Lie Algebra Documentation

July 2023

Contents

1 Introduction

2 How to get things “working" on "Windows"

Warning: currently this method only finds centralizers and normalizers. This will soon be changed where instead you pass arguments to tell the executable what should be computed.

1. Install WSL (Windows Subsystem for Linux) here. You want to install Debian. This can be done by running the following line in PowerShell (which can be opened by pressing the Windows key, typing "powershell" and pressing Enter):

```
wsl --install -d Debian
```

2. You will need to make a username and password (note: these have to be all lowercase). It’s fine to use something simple for both. Open WSL; you can do this by typing "Debian" in the Windows program search and then pressing Enter.

3. Restart your computer at this point.

4. Run the following command (this is just one long command, copy it all together then press enter) in Debian:

```
sudo tee /etc/apt/sources.list<<EOF
deb http://deb.debian.org/debian bullseye main deb-src http://deb.debian.org/debian bullseye m
deb http://security.debian.org/debian-security bullseye-security main
deb-src http://security.debian.org/debian-security bullseye-security main

deb http://deb.debian.org/debian bullseye-updates main
deb-src http://deb.debian.org/debian bullseye-updates main

deb http://deb.debian.org/debian bullseye-backports main
deb-src http://deb.debian.org/debian bullseye-backports main
EOF
```

   Then run the following two commands (one at a time)

```
sudo apt-get update
sudo apt-get install git g++ make wget libginac-dev libcln-dev libgmp-dev
```

   You will need to input your password and type `Y` when prompted.

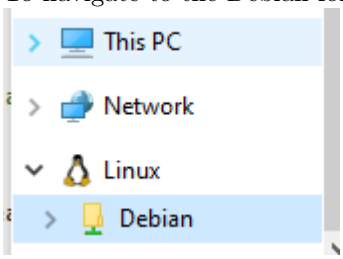
5. Run the following commands by copying each line and then running them (one at a time):

```
mkdir JoeCalc
cd JoeCalc
wget https://github.com/EthanXDX/JosephRepkasSpecialMatrixCalculator/raw/main/joemat.o
touch matrixInput.txt
sudo chmod +x joemat.o
```

   The last step involves entering your password, so make sure you have it copied down.

6. If anything went wrong up to this point, ask for help.

7. To navigate to the Debian folders you will have in File Explorer an option "Linux" at the bottom.



8. Then you can navigate to `Linux/Debian/home/YOUR-USERNAME/JoeCalc`. You should create a shortcut to this folder from your Desktop as the output csv file will be made in this folder.

9. Now to run the calculator:

   (a) Open matrixInput.txt (from windows is fine) and copy paste your desired lie algebra into the text file in the format you get from the gui. (Make sure to save the file)

   (b) Type and enter `~/JoeCalc/joemat.o` into your debian terminal. Alternatively, navigate to the folder (`JoeCalc`) containing `joemat.o` in the debian console, then run `./joemat.o`

10. Extra instructions, to be expanded upon later.

    (a) In the (near) future you will be able to have a concise version of these instructions by adding the `--help` flag as in `./joemat.o --help`.

    (b) If you want to time the execution of the program, then you have to be in the `JoeCalc` directory in your Debian terminal and enter `time -p ./joemat.o`

    (c) The `-i` flag allows you to specify the input file, otherwise it will just use the file named matrixInput.txt, for instance, we may use the command `./joemat.o -i input.txt` to run the program on the file input.txt

(d) The `-o` flag allows you to specify the output file name, analogous to the previous `-i` flag. In order for windows to know how to work with the output, you should include the `.csv` extension in the output name. For instance, we can run the command `./joemat.o -i input.txt -o algebraInfo.csv` which will save the output data to a csv file named algebraInfo.csv

(e) The `--derived` flag computes the derived and all invariants (except the minimum rank) associated with the elements of the series.

(f) The `--lower` flag computes the lower central series and all invariants (except the minimum rank) associated with the elements of the series.

(g) The `--include-basis` flag includes a basis generating set provided in the input file in the output file.

