CINF - Computation Is Not Fun

About the Project

CINF (a.k.a. Computation Is Not Fun) is a course project for COMP2012H, fall 2020. It is a computation tools built with Qt which can automatically evaluate the math expressions given by the users. Here lists the main features:

- Built-in math library `R` which can do evaluations with parameters, demonstrate step-by-step solutions, and providing exact solutions without losing precision.
- OCR function to let you simply scan your formulae by screenshots, saves your time from manually input.
- A built-in parser which can handle very general input with a human-friendly format AsciiMath, you can even embed a matrix in another (if the result is reasonable).
- A third-party linear algebra library Armadillo as a supplement of `R`, which includes support for decimals, decompositions and many other common elementary functions & linear algebra operations
- Basic symbolic computation and variable assignment so that you can traverse between results easily, these variables can be reused directly later in computations by just type in their name in the expression.
- A sleek and elegant rendering of all the math expressions based on MathJax.

Get Started

Troubleshooting Guide for Graders

The project uses `QWebEngine` class for rendering math expressions, therefore a Qt with MSVC correctly setup is required for compilation.

Aside with that, all the other dependencies should be in place, and you can open the cinf_gui.pro under `/gui` and run it directly. If there does exist some problems with linking or compilation, please contact us.

In case you want to test the OCR function, we attached a separate file with API key and id in the submitted files. Internet is required for both the OCR and the rendering of the math expressions since we use the MathJax hosted scripts.

Compilation - Console version Windows x64 (with some finite fields supported)

- Requires MinGW 64
- MinGW64 needs to be run in `64bits` (of course)
- Requires GNU Make
- We used `MinGW64 g++ 8.1.0`, and `GNU Make 4.3`

```
mkdir compile
cd sources
make -f makefile
cd ..
cd compile
run
```

Note that the console requires to press Ctrl+C to quit. Alternatively it is possible to use `build_console_win64.bat`. The program will be compiled inside the compile folder, namely `compile/run.exe`

Compilation - Console version MacOSX (with some finite fields supported)

```
mkdir compile
cd sources
make -f makefile_osx
cd ..
cd compile
run
```

Compilation - GUI version Windows x64

- Requires QT Creator latest version (`5.12.10`).
- Requires QT WebEngine
- Requires MSVC x64 compiler `VS 2015,2017,2019, Windows Kits`
- Open the file `cinf_gui.pro` in the `GUI` folder.
- Choose `MSVC x64 compiler` as build configuration

QT WebEngine installation

- Modify QT using `MaintenanceTool.exe`
- It is in the QT installation folder, e.g. `C:\Qt\Qt5.12.10`

Compilation - GUI version MacOSX

- Requires QT Creator latest version `5.12.10`
- Requires QT WebEngine
- Seems to work on the default MacOSX compiler `clang`.
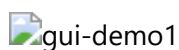- If it doesn't compile, try `64bit` and `32bit`.

Documentation for the Internal Math Library R

Please refer to the well written MathLibDocs.pdf. Source code of this part is under `/sources/math` and `/sources/steps`

Documentation & Users' Guide for GUI Part

1. Input api key

- input api key (for mathpix) to enable OCR

gui-demo1

- choose "Do not use ocr" to skip

2. Scan

- the button works only when OCR is enabled
- click the button to begin screen capture
- during screen capture, specify the area which the user would like to emphasize
- after screen capture, check the interpretation in "interpretation" and the original image in "original image". If the interpretation is incorrect, change it through the textedit window.

3. Input matrix

- If ocr is not enabled, use Asciimath to input matrix through the textedit window

4. Calculation

- choose the type of computation
- click "compute" to start calculation
- sometimes the calculation is slow, user may check on the progress bar at the bottom of the page to make sure the progress is still on
- click "next" or "previous" to goto the next or previous step
- click "Jump to answer" to jump to final step
- click "save" to save current calculation to history
- find previous calculations in "History", click to jump to the chosen one

gui-demo2
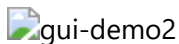
- right click the solution window to copy the solution to the clip board

Docs for Parser, Evaluator & OCR

For Graders: A Quick Overview of the Parser & Evaluator Part

This part is an introduction to the constituent classes for graders and people who are interested. For users, check below for users' guide.

