrouping as *k* does, it continues to increment with each iteration of RegPSO. Consequently, its value at the end of each trial is 1 less than the number of RegPSO "iterations" conducted, which is the number of times positions and velocities have been updated.

  One trial's final value of *RegPSO_k* may be less than the final value for other trials if the true global minimum is achieved without discernable error, which causes the search to terminate so no time is wasted in computation when further optimization is impossible: more specifically, once error in the function value becomes indiscernible (i.e. once $f < 10^{-323}$) so that only the decision variables can possibly have discernable error due to the complexity of the mapping from inputs to output, the search will terminate since decision variables cannot be further optimized once error in the objective function becomes indiscernible. The third trial will demonstrate this property since the true global minimum will be achieved before the trial concludes so that continued "optimization" would be pointless.
- The number of function evaluations or iterations conducted over the course of the trial is displayed. As with *RegPSO_k*, this can be less than the maximum value if the true global minimum is achieved without error, which will be demonstrated by the third trial.
- The global best, *g*, is displayed after each RegPSO trial so that patterns in any problematic dimensions can be discovered and analyzed as soon as possible. (Only one row of matrix *g* is displayed, since each row is the same for purposes of matrix subtraction in the velocity update equation.)
- The range per dimension of the final regrouping space is displayed after each RegPSO trial since it is related to each dimension's solution quality. Conceptually, since the range per dimension of the regrouping space is proportional to the uncertainty on that

dimension, overconfidence on any particular dimension can be counterproductive by preventing proper exploration. Analytically, any dimension of the global best that is not well optimized is likely a result of a range so small as to have eliminated the global minimizer from the regrouping space on that dimension.

- When switch *OnOff_k_after_each_grouping* is active, each update number at which premature convergence was detected is stored to *RegPSO_k_after_each_grouping*. Adding 1 to each column of this row vector would produce the iteration numbers at which regrouping was triggered, with the difference being that iterations include initialization of the swarm before it begins updating.
- When switch *OnOff_fg_after_each_grouping* is active, the function value of each grouping's global best is stored to *RegPSO_fg_after_each_grouping*.
- The trial number is displayed after the data above.

During execution, a progress meter will intermittently display the percentage of calculations completed and the estimated time of completion.

Performance Measures

Success Rate

Once all trials have completed, the toolbox automatically displays the number of trials successful according to the threshold required for success, the number of trials unsuccessful, and the success rate. For the current calculations, *thresh_for_succ* = 100, as displayed near the end of the User Input Validation phase.

Function Evaluations Performed

Also displayed are the mean number of function evaluations required to reach the threshold for success and the mean number conducted per trial (i.e. even after reaching the threshold for success). The latter is less than the maximum of 800,000 function evaluations set in *max_FEs_over_all_groupings* because the third trial reduced the 30-D Rastrigin benchmark to zero with no error discernable by MATLAB after 693,120 function evaluations (i.e. the objective function was minimized to less than 10^{-323}). This shows the ability of RegPSO to enable continued progress long after "standard" PSO would have stagnated.

Global Best

Once the goal of minimizing the objective function to its true global minimum is accomplished, no further optimization of the decision variables is possible regardless of the algorithm employed. Even though the third trial reduced the function value to its true global minimum of 0 without discernable error (i.e. $f < 10^{-323}$), there was still some error discernable in the n-dimensional decision vector due to the complicated mapping from n inputs to one scalar output. As taught in high school, more digits should be retained during computation than desired in final solution quality; consequently, since optimization algorithms are generally applied to functions with fairly convoluted mappings from inputs to output, the inputs should retain greater decimal accuracy than desired in the quality of the final solution. It also follows that the solution accuracy possible on each dimension increases as the problem dimensionality decreases.

Statistics

From the best function value per trial, the median, mean, minimum, maximum, and standard deviation are calculated. As can be seen from the statistics generated for the 30-D Rastrigin benchmark, regrouping was quite effective at liberating the swarm from the state of premature convergence to facilitate continued progress.

Time Required

Also displayed are:

- Average time per trial in seconds,
- Elapsed time over all trials in seconds,
- Iteration numbers at which each trial satisfied the threshold for success,
- Time of completion.

Best Function Value per Trial

The function value of each trial's global best is stored to workspace variable *fg_final_per_trial*, from which the median, mean, minimum, maximum, and standard deviation are calculated.

You might want to save a backup copy of "Control_Panel.m" with these settings intact to facilitate future RegPSO testing.

Summary

This walk through was intended to introduce you to the primary switches and settings of the Particle Swarm Optimization Research Toolbox. The next chapter presents more details on several important topics, including a more in-depth discussion of Using RegPSO to Escape from Stagnation.

III. More Details on Select Topics

Specifying Gbest or Lbest PSO as the Core Algorithm

Improvements to PSO generally build on one of two core algorithms: (i) global best (Gbest) PSO or (ii) local or neighborhood (Lbest) PSO.

To activate Lbest PSO, set switch *OnOff_lbest* to logical(1) under the "ALGORITHM" subheading of section ""(1) BASIC SWITCHES & PSO ALGORITHM SELECTION"" in "Control_Panel.m". For Gbest PSO, set the switch to logical(0). It is recommended to use "logical(1)" and "logical(0)" rather than "true" and "false" since doing so allows switches to be activated and de-activated more quickly.

Setting the Inertia Weight

To use a static inertia weight, de-activate switch *OnOff_w_linear* in section "(1) BASIC SWITCHES & PSO ALGORITHM SELECTION" of the control panel by setting it to logical(0), and set the static value of *w* in section "(2) PARTICLE SWARM ALGORITHM SETTINGS".

When switch *OnOff_w_linear* is active, the initial weight, *w_i*, will be used for the first velocity update, with linear decrease or increase until reaching the final weight, *w_f* – assuming the simulation runs full course; however, if RegPSO or MPSO is activated, the final values of the inertia weight may not be used since a regrouping or restart will be triggered once stagnation is detected to avoid wasteful calculations in a non-productive state.

The relevant portions of the control panel are pasted below. Take care not to accidentally change "if *OnOff_w_linear* == logical(1)" in section "(2) PARTICLE SWARM ALGORITHM SETTINGS" since this merely checks whether initial and final weights or one static weight should be loaded based on the status of switch *OnOff_w_linear*.

```
%$%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%(1) BASIC SWITCHES & PSO ALGORITHM SELECTION%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 OnOff_w_linear = logical(0); %Turn time-varying inertia weight on or off.

...

%$%%%%%%%%%%%%%%%%%%%%%%%%%%%
%(2) PARTICLE SWARM ALGORITHM SETTINGS%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if OnOff_w_linear == logical(1) %In the case of time-varying inertia
   w_i = 0.9; %initial value used in first velocity update
   w_f = 0.4; %final value used in final velocity update
 else %i.e. if a static inertia weight is desired
   w = 0.72984;
end
```

Setting the Velocity Clamping Percentage

Consider an example search space $[x_{\min}, x_{\max}]^n = [-100, 100]^n$, where n represents the problem dimensionality: each dimension has a range of 200 centered at 0. The maximum velocity, $v_{\max}$, has often been taken in the literature to be some percentage of the maximum position, $x_{\max}$; however, this unnecessarily creates dependence on the origin of the search space though particle behavior should be independent of the reference frame chosen.
Since many real-world problems have search spaces that are not symmetric across the origin, consider the same search space shifted 100 units to the right by changing the objective function from $f(\vec{x})$ to $f(\vec{x} - \overrightarrow{100})$. The search space then becomes $[x_{\min}, x_{\max}]^n = [0, 200]^n$. Defining

$v_{\max}$ as the same percentage of $x_{\max}$ would produce twice the velocity clamping value as before the search space was shifted. Such dependence on the origin of the search space is easily removed by defining $v_{\max}$ more generally in terms of the range of the search space.

When the search space is centered at the origin of Euclidean space, the two approaches are identical since the range per dimension in that case is $2x_{\max}$ so that a percentage of the range is also a percentage of xmax. The definition of velocity clamping used by the Particle Swarm Optimization Research Toolbox is therefore consistent with the traditional definition for objective functions whose search spaces are centered at the origin since a percentage of the range is also a percentage of $x_{\max}$. The definition employed simply removes dependence on the reference frame so that particles – like particles in physics – will behave the same way regardless of the reference frame by which their motions are measured.

Furthermore, some real-world problems have different ranges of feasible values for different decision variables. Therefore, the Particle Swarm Optimization Research Toolbox defines $v_{\max}$ as a percentage of the range of the search space per dimension. To set $v_{\max} = 100$ for the example search space used above, simply set "*vmax_perc*" to 0.5 in section "(2) PARTICLE SWARM ALGORITHM SETTINGS" of the control panel, since the range on each dimension of the example is 200. This has the effect of setting $v_{\max} = x_{\max}$ for the original un-shifted search space.

$$v_{\max,j} = v_{\max\_\text{perc}} \times range_j,$$
$$\text{where } range_j = x_{\max,j} - x_{\min,j}$$

This more general definition boils down to the traditional definition for benchmarks that use the same range per dimension, but it allows more flexibility for real-world applications whose decision variables may assume different ranges of values.

When asymmetric initialization is employed, the initialization space, $[\frac{x_{\max}}{2}, x_{\max}]^n$, differs from the search space, $[-x_{\max}, x_{\max}]^n$. In this case, velocities are clamped to the range of the full search space, which corresponds to $x_{\max}$ (i.e. the range equals $x_{\max} -$ - $x_{\max}$, which equals $2x_{\max}$, so that a percentage of the range is also a percentage of $x_{\max}$); this is consistent with what is generally done in the literature. The only difference is that the Particle Swarm Optimization Research Toolbox more generally defines velocities as a percentage of the range of the search space for real-world application problems for which it would be nonsensical to clamp particle's positions based on the value of *xmax* itself: e.g. if optimizing the dimensions of a sofa with length specified to lie between 6ft and 8 ft, clamping velocities based on *xmax* rather than on the range of the search space along that dimension would be nonsensical since 0.5*8 ft = 4 ft maximum step size per dimension, whereas 0.5*(8 ft – 6 ft) = 1 ft; obviously, for a search space of range 2 ft, a maximum step size of 4 ft would be absolutely nonsensical.

Note that the range and center of the initialization space are defined per objective in "Objectives.m".

Selecting the Objective Function(s)

Objectives are listed in alphabetical order within file "Objectives.m," where each is assigned an *objective_id*. ID's control the order in which objectives are tested in the full version of the toolbox, and they determine which objective will be used in the free version. If you are using the free demo, just set the *objective_id* in final section "(6) OBJECTIVES" of the control panel to the id corresponding to the desired objective function. Table 7 below summarizes the correspondence between each objective function and its *objective_id* as set by default within "Objectives.m". You may change how ID's are assigned within that file if you like – these are merely the default assignments.

Table 7: Objective ID's

Objective	**ID**
Ackley	1
Griewangk	2
Quadric	3
Noisy Quartic	4
Rastrigin	5
Rosenbrock	6
Schaffer's f6	7
Schwefel	8
Sphere	9
Weighted Sphere	10
NN Training (via add-in)	11

If you have purchased the full version of the Particle Swarm Optimization Research Toolbox, you can test a PSO algorithm across multiple objectives by (i) setting *TableParams_num_tables* in the control panel's section "(4) TABLE SWITCHES & PARAMETERS" to the total number of objectives to be tested, (ii) activating switch *TableParams_OnOff_Tbl_Incr_objective*, (iii) setting *objective_id* to 0 in the final section of the control panel, and (iv) specifying in "Objectives.m" the order of the desired objectives by assigning id 1 to the first objective, 2 to the second, and so on until reaching the id corresponding to the number of objectives to be tested. It is recommended to test objectives from most difficult to easiest so that any difficulties will be discovered quickly. The free version is able to conduct multiple trials automatically for one objective at a time.

Setting the Center and Range of the Initialization Space

The PSO Research Toolbox allows users to specify real-world initialization spaces rather than requiring initialization to be either symmetric or asymmetric across the origin of n-dimensional Euclidean space. As an example, suppose you are to find the optimal combination of vehicle length, width, and height to maximize fuel efficiency. And suppose that for any reasonable combination of these three inputs, one or more vehicle types can be classified and tested in a simulated environment. Then the curvature of the vehicle and its weight can be mapped to from these three decision variables so that only length, width, and height must be optimized. If we expect the ideal length to lie somewhere between 5ft and 15 ft, the ideal width between 1ft and 8ft, and the ideal height between 2 ft and 10 ft; the range of the initialization space, *range_IS0* would be input as vector [10, 7, 8] and the center of the initialization space, *center_IS0* as vector [10, 4.5, 6].

The Particle Swarm Optimization Research Toolbox allows users to specify the center and range of the initialization space as vectors for practical problems like this – leaving the user only to write his or her problem to be solved. But for objectives whose initialization space is the same per dimension, the user can input the center and range as scalars for convenience. For example, for the standard Rastrigin objective *center_IS0* = 0 and *range_IS0* = 2*5.12 = 10.24 can be input, from which the Particle Swarm Optimization Research Toolbox will internally construct matrices *range_IS* and *center_IS* of order *np* by *dim*.

Using RegPSO to Escape from Stagnation

The motivation behind RegPSO is to efficiently regroup the swarm for continued progress once repeated attractions toward the global best have caused to swarm to collapse. This is achieved via the following steps:

(i) Detect when the swarm has prematurely converged [1],
(ii) Calculate the swarm's uncertainty along each dimension from the swarm state at premature convergence,
(iii) Define the range of each dimension of the regrouping space proportionally to the uncertainty on that dimension,
(iv) Center the regrouping space at the global best,
(v) Re-initialize the swarm within the new regrouping space,
(vi) Re-initialize each particle's personal best to be equal to its current position as done during initialization of the swarm with "standard" PSO,
(vii) Re-calculate the velocity clamping vector using the range per dimension of the new regrouping space,
(viii) Continue Gbest PSO as usual within the regrouping space.

This process is repeated each time premature convergence is detected, which enables the swarm to continue improving solution quality as shown at the end of Section III: A Guided Walk Through.