(h) The `--include-inputs` flag includes the generating set provided in the input file in the output file.

(i) The `--compute-all-min-rank` flag computes the minimum rank of all algebras computed. THIS IS NOT PERFORMANT CURRENTLY.

(j) The `--compute-min-rank` flag includes the min rank of only the inputted algebras. THIS IS NOT PERFORMANT CURRENTLY.

(k) * NOTE: the flag can be put in in any order, the only requirement is that the name of the input/output files directly follow the `-i` or `-o` flags.

3 How to get things working on Windows (if you want more control)

1. Install WSL (Windows Subsystem for Linux) here. You want to install Debian. This can be done by running the following line in PowerShell (which can be opened by pressing the Windows key, typing "powershell" and pressing Enter):

```
wsl --install -d Debian
```

2. You will need to make a username and password (note: these have to be all lowercase). It's fine to use something simple for both. Open WSL; you can do this by typing "Debian" in the Windows program search and then pressing Enter.

3. Run the following command in Debian:

```
sudo apt-get install git g++ make wget libginac-dev libcln-dev libgmp-dev
```

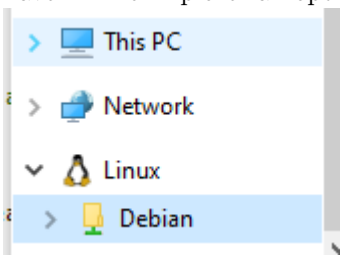
   You will need to input your password and then type `Y` when prompted.

4. Run the following commands by copying each line and then running them (one at a time):

```
git clone https://github.com/Bortoise/Joe-s-Special-Matrix-Calc.git
cd Joe-s-Special-Matrix-Calc/
make all
```

5. If anything went wrong up to this point, ask for help.

6. To navigate to the source files (which you may have to edit to run your examples for the time being) you will have in File Explorer an option "Linux" at the bottom.



7. Then you can navigate to `Linux/Debian/home/YOUR-USERNAME/Joe-s-Special-Matrix-Calc`. You may want to create a shortcut to this folder from your Desktop.

8. Now navigate to `Joe-s-Special-Matrix-Calc/src/` in File Explorer.

9. You can open `example.cpp` in your code editor of choice (VS Code is fine) and write any code you want in the `main()` function. There are (non-extensive) examples of how to use the library in the "example" function of `example.cpp`.

10. To run any code you write you just type the following two commands into the Debian terminal while you are in the "Joe-s-Special-Matrix-Calc" directory

```
make example
./out/example.o
```

    Note: you will have to wait for `make example`, since it can take a little bit to finish. You know it is done when a new blank line is printed.

11. You will have to print anything you want to manually. This can be done as in the example code in `example.cpp` by default.

4 How to get things working on macOS

1. Download the executable `joemat` (no file extension) from Slack.

2. Open Finder.app and navigate to the folder where you wish to run the code (i.e., in which you will place `matrixInput.txt` and output `output.csv`. Drag `joemat` into this folder.

3. Ensure that you have created (or dragged in) the input text file `matrixInput.txt` into this folder.

4. Open Terminal.app and navigate to the directory from the last step. Tip: You can do this by typing `cd` and dragging the above directory from Finder into the terminal window.

5. Type `./joemat` and press Enter. (If you encounter a permissions issue, run `sudo chmod +x joemat`. This allows you to run `joemat` as an executable. You may have to enter your password.)

6. After the program finishes running, you should see a new file `output.csv` in the same folder. It will have the computed basis, dimension, and normalizer and centralizer with their dimensions.

You can also customize the name of the input text file and output csv file with the `-i` and `-o` options. For instance, say you have written your generators in `L_5_6a.txt` and wish to output the invariants to `L_5_6a.csv`. In this case, run `./joemat -i L_5_6a.txt -o L_5_6a.csv`.