This part of the application handles the OCR request, interpertation of the input(either form OCR or manual input), and also the evaluation. **It is just a simple summary here, for more detailed information, please go and check the source code of each files.**

It consists of the following classes:

- `/sources/parser/tokens.h`: The base class `Token`, which is the basis of the lexer and the parser. It is a data type to store the properties and raw values of each token. It has several derived classes representing different token types.

- `/sources/parser/lexer.h`: The class `Lexer` will handle raw input and tokenize it into a token stream. It uses regex to classify each token

- `/sources/utils/math_wrapper.h`: Contains two classes `ROperand` and `ArmaOperand`, the former is a wrapper for our own `R` class so as to unify matrices and scalars as a single data type to facilitate development. It also overload some operators. The `ArmaOperand` is more or less the same. The rest of the header file also includes some handy inline functions for instantiating `R` objects.

- `/sources/parser/expr_ast.h`: The classes related to the abstract syntax tree used in the parser. It contains a base class `ExprAST` and several derived classes, each representing different kinds of node on the AST. The member funtions of each derived classes also handles the evaluation of its own type of tree node. As a result, the whole AST can be constructed, destructed, or evaluated with ease by traversing through the tree.

- In addition, the `expr_ast.h` also contains an `Info` struct for the communications between the parser and the GUI.

- `/sources/parser/parser.h`: Core part of the parser. It includes a giant class `Parser`. It is basically an LL(1) parser with some features of the LL(*) parser. It is also shipped with some dedicated parser components for special cases, variable assignment and management functions, the driver functions for evaluation, and a whole bunch of exception handling.

- `/sources/utils/ocr_api.h`: The `Ocr` class encapsulates the OCR request functions. It manages the credentials, converts the image input, constructs the HTTP requests, does the post and parse the response. Finally it returns the OCR results in LaTeX and AsciiMath format.

For more information, please check the readme under `/sources/parser` as well as the comment in source code.

Use the OCR Function

You can use a screen capture as your input if OCR is enabled.

If you want to use the OCR function, prepare youself with an account of Mathpix Snip. You can find more information about how to get the API key on their official website.

If you do have one, you can simply input them at the login interface. Note that this application will NOT store any of your credentials.

Manual Input

If you do not have the access to OCR function, you can always manually input the expressions.

Input Conventions

The input format largely follows the AsciiMath syntax. Here is a quick summary of the major syntax

- basic binary operators:
  - +,- remain unchanged
  - single asterisk "*" is dot product: .
  - two asterisk " ** " will be asterisk $*$ in AsciiMath
  - "xx" is cross product: $\times$
  - double slash "//" is division: /
  - single slash "/" is now fraction i.e. `/frac{}{}` in LaTeX
  - superscript ^ remains unchanged
  - "%" is parsed but will be treated as ERROR during calculations
  - assume multiplication between two adjacent operands: $xy = x * y$
  - division or fraction is only effective on the next operand: $3 / xy$ will be interpreted as $3 / x * y$

- matrices
  - matrices: [[a, b], [c, d]] now yields to $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$
  - round brackets matrices: ([a, b], [c , d]) now yields to $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$
  - "{}" are interchangeable with "[]": {{a, b}, {c ,d}} has the same effect as [[a, b], {c, d}]
  - however, note that the inner rows of the matrix MUST be enclosed in square brackets, each row should have the same number of entries, seperated by commas
- functions
  - please refer to the evaluation sector or the full token list
- constants, numbers and special symbols
  - π, e, i have their usual meaning, and I is indentity matrix as usual
  - you can use newline to separate your linear equations in linear system mode, in other modes the newline has the same effect as any other whitespace characters
  - you can use integers and decimals with scientific notation: $-1.34e-5, 23E6$ are valid. As convention, please do not put decimal at the exponential part.
- variables and greek letters
  - you can use any alphabet characters or words with any length of subscripts as the name of the variables, as long as they are NOT used as a reserved word for functions of constants.
  - example: $x, X, variable, my_{variable_x}, x_1, a_{1_2}, X_\alpha$ are all valid variable names
  - we support some but not all greek letters: $\alpha\beta\theta\lambda\mu\phi\varphi\omega$, you can simply use their name to represent them, or check the token list if you like. Note that their capital forms are not supported
  - you can add single quotes "'" to your variables, but they must appear before subscripts, if any

Calculation

Overview

For a more detailed internal documentation of the R library and its comparison with Armadillo, please refer to MathLibDocs.pdf

The evaluation of the input expression is based on two math libraries, one is designed by ourselves and the other is an opensource linear algebra library Armadillo.