To detect premature convergence, the normalized swarm radius criterion proposed by Van den Bergh in his PhD thesis is utilized [3]. Particles are considered confident in their approximation along dimensions for which no particle significantly deviates from the global best; whereas, they are considered to be relatively uncertain along any dimension for which any number of particles has a relatively large deviation from the global best. The uncertainty per dimension is simply calculated as the maximum distance of any particle from the global best along that dimension. Each dimension's uncertainty is then multiplied by the static regrouping factor to determine the space within which to regroup the swarm about the global best. After regrouping, Gbest PSO begins as usual with velocities clamped to the same percentage of the new regrouping space to which they were clamped originally, with the global best remembered across groupings to be improved upon.

RegPSO works well with Clerc's equivalent parameters [2, Ctrl + F, "Clerc"]: constriction coefficients $c_1 = c_2 = 1.49618$, and static inertia weight $\omega = 0.72984$. Other parameter combinations can be found to postpone stagnation in PSO without regrouping, as shown in Chapter III of thesis [2]; however, empirical testing showed that it is more beneficial to use Clerc's equivalents with RegPSO since they produce quicker convergence to the global best, which allows the regrouping mechanism more chances to improve solution quality over a search of any finite duration. In Table V-2 of thesis, RegPSO achieves better performance by quickly converging and regrouping than "standard PSO" achieves using high-quality parameters adept at postponing stagnation.

Following the steps below will lead to effective RegPSO implementation.

(1) Activate RegPSO by setting switch *OnOff_RegPSO* to logical(1) in the control panel's first section.
(2) Specify the maximum number of function evaluations or iterations per grouping by pressing Ctrl+F and locating either *max_FEs_per_grouping* or *max_iter_per_grouping* at the end of the control panel's second section (i.e. depending on the status of switch OnOff_func_evals). Limiting each grouping to 100,000 function evaluations or 100,000/np iterations works well, though for some applications another value between 50,000 and 200,000 function evaluations might perform better.
(3) Specify the maximum number of function evaluations or iterations over all grouping by pressing Ctrl+F and locating either *max_FEs_over_all_groupings* or *max_iter_over_all_groupings* at the end of the control panel's third section (i.e. just before the fifth section since the fourth section is not available in the free version of the toolbox). Limiting each trial to 800,000 function evaluations over all groupings or 800,000/np iterations allows enough regroupings for the swarm to significantly improve performance over "standard" PSO.
(4) While the default value of the stagnation threshold, *stag_thresh*, is recommended for general usage, it can be changed within the control panel's third section for specific applications. This setting determines the sensitivity of the threshold at which stagnation is considered to have occurred, after which regrouping is triggered. A larger value for *stag_thresh* triggers regrouping sooner; however, regrouping too soon does not allow for much solution refinement (aka exploitation), and allowing a good amount of solution refinement increases the reliability of the uncertainty inferred since particles tend to bounce around quite a bit early in the search. The default value of *stag_thresh* = 1.1×10^{-4} allows for a decent amount of solution refinement before regrouping and generally produces meaningful measures of uncertainty.
(5) While the default value of the regrouping factor, *reg_fact*, is recommended for general usage, it can be changed within the control panel's third section for specific applications.

A larger value of *reg_fact* produces a larger regrouping space, and a smaller value has the opposite effect. The regrouping factor is defined to be inversely proportional to the stagnation threshold since a smaller stagnation threshold allows particles to collapse within closer proximity of each other during the solution refinement phase, which mandates a larger regrouping factor by which to achieve a decently sized regrouping space. Were the regrouping factor not defined as inversely proportional to the stagnation threshold, tiny stagnation thresholds would produce tiny regrouping spaces, resulting in an undesirable dependency that would not facilitate reliable escape from entrapping local wells. The default value of 1.2/*stag_thresh* performed well across a suite of popular benchmarks as shown in chapter five of thesis [2].

Specifying the Maximum Number of Iterations or Function Evaluations

Decide whether to use a maximum number of function evaluations or iterations as a termination criterion and set switch *OnOff_func_evals* in the control panel's section "(1) BASIC SWITCHES & PSO ALGORITHM SELECTION" accordingly.

Maximum per Grouping

Settings *max_FEs_per_grouping* (used when switch *OnOff_func_evals* is active) and *max_iter_per_grouping* (used when *OnOff_func_evals* is inactive) are located near the end of section "(2) PARTICLE SWARM ALGORITHM SETTINGS". These specify the maximum number of function evaluations or iterations allowed by any one grouping of the swarm. Note that standard PSO without regrouping consists of only one grouping and therefore uses these settings as its termination criteria.

Maximum over all RegPSO Groupings

RegPSO regroups the swarm about the global best when stagnation is detected to continue searching efficiently rather than restarting on the original initialization space. For RegPSO, set the maximum number of function evaluations, *max_FEs_over_all_groupings*, or maximum number of iterations, *max_iter_over_all_groupings*, over all groupings toward the end of section "(3) REGPSO ALGORITHM SWITCHES & SETTINGS" in addition to setting a maximum per grouping.

Maximum over all MPSO Starts

For multi-start PSO (MPSO), which continually restarts PSO on the same initialization space and proposes the best solution found, set *MPSO_max_FEs* or *MPSO_max_iters* toward the end of section "(1) BASIC SWITCHES & PSO ALGORITHM SELECTION" in addition to the maximum per grouping and possibly the maximum over all groupings (i.e. if RegPSO is activated). The user is also given the option to specify a maximum number of restarts via setting *MPSO_max_restarts*. MPSO, as presented by Dr. Van den Bergh, continually restarts guaranteed-convergence PSO (GCPSO), but the Particle Swarm Optimization Research Toolbox allows the user to apply the multi-start concept to any other algorithm in the toolbox.

Confining Particles to the Search Space

For problems such as the example above, the initialization space may be nothing more than an educated guess outside of which other valuable solutions might exist. However, for benchmarks and many real-world problems, it is known in advance that the only feasible solutions lie within the initialization space. In such cases, it is beneficial to confine particles to that space. The Particle Swarm Optimization Research Toolbox provides two methods for accomplishing this.

Velocity Reset

This approach works by resetting the velocities responsible for pushing particles out of bounds to prevent them from continuing out of bounds. It is activated via switch *OnOff_v_reset* in the control panel's first section under subheading "PSO ALGORITHM SELECTION".

Position Clamping

I devised another approach, which is to simply clamp particle's positions the same way their velocities are clamped – i.e. when any dimension is too large, replace it with the maximum value of *xmax*, and when it travels too far in the negative direction, replace it with the minimum value of *xmin.* This method is activated via switch *OnOff_position_clamping* in the control panel's first section under subheading "PSO ALGORITHM SELECTION".

It should be noted that when asymmetric initialization is used, the initialization space, $[x_{\max}/2, x_{\max}]^n$, is different from the search space, $[-x_{\max}, x_{\max}]^n$. In this case, particles are confined to remain within the search space rather than within the initialization space; this is consistent with what is generally done in the literature.

Generating Statistics Automatically

Specify the desired number of trials to be conducted per column (i.e. per set of trials) via setting *num_trials*. Each trial is generated using a different yet repeatable sequence of pseudo-random numbers, which is achieved by incrementing the starting state of the randomizer by prime number 104,729 with each trial.

When *num_trials* is set to 1, the final function value for the single trial conducted will be displayed automatically. When a set of trials is conducted (i.e. when *num_trials* > 1), *fg_final_per_trial* will be displayed automatically following the median, mean, minimum, maximum, and standard deviation.

Generating Figures

To generate figures, first activate main switch *OnOff_graphs* atop section "(5) REGPSO & PSO GRAPHING SWITCHES AND SETTINGS" of the control panel. This is like the main circuit breaker switch: when it is off, all other switches are off. Then select the desired graph types from the "REGPSO & PSO GRAPHING SWITCHES" subsection. Two of the more important graph types are emphasized below. If you have questions about another graph type, please post a comment or email me.

Mean Function Value

One of the most important switches is *OnOff_graph_fg_mean*, which graphs the mean function value of global best versus the iteration number.

Swarm Trajectory

Switch *OnOff_swarm_trajectory* allows the user to watch the swarm move across a contour map of the objective(s) selected. This feature is for illustrative purposes and works only with two-dimensional optimization problems: i.e. problems with two inputs or decision variables to be optimized – constituting the x and y axes – and a scalar output – constituting what would be the z axis – by which to evaluate the quality of the decision vector. Rather than appearing as a three-dimensional image, a colored contour map plots each function value of the z axis using a color scale in which dark blue is assigned to the lower values, dark red to the upper values, and a continuous color spectrum is used for values in between – resulting in a flat, two-dimensional, colored image on which particles can more easily be seen traversing the search space.

When switch *OnOff_contourf* is active, a smooth continuum of colors will be generated. Smoothness (i.e. accuracy of the graph) increases with the value of *GraphParams_contour_map_quality* (recommended values 50 – 450 depending on whether particle movement is simply being observed or high-quality figures are desired for presentation). When *OnOff_contourf* is inactive, contour lines will be generated with a number on each line reflecting the value of the function at that level rather than using the more aesthetically pleasing colored contour map.

Switch *OnOff_mark_personal_bests* marks the location of each particle's personal best. Switch *OnOff_mark_global_best_always* marks the location of the global best on each contour map generated, whereas *OnOff_mark_global_best_on_zoomed_graph* marks it only when zooming on the final state of the swarm at the end of each RegPSO grouping – as controlled via switch *OnOff_zoom_on_final_graph.*

Particles' index numbers can be turned on and off via switch *OnOff_plot_indices* in the "REGPSO & PSO GRAPHING SETTINGS" subsection of "(5) REGPSO & PSO GRAPHING SWITCHES AND SETTINGS".

It is recommended to set *GraphParams_swarm_traj_snapshot_mode* to 4 since the modes were developed in chronological order with each being seen as an improvement upon the previous. This uses *contmap4factor* to generate a contour map slightly larger than the initialization space since particles occasionally leave it when not restrained using velocity reset.

Saving Data Automatically

Activate switch *OnOff_Autosave_Workspace_Per_Trial* in section "(1) BASIC SWITCHES & PSO ALGORITHM SELECTION" to save the workspace after each automatic trial. Activate switch *OnOff_Autosave_Workspace_Per_Column* to save the workspace after each set of trials. If disk space is a concern, activate switch *OnOff_Autodelete_Trial_Data* to automatically delete files to which single trial data had been saved once all such data has been reconstructed for statistical purposes at the completion of all trials, at which point reconstructed data will be saved in the workspace of each column. For the full version of the toolbox, you may also save the workspace after each table generated using switch *OnOff_Autosave_Workspace_Per_Table*.

When RegPSO is active, you can save the workspace per grouping via switch *OnOff_Autosave_Workspace_Per_Grouping*.

Examining Saved Data

Saved data begins with “RM0” for standard PSO without regrouping and “RM1” for RegPSO, which was designed to address the stagnation problem of PSO [2].

Workspaces saved after individual trials are distinguishable from those saved after columns and tables via the designation of the trial number as “Tr#: immediately following the “RM#” discussed above. Filename “RM0Tr3…” would designate the workspace saved after the third trial generated using regrouping method 0 (i.e. standard PSO). “RM1C…” would designate the workspace saved after generating a column (i.e. set of trials) pertaining to RegPSO.

Filenames are designed to transparently display the parameters used to produce the data contained within the file so the user does not have to load a file and examine the workspace to locate the specific file for which he or she is looking. Filenames also have the date and time the data was generated appended at the end, which ensures that each filename generated is unique and helps the user locate the file he or she is looking for based on when it was generated.

For column and table data, values displayed in filenames are prefaced with meaningful designators such as "w" for the inertia weight and "ca" and "cb" for the acceleration coefficients. Filenames for data saved after individual trials eliminate these designators to make room for the globally best function value produced per trial, which can quickly indicate whether an algorithm is working or not by watching the folder to which the data is being saved. With time and familiarity, having data readily accessible at a glance becomes convenient, though it might seem inconvenient at first. For example, if the user sees in the trials' filenames that the function values being produced are ridiculous, he or she can immediately quit and debug rather than wasting time.

Excessively long filenames cause files to become inaccessible on Windows systems unless and until the user is able to shorten the name, which would be quite an inconvenience as a regular requirement; consequently, filenames are densely populated with relevant information without lengthy descriptions as to what those values refer. This results in a system that requires some time to acquire. While filenames are generated in a utilitarian rather than aesthetic manner, one can always load the file and examine the workspace directly if doing so seems more convenient than decoding information in the filenames.

Measures of Efficiency: Iterations, Function Evaluations, and Real Time

According to Merriam Webster's Collegiate Dictionary Eleventh Edition, an "iteration" is "the repetition of a sequence of computer instructions a specified number of times or until a condition is met," which literally does not include initialization of the swarm prior to the repeated update equations. "Function evaluation" is therefore a more transparent terminology in most cases since all evaluations of positions are counted – both at initialization and within the iterative loop. This includes evaluations of additional positions resulting from any perturbation of the global best and evaluations of any additional positions introduced by certain variations of PSO – added complexity not reflected by a simple count of iterations. Function evaluations are consequently generally preferred over iterations except possibly in graphs, where they could be misleading since MATLAB evaluates positions one matrix at a time; consequently, the Particle Swarm Optimization Research Toolbox never uses function evaluations on the independent axes of graphs. The user may choose whether to specify a maximum number of function evaluations or iterations as a termination criterion.

To agree with the usual usage of "iteration," initialization of the swarm is considered to occur at iteration one even though initialization does not make use of the iterative updating process and consequently is not literally an "iteration," as defined in the first paragraph of this section; consequently, the first velocity and position updates occur at "iteration" two. Even though initialization of the swarm is not literally an "iteration," it is counted toward the maximum number of iterations for termination purposes. To avoid ambiguity, and to more accurately compare algorithms of different complexities, it is recommended to use function evaluations as a general habit in discussion, though iterations are preferable for graphs due to the iterative nature of optimization algorithms and the matrix updates implemented in MATLAB.

While function evaluations are generally preferable to iterations except in graphs, bear in mind that an algorithm requiring more function evaluations than another is not necessarily less efficient in real time, which is the ultimate measure of an algorithm's efficiency. There are at least two reasons for this: (i) the complexity required to carry out a function evaluation varies across algorithms (e.g. PSO is computationally simple and operates more quickly in real time than some other algorithms); and (ii) an algorithm using a smaller population size than another to produce the same number of function evaluations will require more iterations, which in turn requires that more code be executed; assuming a uniform quality of code between algorithms, executing more code would consume more real time. Hence, while function evaluations are generally preferable to iterations, they are still imperfect measures. And while real time provides the ultimate measure of an algorithm's efficiency, comparing real-time performance across platforms is impractical.

