Wirt_Hm_Suite Documentation

Note that this research project is ongoing and the programs referenced in this document are still under development and subject to change. This document is currently incomplete and will be periodically updated.

Purpose

The Wirt_Hm_Suite is a set of programs designed to conduct an exhaustive search for surjective homomorphisms from the fundamental group of a topological knot with Wirtinger number *n* to a finite Coxeter group of order *n*. The existence of such a surjective homomorphism is equivalent to a proof of the Cappell-Shaneson Meridional Rank Conjecture (Problem 1.11 on the Kirby List[i]) for the knot. For a proof that a surjective homomorphism to an algebraic group of order equal to a knot's Wirtinger number is sufficient to prove this conjecture, as well as an explanation of the Wirtinger number itself, see R. Blair et al.[ii]

Requirements to Run

1. A Python installation. Python 3.7 or above is recommended.
2. A Microsoft Excel installation. Excel 2010 or later is recommended. A compatible program, for example LibreOffice Calc, should also work.
3. The following Python modules should be installed, if they are not already:
    a. itertools
    b. numpy
    c. xlsxwriter
    d. xlrd
4. The following files should be placed in the same folder:
    a. Wirt_Hm_Suite_Master.py
    b. gauss_processor.py
    c. excel_reader.py
    d. calc_wirt.py
    e. sym_hm.py
    f. d_hm.py
    g. d5_gen.py
    h. d4_gen.py
    i. D5_gens_pruned.txt
    j. D4_gens_pruned.txt

General Usage

Wirt_Hm_Suite_Master.py should be run. The following is an explanation of menu choices:

1. **Compute Wirtinger Number:** Computes the Wirtinger number of a knot using the calc_wirt.py algorithm.
2. **Search for a surjective homomorphism to a symmetric group:** Computes the Wirtinger number *n* of a knot and carries out, if possible, an exhaustive search for a surjective homomorphism to the S_{n+1} symmetric group using the sym_hm.py algorithm
3. **Search for a surjective homomorphism to a D group:** Computes the Wirtinger number *n* of a knot and carries out, if possible, an exhaustive search for a surjective homomorphism to the D_n symmetric group using the d_hm.py algorithm
4. **Quit:** Terminates the program. Does not exit Python.

The following is an explanation of input options:

1. **Input Gauss code of a single knot:** Accepts as input a list of positive and negative integers giving the Gauss code of the knot. The integers must be separated by non-numeric characters, but beyond this the choice of separation characters and the presence or absence of non-numeric leading or trailing characters is irrelevant. (example valid gauss codes: 1 -2 3 -4 5 -6 2 -1 6 -3 4 -5 ; {1,-2,3,-4,5,-6,2,-1,6,-3,4,-5} ; put1w-2ha3t-4ever5y-6o2u-1w6a-3nt4h-5ere!)
2. **Input Excel file:** Accepts the path to a .xlsx file. Ensure that the name and Gauss code of each knot you wish to process are present in the file, that all names are placed in the same column and all codes are placed in the same column, and that there are no empty rows between the first row of knot data and the last row of knot data. The program will scan through each row and run the desired algorithm on each knot in succession.
    a. **Enter the input file path:** Enter the path to the Excel file containing the knot data (e.g. C:\Users\Me\InputFiles\knotList.xlsx)
    b. **Number of the column containing knot names:** Enter the name of the column in the input spreadsheet, converted to a number as indicated in the prompt, containing the names of the knots to be processed. The names can be any combination of characters.
    c. **Number of the column containing gauss code:** Enter the name of the column in the input spreadsheet, converted to a number as indicated in the prompt, containing the Gauss code of the knots to be processed. The Gauss code should be formatted as described in the above "Input Gauss code of a single knot" explanation.
    d. **Number of the row containing the first knot's information:** Enter the number of the row in which the first knot is listed.
    e. **Name of excel output file:** Enter the desired location and name of the Excel file which will be generated to contain the results of the computation. (e.g. C:\Users\Me\OutputFiles\homomorphismList.xlsx)
    f. **Wirtinger number of the knots you wish to analyze:** When searching for homomorphisms to a D_n group, only knots of one Wirtinger number can be processed at a time. Knots with Wirtinger number greater than five cannot be processed. Knots with Wirtinger number less than three should instead be processed using the symmetric group module, as D_n is isomorphic to S_{n+1} for $n<4$.