The two libraries focus on different scenarios. Our R library aims to provide support for abstract operations, therefore it can handle polynomials and matrices with unknown parameters and perform linear operations on them. It has no precision issues and can give exact solutions. R also provides step-by-step functions for linear algebra operations so that users can have a better grasp of the computation if they are linear algebra learners.

On the other hand, Armadillo is shipped with a whole bunch of linear operations including decompositions, factorisations, inverses and equation solvers with high performance. It support basically all common elementary functions but as a result is limited to the floating point number precision. It also cannot demonstrate the intermediate steps if used out-of-box.

Mode Selection

There are four general modes of calculation. You can either choose to let the application automatically detect which computational engine to be used, or you can force using one of them.

In addition, there is a special linear system mode designed to facilitate the input of linear systems.

Auto Dectection

The automatic dectection follows these rules:

- if decimals or scientific constants are detected, Armadillo will be a prior choice
- if variables are found, R will be used
- if functions that R does not support occur, Armadillo will be used

However,

- if the number of variables is greater than 1, both engines will return error. You may need to assign values to redundant variables
- if rules above collide, you need to modify your input or force the application to use on of the engine
- R class can deal with some floating point numbers by converting them into fractions but with compromised precision
- if scientific notation occurs and is beyond the range of built-in `long` type, neither engine will be able handle it. You will see a related error message printed on the screen
- without scientific notation, R will use `mpz_wrapper` to handle integers but Armadillo is limited to `double` and `long`

It is possible that after explicitly choosing one of the engines, error still occurs. You may want to check the error message and correct the input or check the supported operation list to see whether the operation is supported.

Dedicated Linear Systems Mode

Since converting linear systems to augmented matrices is not as natural as input the equations directly, there is a linear system mode provided.

You can input or scan a linear system using this mode, but there are a few restrictions:

- make sure each linear equation starts a newline or separated by "\"
- each variable should be separated, i.e. not bounded by a same pair of parentheses, e.g. $3x + (4 - 5i)y + 6z(7 * 8/9) = 10a - 42$ is a valid linear equation
- however, $3(x + y + z) = 0$ is not, although it is linear
- variable name "t" is reserved for representing the parameter: $tx = 1$ will give a solution $x = 1/t$

The answer will be given with corresponding variable names arranged in ascending order, which may be different from their order in the linear system. A hint for the ordering of the variables in the output will also be given in the interpretation of the AsciiMath input so that you can figure out which variable the column of the augmented matrix in the step-by-step history corresponds to.

Supported Operations

General Operations supported by both

- $+ \ - \ * \ \cdot \ \times \ \wedge$, note that currently $* \ \cdot \ \times$ are not distinguished, and exponential only supports natural numbers in R, complex numbers in Armadillo
- currently we do not support superscript with special meanings e.g. $A^{-1} \ B^* \ C^T \ D^+$. It may be added soon, but so far superscript is only for exponentiation

Linear Operations supported by R with step-by-step

- rref(A): row reduce
- inv(A): inverse
- det(A): determinant
- orth(A): orthogonalization
- solve(A): solve augmented matrix
- charpoly(A): find characteristic polynomial

Functions and Linear Operations in Armadillo

- trigonometry: $\sin(x) \ \cos(x) \ \tan(x) \ \arcsin(x) \ \arccos(x) \ \arctan(x)$
- $\exp(x) \ \ln(x) \ \sqrt{x} \ \sqrt[a]{x}$
- rref(A): row reduce
- orth(A): give the orthnormal basis of the column space of a given matrix
- ker(A): find orthonomal basis of the kernel of given matrix
- trans(A): matrix transpose
- norm(A): norm of a vector/matrix[*]
- det(A): determinant
- trace(A) or tr(A): trace
- inv(A): inverse
- pinv(A): pseudo-inverse
- solve(A, B(optional)): solve[^] an augmented matrix A or a linear system Ax=B
- eigen(A): eigen decomposition
- schur(A: Schur decomposition
- qr(A): QR decomposition
- svd(A): singular value decomposition

[*]: p = 2 for both vectors and matrices

[^]: if no exact solution is found, Armadillo will automatically attempt to find approximated solutions

Acknowledgements

- [GMP](#)
- [Armadillo](#)
- [MathJax](#)