If there are other topics you would like to see discussed in more detail, please email george [at] georgeevers.org with your request.

IV. Guide to Conducting Your Own Research

This chapter will teach you how to apply the Particle Swarm Optimization Research Toolbox to meet your own research needs – whether for solving new problems, improving the particle swarm algorithm, or both. To facilitate pasting code into MATLAB, the .doc version of this documentation can be downloaded from www.georgeevers.org/pso_research_toolbox_documentation.doc .

Adding Constraints

Hu & Eberhart's Approach

One simple approach for adding constraints to the particle swarm algorithm was presented by Hu and Eberhart in 2002 [5]. Particles are re-initialized until satisfying all constraints, and personal and global bests are updated only with feasible positions. This ensures that all starting points and all points of attraction are feasible without adding much complexity to an otherwise computationally simple algorithm.

This approach can be implemented via the following steps.

(1) Create switch *OnOff_Constraints* in the control panel's first section so that constraints will be implemented only when the user desires.

(2) Create a new function, "Constraints_satisfied.m," to return "true" for each position/row of the input matrix that satisfies all constraints and "false" for each other row. The inputs should be *position_matrix* and *num_particles_2_evaluate*, and the output should be a column vector (e.g. *Satisfied*) of logical 1's and 0's.

(3) Create a new function, "Constraints.m," to correct any violations within position matrix *x* during swarm initialization. The inputs should be *position_matrix* and *num_particles_2_evaluate*, and the output should be the corrected position matrix, *x*. "Constraints.m" should make use of function "Constraints_satisfied.m," which will also be used in step 5.

(4) Paste the code in (a) below to the locations specified in (b).

(a)
```
if OnOff_Constraints
        Constraints(x, np)
end
```

(b) after "*x*(1:np, 1:dim) =" within:
   - Reg_Methods_0And1.m
   - Reg_Method_1_regrouping.m

(5) Paste the code in (a) below to the locations specified in (b).

(a)
```
if OnOff_Constraints
        Constraints_satisfied(x, np)
end
```

(b) after "*x* = *x* + *v*" within: s
   - gbest_core_loop.m
   - lbest_core_loop.m

(c) after "*x = OPSO_selections.*ox + (1 - OPSO_selections).*x*" within:
   - gbest_initialization.m

- lbest_initialization.m

(6) Replace the code in (a) below with the code in (b) at each location specified in (c) so that personal bests will only be updated with positions that satisfy the constraints.
   (a) `if f(Internal_i) <= fp(Internal_i)`
   (b) `if (Satisfied(Internal_i) && (f(Internal_i) <= fp(Internal_i)))`
   (d) within:
      - gbest_core_loop.m
      - lbest_core_loop.m

(7) Replace the code in (a) below with the code in (b) at each location specified in (c) so that global bests will only be updated with positions that satisfy the constraints.
   (a) `fg = min(fg, min(f))`
   (b) `fg = min(fg, min(fp))`
   (c) within:
      - gbest_core_loop.m
      - lbest_core_loop.m

An Alternative Idea

If you would like to ensure that all positions encountered during the search satisfy the constraints, you could implement the following approach; however, it has not been demonstrated to outperform the approach above (i.e. it has not even been tested yet).

(1) Create switch *OnOff_Constraints* in the control panel's first section so that constraints will only be implemented when the user desires.

(2) Create a new function, "Constraints.m," to correct any violations found within position matrix *x* during swarm initialization. The inputs should be *position_matrix* and *num_particles_2_evaluate*, and the output should be the corrected position matrix, *x*.

(3) Paste the code below to the locations specified in (a) and (b).

```
if OnOff_Constraints
      Constraints(x, num_particles_2_evaluate, [and possibly v])
end
```

   (a) after "*x* = *x* + *v*" within:
      - gbest_core_loop.m
      - lbest_core_loop.m
   (b) after "*x*(1:np, 1:dim) =" within:
      - Reg_Methods_0And1.m
      - Reg_Method_1_regrouping.m

This alternative approach corrects all new positions that violate constraints; however, failure to correct velocities responsible for pushing particles out of bounds could result in repeated violations; therefore, a thorough implementation of this approach would account for positions firstly and velocities secondly, both of which could be corrected by "Constraints.m".

Enhancing Velocity Updates

If your research involves enhancing the velocity update equation, you can modify it as follows.

(1) Create switch, *OnOff_[YourSwitch],* in the first section of "Control_Panel.m".

(2) Paste the code below over the codes specified in (a) and (b).

```
if OnOff_YourSwitch
    [Insert your velocity update equation here.]
else %i.e. standard velocity update equation
    v = w*v + c1*r1.*(p - x)+c2*r2.*([g or l] - x)
end
```

(a) "*v = w*v + c1*r1.*(p - x)+c2*r2.*(g - x)*" within "gbest_core_loop.m" and
(b) "*v = w*v + c1*r1.*(p - x)+c2*r2.*(l - x)*" within "lbest_core_loop.m"

Enhancing Position Updates

If your research enhances the position update equation, you can modify it as follows.

(1) Create switch, *OnOff_YourSwitch,* in the first section of "Control_Panel.m".
(2) Replace "*x = x + v*" in "gbest_core_loop.m" and "lbest_core_loop.m" with the following:

```
if OnOff_YourSwitch
    [Insert your position update equation here.]
else %i.e. standard position update equation
    x = x + v
end
```

Varying the Inertia Weight Non-linearly

Control_Panel.m

If your research involves non-linear variation of the inertia weight, the first step is to create new switch *OnOff_w_nonlinear_variation* in "Control_Panel.m". For sake of organization, please create the new switch below *OnOff_w_linear*.

Then change

```
if OnOff_w_linear %i.e. if inertia weight is time-varying
    w_i = 0.9; %initial value used in first velocity update
    w_f = 0.4; %final value used in final velocity update
else %i.e. if a static inertia weight is desired
    w = 0.72984;
end
```

in section "(2) PARTICLE SWARM ALGORITHM SETTINGS" to

```
if OnOff_w_linear || OnOff_w_nonlinear_variation %i.e. if inertia
        %weight is time-varying
    w_i = 0.9; %initial value used in first velocity update
    w_f = 0.4; %final value used in final velocity update
else %i.e. if a static inertia weight is desired
    w = 0.72984;
end
```

to allow the user to specify the initial and final values without creating new programming variables unnecessarily.

gbest_core_loop.m & lbest_core_loop.m

After "if *OnOff_w_linear*" in these two files, add an "elseif *OnOff_w_nonlinear_variation*" to implement the formula by which *w* is to be updated.

Input_Validation.m

Since linear and non-linear variation cannot occur simultaneously, please add a check to "Input_Validation.m" such as the following. In addition to being a courtesy to the user, this explicit check will prevent unexpected behavior.

```
if OnOff_w_nonlinear_variation && OnOff_w_linear
    error(['Since the inertia weight cannot be varied both '...
        'linearly and non-linearly, please de-activate either '...
        '"OnOff_w_nonlinear_variation" or "OnOff_w_linear" in the '...
        '"PSO ALGORITHM SELECTION" subsection of the control '...
        'panel's section "(1) BASIC SWITCHES & RELATED ALGORITHMIC '...
        'SETTINGS".'])
end
```

Display_Settings.m

At each occurrence of "if *OnOff_w_linear*" within "Display_Settings.m," add a corresponding "elseif *OnOff_w_nonlinear_variation*" to display an appropriate message during the User Input Validation phase of program execution.

Title_Graphs.m

Below each occurrence of "if *OnOff_w_linear*" within "Title Graphs.m," add an "elseif *OnOff_w_nonlinear_variation*" to display the same title under "if *OnOff_w_linear*" except with the "Linearly Varying Inertia" entry changed to your liking.

Adding Your Problem to the Toolbox

Writing Your Function

Naming Convention

If you are using the Particle Swarm Optimization Research Toolbox to solve an application problem or would like to add a new benchmark to the suite, firstly write the new objective into the toolbox as a function. Feel free to begin with a file such as "ObjFun_Ackley.m" as a template, but remember to save the function as a new file. The first line of this new file is shown below, where "Name" is used generically.

```
function [f] = ObjFun_Name(position_matrix, num_particles_2_evaluate)
```

Each objective's name begins with "ObjFun_" to organize objectives together when viewed by name in their containing folder. The filename should be the same as the function name: in this case "ObjFun_Name.m". Regardless of the name, the function will be assigned function handle *ObjFun* during program execution, which allows the toolbox to use one generic name for all objectives as in the code pasted below from "gbest_core_loop.m".

```
f = ObjFun(x, np)
```

Inputs to be Accepted

Since generic function handle *ObjFun* is used internally, all objectives must accept the same inputs: (i) a matrix of positions, and (ii) the number of positions/rows to be evaluated. In the example line pasted above, the full position matrix, *x,* is input for evaluation; the second input argument specifies that all *np* rows of the input matrix are to be evaluated (i.e. the full swarm size). Though “size(position_matrix, 1)” could be calculated iteratively within the function rather than passing in the number of rows to be evaluated, the approach utilized lends itself to efficient computing by eliminating the need to repeatedly calculate knowledge already possessed by the toolbox.

When evaluating a single position vector, pass in a 1 for the number of rows to be evaluated as shown in the code below, which evaluates the quality of a mutated global best, *gm*, within “gbest_core_loop.m” when switch *OnOff_Cauchy_mutation_of_global_best* is active.

```
fm = ObjFun(gm, 1)
```

Global Variables

Some objectives also make use of the problem dimensionality, *dim*, which is the number of columns in the input matrix/vector. For this reason, *dim* is declared global in the control panel and is accessible by any function that also declares it global, which eliminates the inefficient need to iteratively calculate “size(position_matrix, 2)” within the function. The Ackley function, “ObjFun_Ackley,” is one example of a function that makes use of global variable *dim*.

If you need to access any other workspace variable within your function, you can simply define it to be global: (i) where it is created by the toolbox, and (ii) within the new function. For example, *dim* is declared global in the control panel immediately prior to its creation so that it can be accessed by “ObjFun_Ackley.m,” which also declares it global; the same can be done for any other workspace variable and function to grant immediate access to variables in the workspace that do not change when the function is called.

Listing Your Function in Objectives.m

Once you have added your problem to the Particle Swarm Optimization Research Toolbox, the next steps are: (i) assign it an objective ID, and (ii) specify a reasonable initialization space within “Objectives.m,” which lists all objectives in alphabetical order. Please find the alphabetical location for your objective and paste the code below into “Objectives.m”.

```
elseif objective_id == 12
   objective = 'NameYourObjectiveHere';
   if OnOff_asymmetric_initialization
      range_IS0 = 2.56; %asymmetric [2.56, 5.12]
      center_IS0 = 3.84; %i.e. (lower bound + upper bound)/2
   else
      range_IS0 = 2*5.12; %[-5.12, 5.12]
```

```
    center_IS0 = 0; %Specify the initial center_IS of the initialization space (null vector
                    %default for each particle).
  end
  if OnOff_SuccessfulUnsuccessful
    if dim == 30
      thresh_for_succ = 0.01;
    elseif dim ==2
      thresh_for_succ = 10^(-7);
    end
  end
```

Name

After pasting the code above into “Objectives.m,” Replace 'NameYourObjective' with the name your objective as you would like it to be displayed by the toolbox (i.e. without “ObjFun_” at the beginning).

Initialization Space

Specify the center and range of the initialization space. If each dimension to be optimized has the same range of potential values (e.g. as is the case with simple benchmarks), then the range and center of the initialization space can be input as scalars, which tells the toolbox to use the same values per dimension. However, if different dimensions represent different variables to be optimized – each of which has its own range of values – a different range of values can be specified per dimension by designating the range and center of the initialization space as vectors; an example problem of this type is given in chapter “III: More Details on Select Topics” in section “Setting the Center and Range of the Initialization Space”.

Threshold for Success

Specify a threshold of success for each problem dimensionality of interest. The higher the problem dimensionality, the more difficult the problem will become. For this reason, the threshold for success should increase with dimensionality while accounting for the difficulty of the particular problem. The threshold for success merely determines the subjective function value at which a trial should be considered successful; yet despite its subjective nature, *thresh_for_succ* complements the generated statistics by showing the number of trials performing worse than the value of interest, which gives an idea of how such outlying trials affect the mean, standard deviation, and median.

Check for Redundant Id Assignment

Lastly, check that no other problem is assigned the same objective id within “Objectives.m,” and set *objective_id* at the end of the control panel to the same value assigned to your problem within “Objectives.m”.

Other Enhancements

If your research involves more than just adding constraints, modifying the velocity update equation, modifying the position update equation, adding your problem to the Particle Swarm

Optimization Research Toolbox, and/or training ANN's, you are welcome to email me for guidance. I will be happy to provide guiding hints as time permits to help you add the needed functionality to the Particle Swarm Optimization Research Toolbox as done at this MathWorks thread.

Benefiting from Updates

Integrating improvements into the PSO Research Toolbox will allow the contributor to benefit from future fixes and enhancements made to the toolbox. Simply email me any improved code with detailed comments describing its functionality, and I will add it to the toolbox as time permits. The user community would appreciate your contribution, and formal acknowledgement will be made.

V. Contribution Standards

Following Naming Conventions When Creating New Variables

- As much as possible, please name **new programming variables** so that they **will be grouped together when "whos" is typed at the command prompt** (e.g. switches begin with "OnOff_," graph settings begin with "GraphParams_," table settings begin with "TableParams_," RegPSO settings begin with "RegPSO_," GCPSO settings begin with "GCPSO_," and multi-start settings begin with "MPSO_".
- Since capitalized names display before lower case names when "whos" is typed at the command prompt, capitalized names are pushed off the screen; hence **switches, internal variables**, and the like should **begin with capital letters,** while **outputs** and settings to frequently referenced (e.g. *np, dim, w*) should **begin with lower-case letters** to be more easily accessible.
- **Please use transparent names** for all programming variables to help users easily make sense of the workspace and code**.** For example, after writing and testing the code to calculate the number and percentage of trials successful, I replaced all occurrences of "Ns" with *num_trials_successful* and "Nu" with *num_trials_unsuccessful*. This grouped them in the workspace with the total number of trials, *num_trials*, and made their meaning obvious to all users.
- Before creating new variables, please: (i) press Alt + {e, i} within MATLAB to "Find Files," (ii) type the name you are thinking of assigning to the new variable into the "Find files containing text" field, (iii) for the "Look in" field "Browse" to the folder containing the most recently updated version of the Particle Swarm Optimization Research Toolbox, (iv) change the "search type" at the lower left to "Matches whole word," and (v) press Enter. If no results are displayed, the variable name is available for use.

