Last updated: December 2021

Contact:
Andrew Brady
andrew.brady1@richmond.edu

This is the README for the code and data associated with the paper:
Negative Latin Square Type Partial Difference Sets in Nonabelian Groups of Order 64 (Brady)

Note: I have commented with #CHANGEPATH anywhere you may need to change a file path to make a program work

Note: The format of a PDS is always outputted in position form. To recover the actual group elements from a group with id gid, do the following:

```
g := SmallGroup(64,gid);
e := Elements(g);
pds_elementform = List(pds_position_form, x->e[x]);
```

Ex. let pds := [2,3, ... ,40, 56]. Then e[2], e[3]. e[40], and e[56] are all elements of the PDS.

CheckPDSInequivalence3.g:

Checks PDSs found to see if they are equivalent

Input: The lists of PDSs found in the PDS18_output folder
Output: Stored in PDS18Ineq_table.txt. A list of lists. Each element of the list is of the form [gid, [x1, x2, ..., xn]]. gid is a group number, and x1, x2, ..., xn are the inequivalent PDSs in the group SmallGroup(64,gid)

DisjointPDS.g

Check PDSs found to see which groups contain a pair of disjoint PDSs
Input: The PDSs found by exhaustive search
Output: Stored in PDS18_disjointPDSs_list.txt. A list of lists. Each element of the list is of the form [gid, pds1, pds2].
gid is a group number (representing the group SmallGroup(64,gid))
pds1 and pds2 are PDSs in position form, corresponding to the list of group elements found by Elements(SmallGroup(64,gid))

Acknowledgements:
This code takes inspiration from Dr. Ken Smith's disjoint difference set program.

PDS_18_EA8img2.g

Create input files to allow Java to run an exhaustive PDS search

Note: An empty folder PDS18_gap_tables must be manually created BEFORE the program is run.

Input: The groups we want to precompute (in our case, [55..266])
Output: A folder, PDS18_gap_tables, containing files, gaptables64.gid (where gid is the group number), containing the precomputed group values for SmallGroup(64,gid)

Acknowledgements:
The procedure within the code that iterates over normal subgroups to find the groups we are interested in is based on code written by Dr. Smith.

PDS18_looprun.java

Using the group data computed by PDS_18_EA8img2.g, finds all of the PDSs with k=18 in groups 55 through 266

Note: An empty folder PDS18_output must be manually created BEFORE the program is run.

Input: The GAP table files created by PDS_18_EA8img2.g
Output: A folder, PDS18_output, containing files of the form pds18_64.gid.txt (gid is group id). Each file pds18_64.gid.txt contains a list, pds_list, containing all of the PDSs from the group SmallGroup(64,gid), in position form. Note that the file pds18_64.gid.txt is readable by GAP, and puts all of the PDSs from the group with id# gid in the list pds_list

Acknowledgements:
Thanks to Nikita Morozov & Sreya Aluri for advice on how to make this code more efficient: specifically, using index assignment to avoid a remove call and not using a recursive method (iterative or for loops instead).
Thanks to Dr. Prateek Bhakta for suggesting checks in the middle of the nested PDS search loops to reduce runtime.
The convolution check for PDSs was adapted from Drs. Smith and Kaliszewski's convolution table procedures.

NumberOfPDSs.g

This program calculates how many PDSs were found in the search. I include this example to demonstrate how the data can be read into a GAP program.

Input: The PDSs found by the search
Output: A single number, the total amount of PDSs found by the search

---

PDS18_cosetdistribution.g

This program determines which PDSs from our search can be written as the union of three reversible difference set translates. This program only checks the list of inequivalent PDSs found by checkPDSInequivalence3.g.

Input: PDSs found from search, inequivalent PDS list
Output: Stored in PDS18_revablePDSs.txt. A list of lists of the PDSs that can be broken down into three reversible difference sets. The output is formatted as follows: Each element of the list is of the form [gid, [x1, x2, ..., xn]]. gid is a group number, and x1, x2, ..., xn are the inequivalent PDSs in the group SmallGroup(64,gid).

Ex. [56, [ ]] means that no PDSs in group 56 can be broken down
Ex. [232, [1,19,193, 257]] means that PDSs #1, #129, #193, and #257 can be broken down. The numbers 1,19,193,257 refer to the indices in pds_list in pds18_64.232.txt that correspond to the inequivalent PDSs that can be broken down.

PDS18_srgtest.g

This program finds the strongly regular graph isomorphism classes of the inequivalent PDSs found and compares them to Haemers & Spence's (2001) SRGs.

Input: The PDSs found, the inequivalent PDS list, a list of Haemers & Spence's (64,18,2,6) SRGs, in GAP readable format (derived from http://www.maths.gla.ac.uk/~es/srgraphs.php)

Output: Stored in PDS18_fullsrgWithSpence_graphiso.txt. A list of lists. Each element of the outer list is of the form [SRGid1, [[gid1, [a1, a2, ..., an]], [gid2, [a1, a2, ..., am]], ... [gidk, [a1, a2, ..., ap]]], SRGid2, [[gid1, [a1, a2, ..., aq]], [gid2, [a1, a2, ..., ar]], ... [gidk, [a1, a2, ..., as]]], ...]

SRGid is the id in Haemers & Spence's list of the SRG selected (if SRGid=3, we are considering the 3rd SRG in their list). Each SRG is equivalent to the SRGs created by some PDSs in some groups. For each group with id gid containing an SRG equivalent to SRGid, we list all of the

indices a1, a2, ..., an that correspond to an inequivalent PDS creating the SRGid. Note that these indices refer to the index of the PDS within pds_list in pds18_64.gid.txt.

Ex. [6, [[123, [1, 129]], [167, [1]], [174, [1]], [179, [1]]]] means that, among the list of inequivalent PDSs we have generated, the SRG with id 6 can be created from PDSs 1 and 129 in group 123, PDS 1 in group 167, PDS 1 in group 174, and PDS 1 in group 179.

---

helper_functions.g

Contains a couple of nice helper functions that are occasionally used in other programs.

Input: None
Output: Adds the functions described in the program so that they can be used later.