4.1 Troubleshooting

4.1.1 Library not loaded

If get receive an output like `Library not loaded`, it is possibly because you do not have C++ installed on your computer. To fix this, either install all of Xcode or simply install the Xcode command line tools with

```
xcode-select -install
```

5 How to use the C++ code in MATLAB on Linux or macOS

If you are using Linux or macOS, it is (currently) possible to run our C++ code through ordinary MATLAB functions. More technically inclined users might want to build the interface themselves and read about MATLAB interfaces to C++ libraries.

This section describes how to get things working on macOS. The process is almost the same on Linux, except that files with the `.dylib` extension will instead end with `.so`.

5.1 Installing the MATLAB interface

1. Download `joemat-interface-macos.zip` from here.
2. Unzip it.
3. Copy and paste everything into `MatrixCalc/`.

The resulting folder structure should look something like this:

```
MatrixCalc/
|
 ---bin/
|   |   libjoemat.dylib
|
|   joematInterface.dylib
|   getLieAlgebraBasis.m
|   getLieAlgebraCentralizer.m
|   getLieAlgebraDim.m
|   getLieAlgebraNormalizer.m
|   matrixSeqToString.m
|   matrixToString.m
|   stringToMatrix.m
|   stringToMatrixSeq.m
|   (+ all the original files and folders)
```

5.2 How to use the functions in MATLAB (basic)

Our functions are mostly drop-in replacements for the old MATLAB functions (e.g., `getLieAlgebraNormalizer.m` pretty much does what `NormalizerV2.m` did). The main exception is that `getLieAlgebraCentralizer.m` returns a basis for the centralizer rather than a matrix with parameters.

To be precise, here's what each function takes as input and returns as output. Below, `matrixSeq` indicates that an object is a MATLAB array of n-by-n matrices (e.g., a list of generators or a basis).

- `getLieAlgebraBasis(matrixSeq generators) -> matrixSeq`: takes an array of matrices and returns a basis for the generated Lie algebra
- `getLieAlgebraDim(matrixSeq generators) -> int`: takes an array of matrices and returns the dimension of the generated Lie algebra
- `getLieAlgebraCentralizer(matrixSeq generators) -> matrixSeq`: takes an array of matrices and returns a basis for the centralizer of the generated Lie algebra
- `getLieAlgebraNormalizer(matrixSeq generators) -> matrixSeq`: takes an array of matrices and returns a basis for the normalizer of the generated Lie algebra

For instance, this is how one might adapt `matrixInputTextFile.m` to use the C++ code for computing the normalizer. The original code reads

```
normBasis = NormalizerV2(algebrasWInvariants{i,1});
```

(or something like that). New code might look like

```
normBasis = getLieAlgebraNormalizer(algebrasWInvariants{i,1});
```

and should generate a similar result.

5.2.1 Troubleshooting

You might encounter an error similar to the following:

```
Unable to load interface library: '/path/to/MATLAB/MatrixCalc/joematInterface.dylib'.
Reason: The specified module could not be found.
Ensure the C++ dependent libraries for the interface library are added to run-time path.
```

This is because MATLAB can't find the library `libjoemat.dylib` in the run-time path. To fix this, right click the `bin/` folder and click `Add to Path > Current Folder`.

5.3 How to use the functions in MATLAB (more advanced)

Warning: You don't really need to read any of the following unless you are interested in understanding how everything works or you have expanded on the original C++ library.

Technically, all the functionality from our library is exposed through just `libjoemat.dylib` and `joematInterface.dylib`. Unfortunately, this code alone is still hard to use in MATLAB, because the exposed functions only take and return *string representations* of sequences of matrices rather than actual MATLAB arrays of matrices. The functions `matrixSeqToString.m`, `matrixToString.m`, `stringToMatrix.m`, and `stringToMatrixSeq.m` bridge this gap, allowing for easy conversion between MATLAB objects and their string representations.

This is best shown by example. Consider `getLieAlgebraBasis.m`:

```
function basis = getLieAlgebraBasis(generators)

generatorsString = matrixSeqToString(generators)
clib.joemat.setLieAlgebra(generatorsString)
basisString = clib.joemat.getLieAlgebraBasis()

basis = stringToMatrixSeq(basisString)
```

First, the MATLAB array of generators is turned into its string representation so that it can be read by 'clib.joemat'. Now we do the computation via the C++ library by first setting the Lie algebra and then getting its basis. Finally, we turn the basis string representation into a MATLAB array.