Minimizing Hard Coding via "Control_Panel.m"

- **Please create a switch for new code so that it will run only when activated.**
  - Please name switches "OnOff_[Relevantly_Named_Switch]" using transparent names.
  - Please activate switches using syntax "OnOff_[Relevantly_Named_Switch] = logical(1)" and de-activate by setting "OnOff_[Relevantly_Named_Switch] = logical(0)" to enable

the user to activate or de-activate switches simply by changing 1 to 0 or vice versa. This is more efficient than changing "true" to "false" or vice versa.

- Please describe the switch's functionality with a comment.
- Please add your name and email address after switches you create so that future users will know who to contact with questions (e.g. OnOff_[YourSwitch]= logical(1) %JonDoe@Jon'sUniversity.edu, Jon Doe, 2009).

- **If you find yourself regularly changing a hard-coded value, try creating a variable whose value can be set in "Control_Panel.m*".*** For example, in addition to saving figures to .fig format to facilitate modification within MATLAB prior to publication, it became desirable to eliminate the need to manually save figures to other formats. To specify the desired format without hard coding it, parameter *GraphParams_autosave_format* was introduced in the control panel's section "Graphing Switches & Parameters". To avoid workspace clutter, it is only created when switch *OnOff_autosave_figures_to_another_format* is active.
- **Please take care to place new programming variables in the appropriate sections** of "Control_Panel.m". The control panel is divided into the following sections: (i) Basic Switches & PSO Algorithm Selection, (ii) Particle Swarm Algorithm Settings, (iii) RegPSO Algorithm Switches & Settings, (iv) Table Switches & Settings, (v) RegPSO and PSO Graphing Switches & Settings, and (vi) Objectives.

  Most of the settings pertaining to the objective function are set within "Objectives.m," which is called from section "(6) OBJECTIVES" of "Control_Panel.m". This script is kept separate from "Control_Panel.m" since it is re-used when objectives are automatically incremented by the professional version of the toolbox.

Validating Inputs via "Input_Validation.m"

"Input_Validation.m" is called to check the validity and consistency of user-specified switches and settings. While debugging new code, you may encounter unintended parameter combinations that lead to errors or unexpected behavior. **When a contradictory combination of switches and settings in "*Control_Panel.m*" produces or could conceivably produce unexpected behavior, please write a check within "Input_Validation.m"** to either (i) prevent execution altogether and display a message directing the user to correct the conflicting inputs, or (ii) correct the flawed parameter automatically by inferring the user's intention and display a message stating what change was made. An example of the second case is displayed below.

```
if OnOff_graph_f && ~OnOff_fhist %If the function value of each
    %particle is to be graphed (instead of only the function value
    %of the global best),
  OnOff_fhist = logical(1); %matrix "fhist," which maintains in each
     %row the function value of the corresponding particle, must be
     %maintained.
  disp(['Because you have activated "OnOff_graph_f," which graphs'...
    ' the function value versus iteration for every '...
    'single particle, matrix "fhist" must be maintained; '...
    'therefore, "OnOff_fhist" has been activated for your convenience.'])
```

```
        fprintf('\r') %Insert a carriage return before future outputs.
    end
```

Steps for Adding an Automatic Correction to “Input_Validation.m”

- Locate the proper location for the “if statement”. For example, graphing switches are only created in the workspace when parent switch *OnOff_graphs* is active. So the example code above appears after that parent switch has been checked since the attempt to check the status of non-existent switches would terminate execution with an error.
- Generate a message appropriate to the severity of the conflict.
  - Use the “disp” command to merely display a caution of which the user should be aware. The code above exemplifies this usage.
  - To indicate a more urgent message, “warning” can be used instead of “disp”. This just adds “Warning:” before the message to be displayed.
  - Use “error” to direct the user to conflicting inputs and prevent execution, which is appropriate when the conflicting inputs are severe enough to require correction before execution. This would be used when the user’s intentions cannot accurately be inferred as they can be in the example code pasted above.
- To display a detailed message, please use ellipses (i.e. “…”) as done in the example code pasted above to indicate to MATLAB that the message continues on the following line. This creates a message that wraps nicely within the user’s Command Window regardless of the window’s size.
- After each “disp(…)” or “warning(…),” please type "fprintf('\r')" to add a carriage return so that any other messages will not be run together.

Displaying the Most Important Settings via “Display_Settings.m”

For each new algorithm or enhancement, please display the most important settings for user verification in script “Display_Settings.m,” which is called after “Input_Validation.m”. For example, when RegPSO is activated via switch *OnOff_RegPSO* in the control panel’s first section, the following lines of “Display_Settings.m” will display relevant settings for user verification before program execution. Because the user is given direct control of so many settings in the control panel, it is useful to display the most important settings for verification prior to execution.

```
if Reg_Method == 1
   disp('Regrouping PSO (RegPSO) utilizing')
elseif Reg_Method == 2
   disp('Regrouping Method 2 utilizing') %not as reliable as RegPSO: code
      %not yet perfected
end

...

if OnOff_func_evals
   if Reg_Method ~= 0 %i.e. a regrouping PSO will be used
```

```
            disp([num2str(max_FEs_per_grouping), ' FE''s maximum '...
                '(per grouping)'])
            disp([num2str(np*ceil(max_FEs_over_all_groupings/np))...
                ' FE''s maximum (total over all groupings)'])
        else %i.e. Reg_Method == 0 selects standard PSO
            disp([num2str(max_FEs_per_grouping), ' FE''s maximum'])
        end
    else %i.e. if iterations are to be used instead of function evaluations
        if Reg_Method ~= 0  %i.e. a regrouping PSO will be used
            disp([num2str(max_iter_per_grouping), ' iterations '...
                'maximum (per grouping)'])
            disp([num2str(max_iter_over_all_groupings), ' iterations '...
                'maximum (total over all groupings)'])
        else %(i.e. Reg_Method == 0 for the standard case without
                %regrouping)
            disp([num2str(max_iter_per_grouping), ' iterations maximum'])
        end
    end
```

Maintaining Workspace Integrity

"gbest_core_loop.m" and "lbest_core_loop.m" were written as scripts after tests showed scripts to be more efficient than functions regardless of whether variables were passed in or declared as globals. The common workspace is convenient and efficient, but the user should take care to maintain an organized workspace. It is better for an error to terminate execution because a variable has been cleared than to allow for the possibility of a variable being used unexpectedly: if in doubt, clear the variable. And always clear the variable after its final usage unless you want it to remain in the workspace for eventual analysis.

Steps to Avoid Unintentional Usage of Data

- **Variables should be cleared when they are no longer needed** to (i) maintain an organized workspace, (ii) reduce both the size of the automatically saved workspaces and the time required to save them, and (iii) eliminate the possibility of an incorrect value being used for another purpose should the same basic variable name accidentally be use by another contributor. This can be done by typing "clear" followed by the name of the variables to be cleared. To clear a global variable, type "clear global" instead of "clear" and declare the variable global again on the following line.
- **Variables should be re-initialized where appropriate (e.g. with each new table, column, regrouping, or automatic restart of the algorithm)** to ensure that each variable initializes with a fresh slate with each new execution. For example, the variables pertaining to each trial are initialized within "Trial_Initializations.m", and variables pertaining to RegPSO are initialized within "RegPSO_main.m" under heading "`%Initialize "RegPSO_main" Variables%`".
- List any new variables that you have created in the Definitions of Appendix A so that other users will understand their purpose. This will help other contributors avoid using the same

name for another purpose so that your functionality continues working as planned. Simply email me any edits to be added to the documentation.

Testing New Code Independently of the PSO Toolbox

It is important to avoid overconfidence in one's ability to write flawless code. New blocks of code should be tested independently of the overall toolbox to ensure that they work as expected since slight deviations from expected behavior might go unnoticed in overall performance. One healthy approach is to write new code in a new file; whether as a function or as a script, it will be easier to debug this way. A test file can then be used to (i) pass test values for all expected variables into a function and display the outputs, or (ii) create the expected variables in the workspace if a script is being tested instead of a function. When testing a script, this test file should state "clear all" near the top to ensure that previously created values do not pollute the integrity of your tests. Saving these test files and organizing them well will help when testing future modifications.

If the new code applies to the iterative portion of the toolbox (e.g. "gbest_core_loop.m" or "lbest_core_loop.m," it may be more efficient to paste the code directly into the calling script once it has been verified to work correctly; however, this should be not done until the code has been thoroughly tested.

Modifying all Relevant Gbest & Lbest Scripts

For clarity of thought, separate scripts are maintained for the *Gbest* and *Lbest* variants of PSO. Some contributions may not apply to all scripts – for example, any perturbation or other treatment of the global best is irrelevant to *Lbest* PSO, which utilizes a local or neighborhood best rather than a global best. After independently testing contributions to iterative portions of the code, please be sure to apply them to all of the following files as relevant:

1. gbest_core.m,
2. lbest_core.m,
3. gbest_initialization.m,
4. lbest_initialization.m
5. gbest_core_loop.m, and
6. lbest_core_loop.m.

Commenting After "else"

When using an "if, else" sequence, please paste a comment after each "else" clarifying to which "if" statement it corresponds. This improves the efficiency of editing since one can more easily navigate to the relevant section to be edited without needing to refer back to the corresponding "if" statement.

For example, at the time of this entry, "gbest_core.m" contained the following:

```
Line 30:        if OnOff_MPSO
                ...
Line 140:       else %i.e. if ~OnOff_MPSO
```

```
                ...
Line 236:       end
```

In the Editor, hovering the cursor over “if” on line 30 shows a clickable link at the bottom left of the Editor to the corresponding “end” on line 236. Similarly, hovering the mouse over “else” or “end” produces a clickable text box at the top left of the Editor displaying the line number of the beginning “if” statement; however, the line number corresponding to “else” is nowhere shown, so that its location can only be estimated to lie about half way between “if” and “end”. Consequently, commenting “else” lines helps keep code organized and readable, which is beneficial during editing. Hopefully, The MathWorks will expand the display boxes in the future to list all relevant “elseif” and “else” lines rather than just the beginning and ending lines, though the current functionality is definitely a good start.

Using Short-Circuit Logical Operators

In logical checks, please use “&&” for “and” and “||” for “or” since these check the second expression only when the first expression does not itself determine the result. For example

```
if (something == 1) || (something_else == 0)
    disp(['There is no need to check the second expression since '...
        'the first evaluates to TRUE.  To maximize efficiency, list '...
              'first the condition most likely to be TRUE.'])
end
if (something == 0) && (something_else == 1)
    disp(['There is no need to check the second expression since'...
        'the first evaluates to FALSE. To maximize efficiency, list '...
              'first the condition most likely to be FALSE.'])
end
```

MATLAB allows use of “&” and “|,” but these are generally expected to be less efficient than “&&” and “||” since they check subsequent expressions regardless of the status of the first.

Adding New Algorithms to Graph Titles

After contributing new functionality to the toolbox, please add a relevant title to the auto-titling code as follows:

(1) Paste at the beginning of “Title_Graphs.m” the text below using the actual name of a new switch:
```
if OnOff_NewSwitch
else %i.e. ~ OnOff_NewSwitch
end
```
(2) Highlight the existing titling code,
(3) Press Ctrl +] to right indent it,
(4) Press Ctrl + X to cut it,
(5) Click after the new “else” line and paste the code that you just cut,

(6) Click after the new “if” line and paste the code again,
(7) Make a note of the line the cursor is currently on,
(8) Press Ctrl + Home,
(9) Highlight the series of characters common to each title that you would like to modify: such as “' Dimensions;'],”
(10) Press Ctrl + H,
(11) Type the text with which to replace the common characters – such as “' Dimensions;'], 'My New Algorithm Is Enabled!',”
(12) Place the cursor over the Replace button and tap until reaching the line number noted in step (7) (don’t worry, you can press Ctrl + Z to undo if you happen to go too far: hopefully, MathWorks will introduce the ability to replace all occurrences between specified line numbers, but for now the tapping is only a minor inconvenience), and
(13) Press Ctrl + S to save the auto-titling code with your new title included.

Avoiding Pre-Allocated Zeros

MATLAB suggests pre-allocating the dimensions of vectors and matrices rather than having them grow within a loop. While this would be possible when PSO is to be executed for a specific number of iterations, the exact number of iterations is not known in advance for all histories in all cases. As examples, (i) RegPSO efficiently regroups the swarm when stagnation is detected rather than simply executing a pre-specified number of iterations, and (ii) Opposition-Based PSO (OPSO) randomly selects other positions to be evaluated so that if a maximum number of function evaluations is used as a termination criterion, there is no direct translation to the number of iterations to be conducted.

Additionally, examining the structures of histories at completion to be sure they are as expected is a useful debugging practice that offers extra insight into whether the program has functioned as expected.

Furthermore, pre-allocating a matrix or vector with zeros could be dangerous for an optimization toolbox since zero is the optimal function value for many objectives, and the null vector is the ideal decision vector; hence, a bug in the program could erroneously interpret a pre-allocated zero – not overwritten as expected – to mean that perfect success was achieved when it actually was not. Consequently, rather than risking the pollution of data with values not actually generated by the algorithm – especially zeros – this toolbox saves data to workspaces during execution and reconstructs it to generate statistics. When RegPSO is enabled, this prevents matrix histories from becoming excessively long so that growing within a loop is not a major problem. Currently, the toolbox can automatically save and reconstruct the data of each trial and of each grouping within each trial (i.e. the latter for RegPSO only).