The following is an explanation of outputs:

1. For single-knot entries:
    a. **Wirtinger number:** The Wirtinger number of the knot
    b. **Seed strand set used to color the knot:** The set of strands of the knot used as seeds to generate a complete Wirtinger coloring. The number of strands is equal to the Wirtinger number, and is the minimum number of seeds needed to completely color the knot. The labels given correspond to entries in the knot dictionary
    c. **Knot Dictionary:** A Python dictionary with strand labels as keys and two-element lists as entries. The first element of each list is the Gauss code segment corresponding to the strand. The second element is a list of two-tuples, one for each crossing for which this strand is the overstrand. The elements of the two-tuples are the understrands of the crossing.
    d. **Seed strand to symmetric group generating set mapping:** A Python dictionary in which the keys are seed strands of the knot (see above) and the entries are transpositions of the S_{n+1} symmetric group, where n is the Wirtinger number. Together, the n transpositions form a generating set of the group and satisfy the Wirtinger relations of the knot. This mapping thus provides a surjective homomorphism from the fundamental group of the knot to the S_{n+1} symmetric group.
    e. **Seed strand to d group generating set mapping:** A Python dictionary in which the keys are seed strands of the knot (see above) and the entries are matrix elements of the D_n group, where n is the Wirtinger number. Each matrix is represented as a list of its row vectors separated by a "|" symbol. Together, the n elements form a generating set of the group and satisfy the Wirtinger relations of the knot. This mapping thus provides a surjective homomorphism from the fundamental group of the knot to the D_n group.
2. For Excel entries: Output is an Excel file with the following format:
    a. **First column:** Knot names, same as those in the input file
    b. **Second column:** Gauss code of the knot
    c. **Third column:** Seed strands used to color the knot as described above. This is a Python dictionary, where the keys are strand labels and the entries are the Gauss code segments corresponding to the strands.
    d. **Fourth column:** Wirtinger number of the knot
    e. **Fifth column:** Empty if only the Wirtinger number is being computed. If a homomorphism search to the *D* or *S* group is being carried out, contains the mapping of seed strands to *D* or *S* group generators which gives a surjective homomorphism from the fundamental group of the knot to the group generated by the generators.

Module Documentation-Wirt_Hm_Suite_Master

Functions:

1. menu1(): Displays primary menu, offering choice between compyuting Wirtinger number only, searching for symmetric group homomorphisms, and searching for *D* group homomorphisms.
    a. Input: None
    b. Output: Integer from 1 to 4 corresponding to menu choice

2. menu2(): Gives user the choice to input a single knot's Gauss code or an Excel file, and collects the Gauss code or Excel directory.
    a. Input: None
    b. Output: Integer, either 1 or 2, corresponding to menu choice; and data, a string consisting either of the Guass code of the knot or the file name and directory of the input Excel file.
3. main(): Manages Wirt_Hm_Suite operations and displays results of other modules to the user.
    a. Input: None
    b. Output: None

Module Documentation-gauss_processor

Functions:

1. process_gauss_code(raw_gauss_code): Converts a string containing the Gauss code of a knot into a list of integers consisting of the same Gauss code.
    a. Input:
        i. raw_gauss_code: a string containing the Gauss code of a knot. Must be a series of positive and negative numbers each separated by one or more nonnumeric characters; nature of the separation characters and presence or absence of leading and trailing characters is irrelevant.
    b. Output:
        i. code: a list of positive and negative integers consisting of the Gauss code of a knot.

Module Documentation-excel_reader

Functions:

1. knot_processor(cur_row, knot_workbook, knot_table, name_col, gauss_col): Retrieves knot name and Gauss code from an Excel file.
    a. Input:
        i. cur_row: integer, the number of the row containing the name and Gauss code. Row 1 in Excel is row 0 in here.
        ii. knot_workbook: xlrd workbook object corresponding to the Excel file
        iii. knot_table: xlrd worksheet object corresponding to the worksheet in the Excel file containing the knot name and Gauss code
        iv. name_col: integer, the number of the column containing the knot name. Column A in Excel is column 0 here.
        v. gauss_col: integer, the number of the column containing the Gauss code. Column A in Excel is column 0 here.
    b. Output:
        i. name: string consisting of the name of the knot
        ii. raw_gauss_code: string containing the Gauss code of the knot and any number of separation, leading, and trailing characters.
2. excel_creator(excel_name, choice): Creates an Excel file to hold program output and conducts initial formatting operations.