But what is a string representation?

5.3.1 String representations

String representations of matrices, lists of matrices, etc. are pretty much the same as the text format that is currently used in `matrixInput.txt`. This representation **should** satisfy the following to be processed correctly:

- each matrix must be surrounded by a pair of square brackets (`"["` and `"]"`);
- a semicolon (`";"`) separates every two rows of the same matrix;
- a space (`" "`) separates every two elements of the same row;
- a newline (`"\n"` in C++ or `newline` in MATLAB) separates every two matrices in the basis for the same Lie algebra;
- and there are no additional characters (including whitespace or newlines).

There is some tolerance for additional whitespace and newlines, but we make no guarantee. Check first for possible extra characters if you encounter any problems doing string-MATLAB conversions.

As an example, here is a correct string representation of the standard basis for $\mathfrak{gl}(2,\mathbb{C})$, followed by the standard basis for $\mathfrak{sl}(2,\mathbb{C})$:

```
[1 0;0 0]
[0 1;0 0]
[0 0;1 0]
[0 0;0 1]
@
[0 1;0 0]
[0 0;1 0]
[1 0;0 -1]
```

5.3.2 Functionality available in the interface

A few sections ago, we highlighted `clib.joemat.getLieAlgebraBasis()`. Here are all five directly available functions:

- `clib.joemat.setLieAlgebra(str generatorsString) -> (nothing)`: set the working Lie algebra to the one generated by the generators in the provided string representation
- `clib.joemat.getLieAlgebraBasis() -> str`: get a string representation of a basis of the working Lie algebra (the last Lie algebra set by `clib.joemat.setLieAlgebra`)
- `clib.joemat.getLieAlgebraDim() -> num` get the dimension of the working Lie algebra
- `clib.joemat.getLieAlgebraCentralizer() -> str` get a string representation of a basis for the centralizer of the working Lie algebra
- `clib.joemat.getLieAlgebraNormalizer() -> str` get a string representation of a basis for the normalizer of the working Lie algebra

One benefit of directly accessing the interface is to avoid unnecessary string representation conversions. Below is an example which simply prints out a basis for the centralizer (`generators` is provided).

```
# First, convert to a string usable by ‘clib.joemat‘
generatorsString = matrixSeqToString(generators)

clib.joemat.setLieAlgebra(generatorsString)
# Get a string representation of the centralizer
centString = clib.joemat.getLieAlgebraCentralizer()

# Print it to the command window or to a text file
```

Using `getLieAlgebraCentralizer()` would have done an unnecessary conversion of the centralizer string into a MATLAB array consisting of a basis for the centralizer.

6 C++ Library Specification (WIP)

The part of the C++ Library which handles computations is mainly divided into two files: `lie_algebra.h` and `lin_alg.h`

6.1 lin_alg.h

`lin_alg.h` is structured as a colleciton of C++ functions under the namespace `lin_alg`. The functions in this namespace are desined to perform linear algebra operations on GiNaC matrices `g::matrix` and vectors `g::exvector`. Below is a brief desciption of each method. In this file `mat_vec` is an alias for `std::vector<g::matrix>`

- `g::matrix bracket(g::matrix &m, g::matrix&n)`: returns the Lie bracket of the matrices `m` and `n`.
- `mat_vec spanning_subsequence(mat_vec matrices)`: returns a subsequence of `matrices` which is also a basis for the linear span of `matrices`.
- `matrix_to_vector_in_basis(g::matrix &m, mat_vec &basis) -> g::exvector` returns a vector v which represents the matrix `m` in the basis `basis`
- `void swap_rows(g::matrix &m, int i, int j)` mutates the matrix `m` by swapping the `i`th row and the `j`th row. If `i` or `j` are out of range, then the program segfaults.
- `gaussian_elimination(g::matrix &m)`

6.2 lie_algebra.h