For vectors whose final lengths are not determinable in advance, the length could be incremented by a value such as 200 each time an increased length becomes necessary instead of iteratively increasing the length by 1 each time a new value is stored. However, when this approach is compared to saving data per 200 iterations and reconstructing the full vector at the end of the trial, automatic saving and reconstruction is the clear winner since it clears RAM at each checkpoint rather than searching for more contiguous RAM by which to increment a vector's length per 200 iterations. Due to the iterative nature of the Particle Swarm Optimization Research Toolbox, which requires that matrix and vector histories be written to rapidly, incrementing the length of a vector by any finite size when its pre-allocated length has been reached is less advantageous than simply saving the data to free memory and reconstructing the vector at the end of that trial. Therefore, to anyone considering improving the quality of the toolbox via pre-allocation of vectors whose lengths are not actually knowable in advance, please consider automatically saving the data every x iterations rather than automatically incrementing the vector by size x when its pre-allocated length is reached.

Note that "RegPSO_save_grouping_data.m" and "RegPSO_load_grouping_data.m", which are used to save and reconstruct data per grouping, could be improved to save standard PSO data every 200 iterations or so and reconstruct it at the end of each trial. This would further improve the speed of execution for lengthy trials while still: (i) allowing structures of histories to be examined for consistency with other values such as the number of iterations conducted, and (ii) preventing the potential pollution of data with pre-allocated values in the presence of an inadvertent bug. Because this improvement would apply both to algorithms whose primary vectors have predictable lengths as well as to those of unpredictable lengths, it is an all-encompassing improvement of higher value than pre-allocating the lengths of vectors having predictable lengths.

For relatively short vectors whose lengths are *always* knowable in advance (i.e. regardless of the algorithm employed), a good option is to pre-allocate using NaN's (i.e. "not a number") since NaN's prevent the possibility of data becoming polluted with values not generated by the toolbox. This was done for *iter_success,* which measures the iteration number at which each trial achieves success. Its length is exactly equal to the number of trials to be conducted as specified by the user in the control panel via setting *num_trials*. Notice that the use of NaN's in *iter_success* prevents the possibility of a miscalculated value being returned in *iter_success_mean*, which returns the average iteration number at which success was achieved – i.e. since no zero or other number can accidentally be applied to the mean in the event that success is not achieved. This reduces the probability of error and improves the integrity of the toolbox. Using NaN's for pre-allocation is safer than pre-allocating with numerical values since it acknowledges the possibility of mistaken code and prevents any such mistaken code from returning incorrect values, since data cannot become polluted with values that were not actually generated by the algorithm.

Defining New Switches & Programming Variables

In addition to clearly commenting new code, it is important to list new variables in Appendix A of this Product Documentation to enable others to easily: (i) efficiently reference each programming variable to understand its functionality, and (ii) avoid using the same name for new variables – a mistake that could cause some data to be overwritten and other data to be used for unintended purposes.

Applying Changes Throughout the Particle Swarm Research Toolbox

If you decide at some point **to improve the name of a new programming variable or file** contributed, you can highlight the text to be changed, press Alt + {E, I} to Find Files, browse to the directory to "Look in," and click "Find". This will show each containing file and the line number of each occurrence. You can then double-click the first occurrence of the variable in each file, press Ctrl + H to replace other occurrences within the same file, check the "whole word" box if applicable, type the new name of the variable or file to replace the old name, and press Alt + R to replace each occurrence found (or Alt + N to skip an occurrence without replacing it). Using Alt + R is safer than using Alt + A to replace all occurrences since there often exist occurrences that should not be replaced for one reason or another. The file is then efficiently closed by pressing "Esc," Ctrl + S to save the changes, and Ctrl + W to close the file.

This can also be used to hunt down and debug any malfunctioning programming variable or just fully understand how any variable is used.

Again, please use transparent names to help other users understand the purposes of files and programming variables. It is often convenient to initially write code using short names that are easy to type and replace them with more transparent names once the code is complete. (Mathworks does not currently offer the ability to automatically replace whole words within all files of a directory/toolbox, but I have received confirmation of an interest to develop it in future versions.)

VI. Program Flow

This is a basic overview and may be expanded upon request.

Open_Files.m

This is an optional file that opens files specified within it before selecting "Control_Panel.m" as the current tab. To use the file, right click "Open_Files.m" and click "Run," which opens all files except those commented out within the file.

"Open_All_Files.m" functions the same way except that no files are commented out within it so that it opens all files.

Control_Panel.m

All program settings are specified within the control panel except for the initialization space and threshold for success, which are specified in "Objectives.m".

Objectives.m

Once the switches and settings of "Control_Panel.m" have been loaded, the initialization space, threshold for success, and objective name are loaded for the *objective_id* specified at the end of the control panel.

Input_Validation.m

When conflicting switches or settings are detected, the Particle Swarm Optimization Research Toolbox attempts to infer the user's intentions and correct conflicts automatically. Notifications of corrections made are displayed during the "Automatic Input Validation" phase. When a conflict is detected but the user's intention cannot be inferred reliably, the user will be notified of the nature of the conflict and how to fix it rather than being left to infer as much from the error message that would otherwise be generated by MATLAB.

The more familiar you become with automatic changes, the more convenient they become. For example, when I activate RegPSO via switch *OnOff_RegPSO*, I don't bother activating *OnOff_NormR_stag_det* since I know it will be done for me automatically.

Display_Settings.m

When the automatic input validation phase completes, the most relevant switches and settings are displayed for verification, which allows you to detect before execution begins any setting that you may have forgotten to change.

RegPSO_main.m

After verifying the correctness of displayed settings, primary file "RegPSO_main.m" executes the following files according to the switches as they are set after the input validation phase.

Trial_Initializations.m

For each trial conducted, "RegPSO_main.m" initializes the histories and counters pertinent to each trial and resets the initialization space.

MPSO.m

If switch *OnOff_MPSO* is active, Van den Bergh's multi-start PSO will be executed by "MPSO.m," which repeatedly restarts "Reg_Methods_0And1.m" (i.e. and "Reg_Method_1.m" if *OnOff_RegPSO* is active) until termination criterion *MPSO_max_FEs*, *MPSO_max_iters,* or *MPSO_max_starts* is reached. If switch *OnOff_MPSO* is inactive, "MPSO.m" will not be executed.

Reg_Methods_0And1.m

Following initialization, each trial begins. For all PSO varieties, this code executes whichever of "gbest_core.m" or "lbest_core.m" below is relevant based on the status of switch *OnOff_lbest*.

gbest_core.m

This executes "gbest_initialization.m" to randomly initialize positions and velocities within the initialization space and repeats "gbest_core_loop.m" until any one of the following termination criteria is satisfied:

- *max_FEs_per_grouping* if *OnOff_func_evals* is active,
- *max_iter_per_grouping* if *OnOff_func_evals* is inactive,
- *MPSO_max_FEs* if *OnOff_func_evals* and *OnOff_MPSO* are active,

- *MPSO_max_iters* if *OnOff_func_evals* is inactive and *OnOff_MPSO* is active,
- *thresh_for_succ* if *OnOff_SuccessfulUnsuccessful* and *OnOff_Terminate_Upon_Success* are active,
- *true_global_minimum,* in which case continued computation would be useless.

lbest_core.m

This executes "lbest_initialization.m" to initialize the swarm and then repeats "lbest_core_loop.m" until a termination criterion specified for the grouping is satisfied as listed under "gbest_core.m" above.

Reg_Method_1.m

When RegPSO is activated via switch *OnOff_RegPSO* in subsection "PSO ALGORITHM SELECTION" of the control panel's first section, Van den Bergh's normalized swarm radius termination criterion detects the onset of premature convergence, and the swarm is regrouped to facilitate continued progress.

Data is stored to the activated histories, and "Reg_Method_1_regrouping.m" is repeated until satisfying one of the following termination criteria specified for the trial:

- *max_FEs_over_all_groupings* if *OnOff_func_evals* is active,
- *max_iter_over_all_groupings* if *OnOff_func_evals* is inactive,
- *MPSO_max_FEs* if *OnOff_func_evals* and *OnOff_MPSO* are active,
- *MPSO_max_iters* if *OnOff_func_evals* is inactive and *OnOff_MPSO* is active,
- *thresh_for_succ* if *OnOff_SuccessfulUnsuccessful* and *OnOff_Terminate_Upon_Success* are active,
- *true_global_minimum,* in which case continued computation would be useless.

Save_data_per_trial.m

When switch *OnOff_Autosave_Workspace_Per_Trial* is active, the workspace is automatically saved at the completion of each trial.

Load_trial_data_for_stats.m

When all trials have completed, each trial's workspace variables are loaded and analyzed to calculate statistics and mean function values for graphing purposes.

Standard_Output.m

The standard outputs of interest are displayed; these are in addition to any outputs activated by specialized switches.

Autosave_Workspace_Per_Column.m

When switch *OnOff_Autosave_Workspace_Per_Column* is active, the workspace is saved after each set of trials.

Graphs.m
When switch *OnOff_graphs* is active, user-specified graphs are generated.

VII. Disclaimer
Due to the magnitude of the Particle Swarm Optimization Research Toolbox and changes made since generating thesis data, it is possible that unknown bugs might exist. No guarantee is made that the code is free of flaw. It is your responsibility to validate the accuracy of data before publishing it or otherwise asserting its accuracy.

Neither MATLAB nor any other mathematical tool should be used blindly: e.g. (i) MATLAB's definition of the "mode" function produces mode([1, 500, 500.0001, 500.5, 501]) = 1, which is considered by some to be a misleading "measure of central tendency" that redefines "mode" as the most repeated *or* lowest number in a list rather than more stably reporting a NaN or treating every number as the mode since each "repeats" exactly zero times, and (ii) MATLAB's "ackleyfcn" function produces function values that are everywhere different than those produced by the traditional Ackley function – partly due to a closing parenthesis placed in an unorthodox position, which again results in a mathematical redefinition of a common function where the user expects no such a redefinition. This is not to somehow disparage MATLAB, which is a wonderful tool, but to illustrate clearly that no mathematical tool is free of flaw, and that all mathematical tools should be distrusted to a healthy degree. Similarly, the Particle Swarm Optimization Research Toolbox should not be used blindly or as a substitute for hard work; rather, it is meant to be contributed to and improved by other analytical and hard-working researchers.

The Particle Swarm Optimization Research Toolbox is thought to be a good starting point for your research, and it is hoped that you will contribute to it. George Evers does not guarantee the accuracy of data produced by the Particle Swarm Optimization Research Toolbox, though it is thought to be accurate and produces data comparable to that found in other papers when parameters are set identically.

VIII. Add-ins
Contributors to the PSO Research Toolbox Community-Based Development Project are motivated researchers who give back to the project.

NN Training Add-in
This add-in by Tricia Rambharose allows MATLAB's neural network toolbox to call the PSO Research Toolbox so that any of its included algorithms can be harnessed for training purposes: this is much more powerful than simply utilizing basic Gbest PSO for training purposes.

References
[1] G. Evers and M. Ben Ghalia, "Regrouping Particle Swarm Optimization: A New Global Optimization Algorithm with Improved Performance Consistency Across Objectives," *2009 IEEE International Conference on Systems, Man, and Cybernetics,* in press. (pdf)

[2] G. Evers, "An Automatic Regrouping Mechanism to Deal with Stagnation in Particle Swarm Optimization," M.S. thesis, The University of Texas – Pan American, Edinburg, TX, 2009 (pdf)

[3] F. van den Bergh, "An Analysis of Particle Swarm Optimizers," PhD thesis, Department of Computer Science, University of Pretoria, Pretoria, South Africa, 2002

[4] X. Hu and R. C. Eberhart, "Solving constrained nonlinear optimization problems with particle swarm optimization," in *Proceedings of the Sixth World Multiconference on Systemics, Cybernetics and Informatics (SCI 2002)*, Orlando, 2002.

[5] G. Venter and J. Sobieski, "Particle swarm optimization," in *Proceedings of the 43rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, Denver, CO, 2002.

Appendix A: Definitions

It might be helpful to print the definitions of workspace variables for quick reference while using the toolbox.

Of Programming/Workspace Variables

To request the definition of additional programming variables, or if any existing definition seems ambiguous or lacking, please email george [at] georgeevers.org.

center_IS

For "standard" PSO and the initial grouping of RegPSO, the center of the initialization space is *center_IS0,* which is set in "Objectives.m". Each new regrouping space is centered at the global best of the previous grouping, which is accomplished via line

```
        center_IS = g
```

within "Reg_Method_1_regrouping.m".

center_IS0

This is the initial center of the initialization space as set by the user in Objectives.m. It may be set as a scalar when all dimensions have the same center, or as a vector when any dimension has a unique center; an example of the latter is given in section Setting the Center and Range of the Initialization Space of chapter "III: More Details on Select Topics".

dim

This is the dimensionality of the problem being solved. It is static except in the professional version of the toolbox when auto-incrementing objectives, in which case the dimensionality is allowed to change with the objective if the user would like.

f

Column matrix *f* is generated by passing any number of rows from position matrix *x* (i.e. usually the number of particles, *np*) into a objective function and requesting the function value for those particles. For example, "ObjFun_Ackley(*x*, *np*)"
returns the Ackley function value of each particle in the swarm. Each row of *f* holds the function value of the position vector in the corresponding row of *x*. Typing "ObjFun_Rastrigin(x(3, :), 1)" would return only the function value of the third particle in position matrix *x*.

fg

Scalar workspace variable *fg* stores the function value of the global best and is updated any time a better global best is found. At the conclusion of the search, the variable stores the function value of the best position occupied by any particle at any time during the search.

fg_array

This stores the function value of each grouping's global best when Regrouping PSO (RegPSO) is used to help the swarm escape from the state of premature convergence to continue searching for better solutions. RegPSO is much more effective than simply restarting the search.

fghist

This row vector stores the function value of the global best through each iteration for one grouping of RegPSO or for one trial of standard PSO (i.e. since standard PSO does not regroup the swarm and consequently utilizes only one grouping). To enable the history, activate switch *OnOff_fghist* in the "PSO HISTORIES TO BE MAINTAINED" subsection at the end of the control panel's first section.