a. Input:
    i. excel_name: string consisting of the desired name and directory of the output Excel file, including .xlsx suffix.
    ii. choice: Integer from 1 to 3 corresponding to the user's menu selection from menu1() in Wirt_Hm_Suite_Master. 1 = only return the Wirtinger number information of the knot. 2 = carry out an *S* group homomorphism search. 3 = carry out a *D* group homomorphism search. An extra column will be added to the output file to house the seed-strand-to-generating-set information for each knot if 2 or 3 is entered.

b. Output
    i. worksheet: xlsxwriter worksheet object consisting of the worksheet on which output data will be written. The first row will contain column headers (first column: knot name; second column: Gauss code; third column: seed strand set; fourth column: Wirtinger number; fifth column: empty or seed-strand-to-generating-set information). The remainder of the worksheet will be empty.
    ii. workbook: xlsxwriter workbook object consisting of the output Excel file.

3. excel_writer(name, seed_strand_with_gauss, knot_dict, raw_gauss_code, wirt_num, sym_gen_set, Knot_Number, worksheet, workbook, choice): Writes data for a processed knot to the output Excel file.
    a. Input:
        i. name: string consisting of the knot's name, as provided in the input Excel file.
        ii. seed_strand_with_gauss: dictionary with seed strand labels (strings) as keys and Gauss code segments (tuples of integers) corresponding to the strands as entries.
        iii. Knot_dict: dictionary with strand labels (strings) as keys and two-element lists as entries. The first element of each list is the Gauss code segment (tuple of integers) corresponding to the strand. The second element is a list of two-tuples, one for each crossing for which this strand is the overstrand. The elements of the two-tuples (strings) are the understrands of the crossing.
        iv. raw_gauss_code: string consisting of Gauss code of the knot as it was written in the input Excel file.
        v. wirt_num: integer equal to the Wirtinger number of the knot.
        vi. sym_gen_set: either an empty list if only the Wirtinger information of the knots is being computed or a dictionary with seed strand labels (strings) as keys and *S* or *D* group generators as entries.
        vii. Knot_Number: integer corresponding to the row of the output worksheet in which the knot information will be written. Row 1 in Excel is row 0 in here.
        viii. worksheet: xlsxwriter worksheet object consisting of the worksheet on which output data will be written. The name will be written in the first column, the raw_gauss_code in the second, the seed_strand_with_gauss in the third, the wirt_num in the fourth, and the sym_gen_set in the fifth.
        ix. workbook: xlsxwriter workbook object consisting of the output Excel file.
        x. choice: Integer from 1 to 3 corresponding to the user's menu selection from menu1() in Wirt_Hm_Suite_Master. 1 = only return the Wirtinger number information of the knot. 2 = carry out an *S* group homomorphism search. 3 = carry out a *D* group homomorphism search.
    b. Output: None

4. excel_main(input_name, choice): Gathers parameters for input Excel file, scans through input file, and calls other modules to gather data to insert into excel file.
    a. Input:
        i. input_name: file name and directory of the input Excel file.
        ii. choice: Integer from 1 to 3 corresponding to the user's menu selection from menu1() in Wirt_Hm_Suite_Master. 1 = only return the Wirtinger number information of the knot. 2 = carry out an *S* group homomorphism search. 3 = carry out a *D* group homomorphism search.
    b. Output: None

Module Documentation-calc_wirt

Functions:

1. create_knot_dictionary(gauss_code): calls functions building the knot dictionary for a knot.
    a. Input:
        i. gauss_code: list of integers consisting of the Gauss code of the knot.
    b. Output:
        i. knot_dictionary: dictionary with strand labels (strings) as keys and two-element lists as entries. The first element of each list is the Gauss code segment (tuple of integers) corresponding to the strand. The second element is a list of two-tuples, one for each crossing for which this strand is the overstrand. The elements of the two-tuples (strings) are the understrands of the crossing.
2. find_strands(gauss_code): Scans through Gauss code and extracts all segments corresponding to strands of the knot, giving each a label.
    a. Input:
        i. gauss_code: list of integers consisting of the Gauss code of the knot.
    b. Output:
        i. strands_dict: dictionary with strand labels (strings) as keys and two-element lists as entries. The first element in each list is the Gauss code segment (tuple of integers) corresponding to the strand. The second element is an empty list.
3. find_crossings(knot_dict, gauss_code): Determines which strand serves as the overstrand and which serve as the understrands at each crossing of the knot, and generates the knot_dict.
    a. Input:
        i. knot_dict: Actually the strands_dict returned as output from find_strands(); dictionary with strand labels (strings) as keys and two-element lists as entries. The first element in each list is the Gauss code segment (tuple of integers) corresponding to the strand. The second element is an empty list.
    b. Output:
        i. knot_dict: Complete version, populated with crossing information; dictionary with strand labels (strings) as keys and two-element lists as entries. The first element of each list is the Gauss code segment (tuple of integers) corresponding to the strand. The second element is a list of two-tuples, one for each crossing for which this strand is the overstrand. The elements of the two-tuples (strings) are the understrands of the crossing.
4. is_valid_coloring(seed_strands, knot_dict): Checks whether a given set of seed strands provide a complete coloring of the knot as per the Wirtinger coloring algorithm.
    a. Input:

        i. seed_strands: itertools combination object (also accepts lists, sets, and tuples) consisting of strand labels from the knot dictionary. The number of elements is greater than or equal to the Wirtinger number of the knot **if** the is_valid_coloring function returns True.
        ii. knot_dict: dictionary with strand labels (strings) as keys and two-element lists as entries. The first element of each list is the Gauss code segment (tuple of integers) corresponding to the strand. The second element is a list of two-tuples, one for each crossing for which this strand is the overstrand. The elements of the two-tuples (strings) are the understrands of the crossing.
    b. Output:
        i. Returns True if the seed_strands set results in a complete coloring of the knot given by knot_dict as per the Wirtinger coloring algorithm. Otherwise, returns False.

5. calc_wirt_info(knot_dict): Finds a minimal seed strand set to completely color a knot. Generates all possible seed strand sets of length two, then of length three, and so on until is_valid_coloring returns True or all sets of length less than the number of strands in the knot have been tried. The length of the seed strand set that causes is_valid_coloring to return True is equal to the Wirtinger number of the knot.
    a. Input:
        i. knot_dict: dictionary with strand labels (strings) as keys and two-element lists as entries. The first element of each list is the Gauss code segment (tuple of integers) corresponding to the strand. The second element is a list of two-tuples, one for each crossing for which this strand is the overstrand. The elements of the two-tuples (strings) are the understrands of the crossing.
    b. Output:
        i. Two-tuple consisting of the first seed strand set which completely colored the knot and the length of the set, which is equal to the Wirtinger number. If is_valid_coloring never returned true for any seed strand set of length less than the number of strands in the knot diagram, then the function will return all strands as a seed strand set and a Wirtinger number equal to the number of strands in the diagram.
6. wirt_main(gauss_code): Creates knot dictionary and calculates the Wirtinger number of a knot with the entered Gauss code.
    a. Input:
        i. gauss_code: a list of positive and negative integers consisting of the Gauss code of a knot.
    b. Output:
        i. knot_dict: dictionary with strand labels (strings) as keys and two-element lists as entries. The first element of each list is the Gauss code segment (tuple of integers) corresponding to the strand. The second element is a list of two-tuples, one for each crossing for which this strand is the overstrand. The elements of the two-tuples (strings) are the understrands of the crossing.
        ii. seed_strand_set: set of strand labels (strings) which completely color the knot.
        iii. wirt_num: positive integer, the Wirtinger number of the knot.

Documentation for remaining modules in progress

Known Limitations

1. Attempting to input knots with more than 702 strands will lead to a fatal error and/or nonsense output. The current calc_wirt.py algorithm cannot handle more than 702 strands as it runs out of strand labels. With a minimal knowledge of Python, this number may be increased by editing the find_strands() function.
2. Symmetric groups of order greater than 6 and D groups of greater than 5 are not implemented. Thus, no search for surjective homomorphisms can be carried out on knots with Wirtinger number greater than or equal to 6.

---

[i] R. Kirby, Problems in low-dimensional topology, Proceedings of Georgia Topology Conference, Part 2 (R. Kirby, ed.), Citeseer, 1995.
[ii] R. Blair, A. Kjuchukova, R. Velazquez, and P. Villanueva, Wirtinger systems of generators of knot groups, Communications in Analysis and Geometry (In press - arXiv:1705.03108).