To access the best function value found through iteration 5, for example, type "fghist(5)" at the command prompt.

fghist_current_trial

When switch *OnOff_RegPSO* is active in the "PSO ALGORITHM SELECTION" subsection of the control panel's first section and switch *OnOff_fghist* is active in the "PSO HISTORIES TO BE MAINTAINED" subsection, the Particle Swarm Optimization Research Toolbox horizontally concatenates each grouping's history of the function value of the global best, *fghist*, to construct the history over all groupings conducted during the RegPSO trial. The best function value found through any iteration can be extracted from *fghist_current_trial* in the same fashion as from *fghist*.

fghist_mean

When switch *OnOff_fghist* is active in the "PSO HISTORIES TO BE MAINTAINED" subsection of the control panel's first section and *num_trials* is greater than 1 in section "(2) PARTICLE SWARM ALGORITHM SETTINGS", the Particle Swarm Optimization Research Toolbox computes the mean function value per iteration across all trials by averaging column-wise matrix *fghist_over_all_trials*. When switch *OnOff_graph_fg_mean* is active in section "(5) REGPSO & PSO GRAPHING SWITCHES AND SETTINGS," the graph of mean function value per iteration is constructed from row vector *fghist_mean.*

fghist_over_all_trials

When switch *OnOff_fghist* is active in the "PSO HISTORIES TO BE MAINTAINED" subsection of the control panel's first section and *num_trials* is greater than 1 in section "(2) PARTICLE SWARM ALGORITHM SETTINGS", the Particle Swarm Optimization Research Toolbox vertically concatenates each trial's row history of the function value of the global best, *fghist*. The first trial's history is stored to the first row, the second trial's history to the second row, etc. This allows the construction of *fghist_mean*, from which to construct the graph of mean function value per iteration.

fhist

This history of each iteration's function values is activated via switch *OnOff_fhist* at the end of the control panel's first section. Matrix *fhist* stores each iteration's column matrix of function values for one grouping of RegPSO or for one trial of standard PSO. Table 4 in the walk through shows how to extract the column matrix of any iteration's function values from the history.

fhist_current_trial

When switch *OnOff_RegPSO* is active in the "PSO ALGORITHM SELECTION" subsection of the control panel's first section and switch *OnOff_fhist* is active in the "PSO HISTORIES TO BE MAINTAINED" subsection, the Particle Swarm Optimization Research Toolbox horizontally concatenates each grouping's history of function values per particle, *fhist*, to construct the history of function values over all groupings conducted during the RegPSO trial. The matrix of function values corresponding to any iteration can be extracted from *fhist_current_trial* in the same fashion as from *fhist* in Table 4 of section "Analyzing Workspace Variables" in chapter "II: A Guided Walk Through".

fp

Column vector *fp* stores the function value of each particle's personal best and is updated iteratively.

fphist

This matrix stores the function values of the personal bests through each iteration for one grouping of RegPSO or for one trial of standard PSO (i.e. since standard PSO does not regroup the swarm and consequently utilizes only one grouping). To enable the history, activate switch *OnOff_fphist* in the "PSO HISTORIES TO BE MAINTAINED" subsection at the end of the control panel's first section. The column vector of function values of the personal bests through any iteration can be extracted from f*phist* as done for *fhist* in Table 4 of section "Analyzing Workspace Variables" in chapter "II: A Guided Walk Through".

fphist_current_trial

When switch *OnOff_RegPSO* is active in the "PSO ALGORITHM SELECTION" subsection of the control panel's first section and switch *OnOff_fphist* is active in the "PSO HISTORIES TO BE MAINTAINED" subsection, the Particle Swarm Optimization Research Toolbox horizontally concatenates each grouping's history of the function value of the personal bests, *fphist*, to construct the history over all groupings conducted during the RegPSO trial. The column vector of function values of the personal bests through any iteration can be extracted from f*phist_current_trial* as done for *fhist* in Table 4 of section "Analyzing Workspace Variables" in chapter "II: A Guided Walk Through".

g

This stores the global best of the swarm when: (i) Gbest PSO is executed by de-activating switch *OnOff_lbest*, or (ii) Lbest PSO is executed and switch *OnOff_ghist* is activated in the control panel to maintain a history of each iteration's global best for analysis (i.e. though the global best is not used in the update equations of Lbest PSO).

The global best is stored to each row of *g* to facilitate matrix subtraction in the velocity update equation of *Gbest PSO*; g(1, :) can be used to access the global best itself without repetition across all *np* rows of the matrix.

ghist

When switch *OnOff_ghist* is active, the global best is iteratively stored to the row of *ghist* corresponding to the iteration number per grouping of RegPSO or per trial of standard PSO (i.e. since standard PSO utilizes only one grouping). The top row stores the global best from initialization of the swarm at iteration 1, and the last row stores the global best of the final iteration of the grouping.

ghist_current_trial

When switch *OnOff_RegPSO* is active in the "PSO ALGORITHM SELECTION" subsection of the control panel's first section and switch *OnOff_ghist* is active in the "PSO HISTORIES TO BE MAINTAINED" subsection, the Particle Swarm Optimization Research Toolbox vertically concatenates each grouping's history of global bests, *ghist*, to construct the history of global bests over all groupings conducted during the RegPSO trial. The global best of any iteration can be extracted from *ghist_current_trial* as done with *ghist* in section "Analyzing Workspace Variables" of chapter "II: A Guided Walk Through".

Internal_i

This is one of two counters used for various purposes at various locations in the sense of "for i = 1:10". For example, the following code uses *Internal_i* to graph each particle's position versus iteration number according to the user's preferred graphing styles and to generate an appropriate legend.

```
if OnOff_graph_x
  for Internal_i = 1:np
```

```
    …
    if dim == 7
      plot(1:1:size(xhist,2)/dim, xhist(Internal_i, 1:dim:dim*(k + 1)), '-r*')
      plot(1:1:size(xhist,2)/dim, xhist(Internal_i, 2:dim:dim*(k + 1)), '--g*')
      plot(1:1:size(xhist,2)/dim, xhist(Internal_i, 3:dim:dim*(k + 1)), ':b*')
      plot(1:1:size(xhist,2)/dim, xhist(Internal_i, 4:dim:dim*(k + 1)), '-.c*')
      plot(1:1:size(xhist,2)/dim, xhist(Internal_i, 5:dim:dim*(k + 1)), '-m*')
      plot(1:1:size(xhist,2)/dim, xhist(Internal_i, 6:dim:dim*(k + 1)), '--y*')
      plot(1:1:size(xhist,2)/dim, xhist(Internal_i, 7:dim:dim*(k + 1)), ':k*')
      legend('x1', 'x2', 'x3', 'x4', 'x5', 'x6', 'x7', 'Location', 'EastOutside')
    end
    …
  end
end
```

Counters, graph settings, switches, and other variables that are not generally analyzed after execution are capitalized so that they will be displayed first when "whos" is typed at the command prompt. This floats them to the top of the list and off of the screen so that data which is expected to be of interest once the simulation(s) conclude are conveniently grouped together on the screen for efficient analysis. This topic is discussed further in section "Following Naming Conventions When Creating New Variables" of chapter "V. Contribution Standards".

To avoid confusion, "*Internal_*" is prefixed to further organize the workspace by clearly delineating variables that only make sense to the program from an internal perspective rather than from the perspective of the user after execution.

Internal_j

Together with *Internal_i*, this is one of two counters used for various purposes at various locations in the sense of "for j = 1:10". For example, the following code of the Quadric benchmark uses it to increment the problem dimensionality as dimensions are summed across.

```
for Internal_j = 1:dim
 f = sum(position_matrix(1:num_particles_2_evaluate,1:Internal_j), 2).^2 + f;
end
```

iter_mean_per_trial

The mean number of iterations conducted per trial is stored to *iter_mean_per_trial.* Some trials might require less iterations than others if switch *OnOff_Terminate_Upon_Success* is active, in which case the swarm will stop searching for better solutions once the accuracy specified in *thresh_for_succ* has been satisfied.

iter_success

This stores the iteration number at which each trial reaches success. When multiple trials are conducted, it is a vector whose length equals the number of trials conducted. When *iter_success* is displayed as NaN, the corresponding trial failed to minimize the function value to the user-specified threshold for success, *thresh_for_succ*.

iter_success_mean

This stores the average iteration number at which the threshold for success was reached. It is calculated by summing the real parts of *iter_success* and dividing by the number of successful trials. Unsuccessful trials are not considered in this computation so that *iter_success_mean* is the mean number of iterations required for successful trials to reduce the function value to *thresh_for_succ* or better.

iter_tot_all_trials

This counts the number of iterations conducted over all trials to compute the mean number of iterations conducted per trial, *iter_mean_per_trial*.

k

This counts the number of updates that have occurred. Initialization of the swarm occurs at *k* = 0 since no velocity or position update has yet occurred. The first swarm movement takes each particle from its position at *k* = 0 to its position at *k* = 1, which can be seen by examining the phase plots that are generated when switch *OnOff_phase_plot* is active in the control panel.

(Return to walk through)

l

This is the matrix of local bests, which is updated iteratively when Lbest PSO is activated via switch *OnOff_lbest* in the control panel's first section. At the end of each trial, it contains the best position found by each particle over the course of the search.

lbest_neighb_size

This is the neighborhood size as set in the control panel's first section under subsection "PSO ALGORITHM SELECTION". It is only created in the workspace when parent switch *OnOff_lbest* is active.

lhist

This matrix history of each iteration's local bests is activated via switch *OnOff_lhist* at the end of the control panel's first section. Table 5 in the walk through shows how to extract any iteration's matrix from this history per grouping of RegPSO or per trial of standard PSO (i.e. since standard PSO utilizes only one grouping).

lhist_current_trial

When switch *OnOff_RegPSO* is active in the "PSO ALGORITHM SELECTION" subsection of the control panel's first section and switch *OnOff_lhist* is active in the "PSO HISTORIES TO BE MAINTAINED" subsection, the Particle Swarm Optimization Research Toolbox horizontally concatenates each grouping's history of local or neighborhood bests, *lhist*, to construct the history of local bests over all groupings conducted during the RegPSO trial. The local bests of any iteration can be extracted from *lhist_current_trial* as done with *lhist* in Table 3 of section "Analyzing Workspace Variables" of chapter "II: A Guided Walk Through".

max_iter_over_all_groupings

This is the maximum number of iterations allowed in one RegPSO trial. See *max_iter_per_grouping* for more information.

max_iter_per_grouping

When regrouping is activated via switch *OnOff_RegPSO*, the maximum number of iterations allotted per grouping is specified via setting *max_iter_per_grouping*. If it is set equal to *max_iter_over_all_groupings*, regrouping will only be triggered when stagnation is detected according to formulas 4.1 - 4.4 of thesis [2]. The user also has the option of setting *max_iter_per_grouping* less than *max_iter_over_all_groupings* to prompt swarm regrouping even if a stray particle is still wandering about the search space after a considerable number of iterations. This seems to improve general performance and is akin to a group of individuals making a decision rather than being paralyzed by indecision should one individual still dissent after a considerable amount of time.

When "standard" PSO is used without regrouping, *max_iter_per_grouping* is the maximum number of iterations to be conducted by the trial.

max_FEs_over_all_groupings

When switch *OnOff_func_evals* is active, a maximum number of function evaluations is monitored instead of a maximum number of iterations. This setting specifies the maximum number of function evaluations allowed in one RegPSO trial. See *max_FEs_per_grouping* for more information.

max_FEs_per_grouping

When switch *OnOff_func_evals* is active, a maximum number of function evaluations is monitored instead of a maximum number of iterations. When regrouping is activated via switch *OnOff_RegPSO*, the maximum number of function evaluations allotted per grouping is specified via setting *max_FEs_per_grouping*. If it is set equal to *max_FEs_over_all_groupings*, regrouping will only be triggered when stagnation is detected according to formulas 4.1 - 4.4 of thesis [2]. The user also has the option of setting *max_FEs_per_grouping* less than *max_FEs_over_all_groupings* to prompt swarm regrouping even if a stray particle is still wandering about the search space after a considerable number of function evaluations. This seems to improve general performance and is akin to a group of individuals making a decision rather than being paralyzed by indecision should one individual still dissent after a considerable amount of time.

When "standard" PSO is used without regrouping, *max_FEs_per_grouping* is the maximum number of function evaluations to be conducted by the trial.

MPSO_FE_counter

At the end of each MPSO start, *MPSO_FE_counter* adds the amount of function evaluations done in that start to its previous value, which was initialized to zero in "Trial_Initializations.m".

MPSO_FE_counter is used by the relevant while headers within “Reg_Method_1.m” and either “gbest_core.m” or “lbest_core.m” to ensure that the maximum number of function evaluations, *MPSO_max_FEs*, specified in the control panel is not surpassed: since the value of *MPSO_FE_counter* is not updated until the end of each start, while headers add the number of function evaluations done within the current start to *MPSO_FE_counter* to determine the total number of function evaluations done over all starts.

MPSO_k

This is 0 when the swarm is initialized and increments with each ensuing iteration of MPSO when switch *OnOff_MPSO* is active in the control panel's first section.

np

This stores the number of particles or swarm size. It is set in section “(2) PARTICLE SWARM ALGORITHM SETTINGS” of the control panel. It is static except in the professional version of the toolbox when auto-incrementing the swarm size to find high-quality parameter combinations as done for Chapter III of thesis [2].

num_particles_2_evaluate

When calling an objective function, this is the internal variable that specifies the number of positions to be evaluated. It will typically be either 1 or *np* (i.e. the number of particles in the swarm, which is the same as the number of rows in position matrix *x*) and usually assumes the latter value to evaluate each position in the swarm.

num_trials_successful

When multiple trials are conducted, this counts the number of trials for which a function value less than or equal to *thresh_for_succ* is produced.

OnOff_asymmetric_initialization

This switch activates asymmetric initialization, which modifies the initialization space by setting xmin = xmax/2 on benchmarks instead of xmin = -xmax. The idea is to increase the difficulty of locating the global minimizer. This tests the effectiveness of the optimization algorithm at finding solutions that might inadvertently be excluded from the initialization space should the algorithm be applied to problems whose solutions are unknown.

OnOff_func_evals

When switch *OnOff_func_evals* is active in the “PSO DATA TO BE RETAINED & SAVED” subsection of the control panel’s section “(1) BASIC SWITCHES & PSO ALGORITHM SELECTION”, function evaluations will be used as a termination criterion; when inactive, iterations will be monitored as a termination criterion instead. The maximum number of function evaluations or iterations per grouping is set at the end of section “(2) PARTICLE SWARM ALGORITHM SETTINGS” via settings *max_FEs_per_grouping* or *max_iter_per_grouping* respectively; these specify the maximum number allowed per RegPSO grouping or per standard PSO trial (i.e. since a standard trial consists of only one grouping). The maximum number of function evaluations or iterations over all groupings within a RegPSO trial is set at the end of section "(3) REGPSO ALGORITHM SWITCHES & SETTINGS" via settings *max_FEs_over_all_groupings* or *max_iter_over_all_groupings* respectively.

OnOff_NormR_stag_det

Van den Bergh's normalized swarm radius criterion terminates the search when particles are all near the global best, which is a good indication of stagnation [3]. According to quick empirical testing done during thesis, monitoring particles' proximities to detect stagnation seems more effective than monitoring the number of iterations for which the function value fails to improve, which stands to reason since monitoring the source of a problem is more efficient than waiting for an undesirable side effect to occur - the logic behind preventive medicine and firewalls. Switch *OnOff_NormR_stag_det* can be activated in subsection "TERMINATION CRITERIA FOR REGPSO & PSO ALGORITHMS" in the control panel's first section.

RegPSO regroups the swarm about the global best when stagnation is detected instead of terminating the search, which allows the function value to improve over time. If switch *OnOff_RegPSO* is activated in conjunction with switch *OnOff_NormR_stag_det*, the swarm will regroup within a regrouping space defined according to the degree of uncertainty inferred per dimension from the distribution of particles when stagnation was detected [1].

OnOff_position_clamping

This prevents particles from leaving the search space by clamping their positions to [-xmax, xmax] the same way velocity clamping restricts velocities to [-vmax, vmax].

OnOff_RegPSO

When stagnation is detected, RegPSO regroups the swarm about the global best for continued progress instead of terminating the search or wasting computational time; this allows the function value to improve over time. RegPSO can be activated via switch *OnOff_RegPSO* in subsection "PSO Algorithm Selection" of the control panel's first section. An example of the effectiveness of RegPSO is given at the end of Chapter III's walk through.

OnOff_SuccessfulUnsuccessful

When switch *OnOff_SuccessfulUnsuccessful* is activated in subsection "REGPSO & PSO SUCCESS MEASURES" of the control panel's first section, each trial will be classified as successful or unsuccessful. In some cases, users may only be interested in the statistics resulting from the trials conducted and may not be interested in classifying trials as successful or unsuccessful; but the feature can be a useful way of tracking outliers since any function value larger than the threshold for success as set in "Objectives.m" will be classified as unsuccessful, thereby essentially counting the number of outliers.

OnOff_Terminate_Upon_Success

In subsection "REGPSO & PSO SUCCESS MEASURES" of the control panel's first section, if switch *OnOff_Terminate_Upon_Success* is activated along with *OnOff_SuccessfulUnsuccessful*, trials will terminate early upon minimizing the function value to *thresh_for_succ*, the threshold required for a trial to be considered successful.

OnOff_v_reset

This sets to zero any velocity component responsible for pushing a particle outside the search space and recalculates the position so that particles remain within the search space. To my knowledge, the first work on this topic was done by Gerhard Venter [5].

OPSO_ghost_FEs_per_grouping

This counts in "gbest_initialization.m" and "gbest_core_loop.m" the additional function evaluations done by Opposition-Based PSO (OPSO) and the Cauchy mutation of global best. OPSO and the Cauchy mutation can be used separately, and the count will still be correct. It also counts in "lbest_initialization.m" and "lbest_core.m" the additional function evaluations done by OPSO without Cauchy mutation of global best since Lbest PSO does not make use of the global best.

It is initialized to zero within "gbest_initialization.m" or "lbest_initialization.m," depending on the core algorithm selected.

OPSO_ghost_FEs_RegPSO

OPSO_ghost_FEs_RegPSO adds over all groupings across a RegPSO trial the values returned by *OPSO_ghost_FEs_per_grouping* after each grouping. After all RegPSO groupings, it contains the number of additional function evaluations conducted by OPSO and the Cauchy mutation of global best, which also works if only one of OPSO or the Cauchy mutation is enabled.

It is initialized within "Reg_Methods_0And1.m" as "*OPSO_ghost_FEs_RegPSO* = *OPSO_ghost_FEs_per_grouping*" and accrues within "Reg_Method_1.m" as "*OPSO_ghost_FEs_RegPSO* = *OPSO_ghost_FEs_RegPSO* + *OPSO_ghost_FEs_per_grouping*". It is then used within the relevant while header of "gbest_core.m" or "lbest_core.m" to ensure that the maximum number of function evaluations allotted over all groupings of a RegPSO trial is not surpassed: "(*k_RegPSO* + *k* + 1)**np* + *OPSO_ghost_FEs_RegPSO*) < *max_FEs_over_all_groupings*".

p

This is the matrix of personal bests, which is updated iteratively. At the end of a trial, it contains the best position found by each particle over the course of the search.

phist

This history of each iteration's matrix of personal bests is activated via switch *OnOff_phist* at the end of the control panel's first section. Table 3 in the guided walk through shows how to extract the matrix of any iteration from this history per grouping of RegPSO or per trial of standard PSO (i.e. since standard PSO utilizes only one grouping).

phist_current_trial

When switch *OnOff_RegPSO* is active in the "PSO ALGORITHM SELECTION" subsection of the control panel's first section and switch *OnOff_phist* is active in the "PSO HISTORIES TO BE MAINTAINED" subsection, the Particle Swarm Optimization Research Toolbox horizontally concatenates each grouping's history of personal bests, *phist*, to construct the history of personal bests over all groupings conducted during the RegPSO trial. The personal bests of any iteration can be extracted from *phist_current_trial* as done with *phist* in Table 3 of section "Analyzing Workspace Variables" of chapter "II: A Guided Walk Through".

position_matrix

When calling an objective function, this internal variable stores the position or positions passed in for evaluation. It usually assumes the full matrix, *x,* to evaluate each position in the swarm; however, any single position or subset of *x* may be passed in as well (e.g. a perturbation of the global best).

r1

This is the *np* by *dim* matrix of pseudo-random numbers used in the cognitive term of the velocity update equation shown below. It is multiplied element-wise by the difference between the matrix of personal bests, *p*, and the position matrix, *x*.

Matricized Velocity Update Equation

$$\mathbf{v}(k+1)=\mathbf{v}(k)+c_1\mathbf{r}_1(k)\bullet\left(\mathbf{p}(k)-\mathbf{x}(k)\right)+c_2\mathbf{r}_2(k)\bullet\left(\mathbf{g}(k)-\mathbf{x}(k)\right)$$

r2

This is the *np* by *dim* matrix of pseudo-random numbers used in the social term of the velocity update equation shown above. It is multiplied element-wise by the difference between the matrix of personal bests, *p*, and the position matrix, *x*.

Rand_seq_start_point

This controls the state of the randomizer at which each trial begins to: (i) ensure future reproducibility of data, and (ii) use a unique sequence of random numbers for each trial automatically conducted by the toolbox.

range_IS

When each trial is initialized (i.e. within "Trial_Initializations.m"), the user-specified scalar or vector *range_IS0* is replicated across each row of *range_IS* to accommodate simple matrix subtraction: this conversion is done internally to enable the user to specify the range in the simplest terms possible.

For each regrouping of RegPSO, *range_IS* is calculated per dimension proportionally to the degree of uncertainty inferred from the swarm state when premature convergence is detected. The range of each new regrouping space is per dimension the minimum of: (i) the original range of the initialization space and (ii) the product of the regrouping factor and the maximum deviation of any particle from global best along that dimension. The latter is accomplished via

```
        range_IS = repmat(min(reg_fact*max(abs(x - g)), range_IS0), np, 1)
```

within "Reg_Method_1_regrouping.m".

range_IS0

This is the initial range of the initialization space as set by the user in Objectives.m. It may be set as a scalar when all dimensions have the same range or as a vector when any dimension has a unique range; an example of the latter is given under section Setting the Center and Range of the Initialization Space of chapter "III: More Details on Select Topics".

Unlike variable *range_IS*, which is updated with each regrouping, *range_IS0* is unmodified during program execution.

reg_fact

If switch *OnOff_RegPSO* is active, the swarm will regroup within a **regrouping space** defined according to **the degree of uncertainty inferred per dimension** from the distribution of particles when stagnation is detected [2]. The **regrouping factor** is multiplied by each dimension's **uncertainty** to determine the **regrouping space**, while **clamping each dimension's range to its initial value**, as shown in formula 4.6 of thesis, which is accomplished via the following code within "Reg_Method_1_regrouping.m".

```
        range_IS = repmat(min(reg_fact*max(abs(x - g)), range_IS0), np, 1)
```

Once the range of the regrouping space has been defined as shown above, particles are regrouped about the global best according to formula 4.7 of thesis. The velocity clamping vector is then calculated according to formula 4.10 in proportion to the range per dimension of the new regrouping space, and that maximum velocity vector is used to re-initialize velocities as shown in formula 4.11. Personal bests are then re-initialized as shown in formula 4.12, but the global best is remembered to be improved upon. The search then continues as usual until stagnation is detected again.

RegPSO_grouping_counter

When RegPSO is activated via switch *OnOff_RegPSO* in the control panel, *RegPSO_grouping_counter* is initialized to 1 within "Reg_Methods_0&1.m" and incremented by 1 with each regrouping performed by "Reg_Method_1_regrouping.m".

RegPSO_k

If RegPSO is activated via switch *OnOff_RegPSO* in the control panel's first section to liberate the swarm from the state of premature convergence when it is detected, *RegPSO_k* is initialized to 0 during swarm initialization and incremented with each ensuing iteration. Its value at the end of each trial is 1 less than the number of iterations conducted since initialization of the swarm counts as an iteration but not as an update.

RegPSO_k may be less for one trial than for others if that trial terminates early as a result of the true global minimum being achieved without discernable error (i.e. $f < 10^{-323}$ cannot be differentiated from true zero by MATLAB).

RegPSO_k_after_each_grouping

When only one RegPSO trial is conducted, this is a vector storing the iteration numbers at which premature convergence was detected; adding 1 to each column would produce a vector storing the iteration numbers at which regrouping occurred since initialization of the swarm counts as an iteration but not as an update. When multiple trials are conducted, this is a matrix storing each trial's vector in chronological order.

stag_thresh

Activating switch *OnOff_NormR_stag_det* causes the search to terminate when the normalized swarm radius becomes smaller than *stag_thresh,* which is a good indication that the swarm has stagnated [3]. If switch *OnOff_RegPSO* is active concurrently, the swarm will regroup within a regrouping space defined according to the degree of uncertainty inferred per dimension from the distribution of particles when stagnation is detected [2].

Stagnation is detected according to formulas 4.1 – 4.4 of thesis, which are implemented within "gbest_core.m" and "lbest_core.m" via termination criterion

```
"max(sqrt(sum((x - g).^2, 2)))/sqrt(sum((range_IS(1, :)).^2)) > stag_thresh".
```

thresh_for_succ

When the best function value produced by a trial is less than or equal to the threshold for success, *thresh_for_succ*, the trial will be considered successful. The value of *thresh_for_succ* is set within "Objectives.m" and should be appropriate to both the objective and its problem dimensionality.

When switches *OnOff_SuccessfulUnsuccessful* and *OnOff_Terminate_Upon_Success* are active, each trial will terminate upon reaching the threshold for success instead of attempting further optimization.

The threshold determines the subjective value at which a trial will be considered successful. It can be useful in conjunction with statistics since it shows the number of trials performing worse than the fitness value of interest, which gives an idea of how such trials affect the mean and median fitness values, thus complementing the standard deviation.

true_global_minimum

The true global minimum is specified in section "(6) OBJECTIVES" of the control panel so that any trial minimizing the function value to the lowest value possible will terminate rather than spending excess time in computation. Once the number of decimal places carried internally by MATLAB all agree entirely with *true_global_minimum*, no further optimization is possible. Of course, when many decision variables map through a complex formula to a fitness value, the decision variables will not be optimized to the same number of decimal places as the fitness value itself.

v

Each row of velocity matrix *v* stores a particle's velocity. For example, "v(1, :)" holds the velocity vector of the first particle, "v(2, :)" holds the velocity vector of the second particle, and so forth. Matrix *v* is updated iteratively according to the velocity update equation of PSO; so at the completion of a trial, it stores the final resting place of each particle.

Matrix *v* is initialized within "gbest_iteration_0.m" or "lbest_initialization.m," depending on whether Gbest or Lbest PSO is selected according to switch *OnOff_lbest*. It is updated in "gbest_core_loop.m" or "lbest_core_loop.m".

vhist

This history of all iterations' velocity matrices is activated via switch *OnOff_vhist* at the end of the control panel's first section. Table 2 in the walk through shows how to extract the position matrix of any iteration from this history per grouping of RegPSO or per trial of standard PSO (i.e. since standard PSO utilizes only one grouping).

vhist_current_trial

When switch *OnOff_RegPSO* is active in the "PSO ALGORITHM SELECTION" subsection of the control panel's first section and switch *OnOff_vhist* is active in the "PSO HISTORIES TO BE MAINTAINED" subsection, the Particle Swarm Optimization Research Toolbox horizontally concatenates each grouping's history of velocity vectors, *vhist*, to construct the history of velocity vectors over all groupings conducted during the RegPSO trial. The velocity vectors of any iteration can be extracted from *vhist_current_trial* as done with *vhist* in section "Analyzing Workspace Variables" of chapter "II: A Guided Walk Through".

vmax

This is the matrix of maximum velocities used for velocity clamping. It is defined as

```
        vmax_perc.*range_IS
```

within "Reg_Methods_0And1.m" and "Reg_Method_1_regrouping.m". If the same range is used on all dimensions, *vmax* will be the same on all dimensions. But if the range on one dimension is larger than the range on another by any scale factor, its maximum velocity will be larger by the same scale factor so that the same fraction of the search space can be combed per dimension per iteration. When RegPSO is utilized, each dimension of the range of the regrouping space is defined proportionally to the degree of uncertainty on that dimension at stagnation. Since the range per dimension varies, so does the maximum velocity per dimension.

x

Each row of position matrix x stores a particle's position. For example, "x(1, :)" holds the position vector of the first particle, "x(2, :)" holds the position vector of the second particle, and so forth. Matrix *x* is updated iteratively according to the position update equation of PSO; so at the completion of a trial, it stores the final resting place of each particle.

Matrix *x* is initialized within "Reg_Methods_0And1.m" and updated in "gbest_core_loop.m" or "lbest_core_loop.m," depending on whether Gbest or Lbest PSO is selected according to switch *OnOff_lbest*.

xhist

This history of each iteration's position matrix is activated via switch *OnOff_xhist* at the end of the control panel's first section. Table 1 in the walk through shows how to extract the position matrix of any iteration from this history per grouping of RegPSO or per trial of standard PSO (i.e. since standard PSO utilizes only one grouping).

xhist_current_trial

When switch *OnOff_RegPSO* is active in the "PSO ALGORITHM SELECTION" subsection of the control panel's first section and switch *OnOff_xhist* is active in the "PSO HISTORIES TO BE MAINTAINED" subsection, the Particle Swarm Optimization Research Toolbox horizontally concatenates each grouping's history of position vectors, *xhist*, to construct the history of position vectors over all groupings conducted during the RegPSO trial. The position vectors of any iteration can be extracted from *xhist_current_trial* as done with *xhist* in section "Analyzing Workspace Variables" of chapter "II: A Guided Walk Through".

xmax

When velocity reset or position clamping is activated to prevent particles from leaving the search space, *xmax* holds the maximum values that particles are not allowed to exceed. When each dimension of the search space has the same range and center, as is the case for popular benchmarks and many real-world application problems, *xmax* is scalar and all dimensions are clamped to lie within [*xmin*, *xmax*]. For application problems having a different range of values on some dimensions than on others, such as for the example problem in section "Setting the Center and Range of the Initialization Space," *xmax* is a vector storing the maximum value allowed per dimension.

xmin

When velocity reset or position clamping is activated to prevent particles from leaving the search space, *xmin* holds the minimum values that particles are not allowed to exceed. When each dimension of the search space has the same range and center, as is the case for popular benchmarks and many real-world application problems, *xmin* is scalar and all dimensions are clamped to lie within [*xmin*, *xmax*]. For application problems having a different range of values on some dimensions than on others, such as for the example problem in section "Setting the Center and Range of the Initialization Space," *xmin* is a vector storing the minimum value allowed per dimension.

Of Functions

This section is a work in progress. If you have any questions or recommendations, please email george [at] georgeevers.org, where you may also request to be notified of future updates.

ObjFun_Ackley

This evaluates the position(s) passed into the Ackley function. The first input argument is the position vector or matrix to be evaluated. Rather than iteratively re-calculating the number of particles from the dimensionality of the position matrix each time the function is called, this information is simply passed into the function via the second input argument. The output is the column matrix of function values corresponding to each particle/row evaluated.

The objective function is defined as follows.

$$f(\vec{x}) = 20 + e - 20e^{-0.2\sqrt{\frac{\sum_{j=1}^{n} x_j^2}{n}}} - e^{\frac{\sum_{j=1}^{n}\cos(2\pi x_j)}{n}}$$

$$-32 \le x_j \le 32$$

ObjFun_Griewangk

This evaluates the position(s) passed into the Griewangk function. The first input argument is the position vector or matrix to be evaluated. Rather than iteratively re-calculating the number of particles from the dimensionality of the position matrix, this information is simply passed into the function via the second input argument. The output is the column matrix of function values corresponding to each particle/row evaluated.

The function to be minimized is shown below.

$$f(\vec{x}) = 1 + \sum_{j=1}^{n} \frac{x_j^2}{4000} - \prod_{j=1}^{n} \cos\left(\frac{x_j}{\sqrt{j}}\right)$$

$$-600 \le x_j \le 600$$

ObjFun_Quadric

This evaluates the position(s) passed into the Quadric function. The first input argument is the position vector or matrix to be evaluated. Rather than iteratively re-calculating the number of particles from the dimensionality of the position matrix, this information is simply passed into the function via the second input argument. The output is the column matrix of function values corresponding to each particle/row evaluated.

Each particle's position vector is iteratively mapped to a scalar function value as follows.

$$f(\vec{x}) = \sum_{j=1}^{n}\left(\sum_{k=1}^{j} x_j\right)^2$$

$$-100 \le x_j \le 100$$

ObjFun_Quartic_Noisy

This evaluates the position(s) passed into the noisy Quartic function. The first input argument is the position vector or matrix to be evaluated. Rather than iteratively re-calculating the number of particles from the dimensionality of the position matrix, this information is simply passed into the function via the second input argument. The output is the column matrix of function values corresponding to each particle/row evaluated.

Both individually and collectively, particles attempt to minimize the formula below.

$$f(\vec{x}) = random[0,1) + \sum_{i=1}^{n} i \cdot x_i^4$$

$$-1.28 \le x_j \le 1.28$$

ObjFun_Rastrigin

This evaluates the position(s) passed into the Rastrigin function. The first input argument is the position vector or matrix to be evaluated. Rather than iteratively re-calculating the number of particles from the dimensionality of the position matrix, this information is simply passed into the function via the second input argument. The output is the column matrix of function values corresponding to each particle/row evaluated.

Shown below is the formula by which each particle's function value is determined.

$$f(\vec{x}) = 10n + \sum_{j=1}^{n}(x_j^2 - 10\cos(2\pi x_j))$$

$$-5.12 \le x_j \le 5.12$$

ObjFun_Rosenbrock

This evaluates the position(s) passed into the Rosenbrock function. The first input argument is the position vector or matrix to be evaluated. Rather than iteratively re-calculating the number of particles from the dimensionality of the position matrix, this information is simply passed into the function via the second input argument. The output is the column matrix of function values corresponding to each particle/row evaluated.

Personal and global bests are updated based on how well they minimize the following formula.

$$f(\vec{x}) = \sum_{j=1}^{n-1} \left(100(x_{j+1} - x_j^2)^2 + (1 - x_j)^2 \right)$$

$$-30 \le x_j \le 30$$

ObjFun_Rosenbrock_Engelbrecht_VandenBergh

This evaluates positions according to the version of the Rosenbrock formula used in Van den Bergh's PhD thesis. The first input argument is the position vector or matrix to be evaluated. Rather than iteratively re-calculating the number of particles from the dimensionality of the position matrix, this information is simply passed into the function via the second input argument. The output is the column matrix of function values corresponding to each particle/row evaluated.

Notice that every summand produced by an even value of *j* in the Rosenbrock formulation above is excluded by the formulation below. The first summands of the two formulations are identical (i.e. for j = 1), but the formulation below excludes every other summand produced by the formulation above since the first subscript is always even, whereas the first subscript above counts all integers from 1 through (n – 1). Given problem dimensionality, n, the formulation below sums about half as many summands as the formulation above and is therefore a less complicated formula.

$$f(\vec{x}) = \sum_{j=1}^{n/2} \left(100(x_{2j} - x_{2j-1}^2)^2 + (1 - x_{2j-1})^2 \right)$$

$$-2.048 \le x_j \le 2.048$$

ObjFun_Schaffers_f6

This evaluates the position(s) passed into the two-dimensional Schaffer's f6 function. The first input argument is the position vector or matrix to be evaluated. Rather than iteratively re-calculating the number of particles from the dimensionality of the position matrix, this information is simply passed into the function via the second input argument. The output is the column matrix of function values corresponding to each particle/row evaluated.

The formula by which each particle's function value is determined from the two dimensions to be optimized is shown below.

$$f(\vec{x}) = 0.5 + \frac{\left(\sin(\sqrt{x_1^2 + x_2^2}\right)^2 - 0.5}{(1.0 + 0.001(x_1^2 + x_2^2))^2}$$

$$-100 \le x_j \le 100$$

ObjFun_Schwefel

Formulation Used by PSO Research Toolbox

This evaluates the value(s) of the position(s) passed into the Schwefel function. The first input argument is the position vector or matrix to be evaluated. Rather than iteratively re-calculating the number of particles from the dimensionality of the position matrix, this information is simply passed into the function via the second input argument. The output is the column matrix of function values corresponding to each particle/row evaluated.

Each position's function value is determined according to the formula below.

$$f_1(\vec{x}) = 418.9828872724337998079136013 98n - \sum_{j=1}^{n} x_j \sin\left(\sqrt{|x_j|}\right)$$

$$-500 \le x_j \le 500$$

Schwefel's crests and troughs increase in magnitude with distance from the origin so that there is no global minimum or global minimizer when the search space is unbounded; consequently, particles must be confined to the search space to prevent them from finding positions outside the feasible search space producing better function values. This is especially important since the global minimizer of the bounded search space is fairly near its edge. To restrict particles to the search space, use position clamping or velocity reset.

When the search space is bounded, the global minimum of 0 occurs at vector 420.968746*ones(1, dim).

Other Formulation

Some authors use the following formulation instead, which produces the same global minimum of 0 at vector -420.968746*ones(1, dim), which is per dimension the additive inverse of the global minimizer of the first formulation.

$$f_2(\vec{x}) = 418.9828872724337998079136013 98n + \sum_{j=1}^{n} x_j \sin\left(\sqrt{|x_j|}\right)$$

$$-500 \le x_j \le 500$$

The Particle Swarm Optimization Research Toolbox uses the first formulation to produce a global minimizer that is positive on each dimension, but both formulations are of equivalent mathematical difficulty by symmetry across the f axis: i.e. since $f_1(\vec{x}) = f_2(-\vec{x})$. The reader is merely cautioned that both formulations are in circulation for the purpose of understanding which global minimizer produces the global minimum of 0. As a simple mnemonic, Schwefel's global minimizer is per dimension opposite in sign to that seen in the formula itself.

ObjFun_Spherical

This evaluates the position(s) passed into the Sphere function, which is continuous, strictly convex, and unimodal. The first input argument is the position vector or matrix to be evaluated. Rather than iteratively re-calculating the number of particles from the dimensionality of the position matrix, this information is simply passed into the function via the second input argument. The output is the column matrix of function values corresponding to each particle/row evaluated.

The cost function to be minimized is defined as follows.

$$f(\vec{x}) = \sum_{j=1}^{n} x_j^2$$

$$-100 \le x_j \le 100$$

Whether the search space is bounded or unbounded, the global minimum of 0 occurs at the null vector.

ObjFun_Weighted_Sphere

This evaluates the position(s) passed into the weighted Sphere function, which is continuous, convex, and unimodal. The first input argument is the position vector or matrix to be evaluated. Rather than iteratively re-calculating the number of particles from the dimensionality of the position matrix, it is simply passed into the function via the second input argument. The output is the column matrix of function values corresponding to each particle/row evaluated.

Each position vector encountered by a particle is translated to a function value according to the formula below.

$$f(\vec{x}) = \sum_{j=1}^{n} j \cdot x_j^2$$

$$-5.12 \le x_j \le 5.12$$

Whether the search space is bounded or unbounded, the global minimum of 0 occurs at the null vector.

Of Files

If you are interested in seeing this section developed, please email george [at] georgeevers.org, where you may also request notification of future updates. For now, please visit chapter "Program Flow," which describes the most important files.

Appendix B: Error Correction

java.lang.OutOfMemoryError: Java heap space

This error was encountered when generating many swarm trajectory graphs. To address the problem, I introduced switch *OnOff_reuse_figures*, which causes all graphs to be generated within the same figure – resulting in a frame-by-frame video. Each frame can be saved automatically by activating switch *OnOff_autosave_figs* in the "REGPSO & PSO GRAPHING SWITCHES" subsection of the control panel.

Error using ==> save: Unable to write file… No such file or directory

Windows concatenates a file's directory and filename to uniquely identify its location when saving, copying, or moving, which can become problematic when both a file and its containing directory have lengthy names. This happens more often than it should since My Documents is unnecessarily far from the root directory, which produces unnecessarily long directory names for most saved content. Windows 7 makes some progress in this regard by removing "Documents & Settings" from the directory name, but long folder names within the MATLAB subfolder of My Documents may still cause problems.

This error can be voided altogether simply by unzipping the PSO Research Toolbox directly to "C:\". If that is inconvenient: (i) close MATLAB so that each folder comprising the current directory can be renamed, (ii) highlight problematic folders with lengthy names one at a time, and (iii) press F2 to assign a shorter name such as "PSORT".

Appendix C: Efficient MATLAB Shortcuts

Editing Code

Ctrl + [	Decrease indent	(…of highlighted text)
Ctrl +] or Tab	Increase indent	(…of highlighted text)
Ctrl + G	Go to line #	
Ctrl + R	Comment line	(i.e. current line or highlighted lines)
Ctrl + T	Uncomment line	(i.e. current line or highlighted lines)
Ctrl + H	Search or replace	(also works in Word, Word Pad, Notepad, and Notepad++)
Ctrl + A	Select All	
Ctrl + C	Copy	
Ctrl + P	Paste	
Ctrl + Z	Undo	
Ctrl + Y	Redo	

Navigating

Ctrl + 0	Switch to Command Window	
Ctrl + Shift + 0	Switch to Editor	
Ctrl + Page Up	Tab up	(between open tabs in Editor)
Ctrl + Page Down	Tab Down	(between open tabs in Editor)

Managing Files

Ctrl + N	New Editor tab	(also works in Word, Word Pad, Notepad, and Notepad++)
Ctrl + O	Open	(also works in Word, Word Pad, Notepad, and Notepad++)
Ctrl + W	Close Editor tab	(also works with Windows folders and Notepad ++ tabs)
Alt + F4	Close app	(also works in Word, Word Pad, Notepad, and Notepad++)
Ctrl + S	Save	(also works in Word, Word Pad, Notepad, and Notepad++)

Appendix D: Citation of Toolbox or Documentation

When citing the toolbox, please include the version number to ensure the reproducibility of data, and please include URL "http://www.georgeevers.org/pso_research_toolbox.htm".

When citing the documentation, please include the version number to avoid any potential confusion should anyone else try to look up the same information in an older version. Also, please include URL "http://www.georgeevers.org/pso_research_toolbox_documentation.doc".

Paste-able Examples

Evers, G., *PSO Research Toolbox* (Version 20yymmdd), M.S. thesis code, 2011, < http://www.georgeevers.org/pso_research_toolbox.htm >

Evers, G., *PSO Research Toolbox Documentation* (Version 20yymmdd), M.S. thesis code documentation, 2011, < http://www.georgeevers.org/pso_research_toolbox_documentation.pdf >