Simpact Cyan

Simpact Cyan is the C++ version of the Simpact family
of programs. It is an agent based model (ABM) to study the way an infection
spreads and can be influenced, and is currently focused on HIV and HIV-HSV2 co-infection.

The main documentation can be found at
simpactcyan.readthedocs.io.

Source code packages and pre-compiled versions can be found
here.

Compiling Simpact Cyan on OS X

You need to have GSL installed,
as well as the TIFF library, and
CMake is needed to generate the build files.
While you can download the CMake tool from the website, and you
can compile and install the GSL and TIFF libraries manually, I find
it easier to use pre-compiled versions of these things, which I
made available using the Anaconda Python
environment. For these instructions, I'm going to assume that you have
installed Anaconda Python.

1. Download the Simpact Cyan source code and extract it somewhere, or clone
   the GitHub repository somewhere. Below, I'll assume that this has created
   a path called `/Users/me/simpactcyan`.

2. Open a Terminal window, and make sure that the Anaconda environment
   has been activated, i.e. that the `conda` command is recognized. If
   not, it may be necessary to update your `PATH` environment variable
   using a command like this

   ```
   export PATH=/Users/me/anaconda/bin:"$PATH"
   ```

   or run Anaconda's activation command:

   ```
   source /Users/me/anaconda/bin/activate
   ```

3. Execute the following commands to set up the build environment:

   ```
   conda install cmake
   conda install gcc
   ```

PROF

```
conda install -c jori libtiff
conda install -c jori gsl
```

These commands will make sure CMake is available, that the GNU C compiler (gcc) is available, and that precompiled versions of the TIFF and GSL libraries will be installed.

4. We need to create a directory where the build files will be stored, let's create a directory `/Users/me/simpactcyan/build` for this. In that directory, we'll call the `cmake` command, setting the CMake variable `CMAKE_INSTALL_PREFIX` to the Anaconda path (this is needed so that the TIFF and GSL libraries can be located correctly).

```
cmake /Users/me/simpactcyan -
DCMAKE_INSTALL_PREFIX=/Users/me/anaconda
```

5. When the CMake step is finished, you can now just run

```
make
```

to start building the Simpact Cyan executables, or use something like

```
make -j 4
```

to start four build jobs in parallel, which can speed up the process if your computer or laptop has several processor cores.

PROF

6. When the build is complete, running e.g. `./simpact-cyan-release` will most likely show an error message about a library not being loaded. To tell the system where the TIFF and GSL libraries can be found, execute

```
export DYLD_LIBRARY_PATH=/Users/me/anaconda/lib
```

Afterwards, running `./simpact-cyan-release` will show a usage message.

7. To make sure that this version is used from within an R session, execute the following commands **before** loading the `RSimpactCyan` package:

```
Sys.setenv(PATH=paste("/Users/me/simpactcyan/build",Sys.getenv("PAT
H"),sep=":"))
Sys.setenv(PYTHONPATH="/Users/me/simpactcyan/python")
```

```
Sys.setenv(SIMPACT_DATA_DIR="/Users/me/simpactcyan/data/")
Sys.setenv(DYLD_LIBRARY_PATH="/Users/me/anaconda/lib")
```

The first line makes sure that `RSimpactCyan` will be able to locate the compiled executables. The second line makes sure that the Python module that's stored in that directory will be found. The third line tells the library where the data files can be found. Note that the data directory needs to end with a single `/`. The last line makes sure that the executable can find the GSL and TIFF libraries which are stored in the Anaconda directory structure.

Only after these lines have been set can you load the library:

```
library("RSimpactCyan")
```

If you then use e.g. the `simpact.run` command, the newly compiled versions are used.

8. The previous step allows you to run the new executables from within R, but to debug the code this is not the most convenient way to run the program. Instead, first create a configuration file using e.g. the `dryrun` option of `simpact.run`. You can then just run the executable from the command line, as described in the manual, for example:

```
./simpact-cyan-debug /path/to/config.txt 0 opt
```

PROF

Compiling Simpact Cyan on GNU/Linux

Using your package management system, make sure to install the development versions of the GSL and TIFF libraries. For Debian or Ubuntu systems, the GSL package will be called something like `libgsl-dev` or `libgsl0-dev`, and the TIFF package will be `libtiff-dev`, `libtiff4-dev` or `libtiff5-dev`. For Fedora based systems, the packages will be `gsl-devel` and `libtiff-devel`. Apart from a C++ compiler, and the `make` build system, you'll also need to have CMake (can be installed using the `cmake` package).

1. Download the Simpact Cyan source code and extract it somewhere, or clone the GitHub repository somewhere. Below, I'll assume that this has created a path called `/home/me/simpactcyan`.

2. We need to create a directory where the build files will be stored, let's create a directory `/home/me/simpactcyan/build` for this. In

that directory, we'll call the `cmake` command:

```
cmake /home/me/simpactcyan
```

3. When the CMake step is finished, you can now just run

   ```
   make
   ```

   to start building the Simpact Cyan executables, or use something like

   ```
   make -j 4
   ```

   to start four build jobs in parallel, which can speed up the process if your computer or laptop has several processor cores.

4. When the build is complete, running e.g. `./simpact-cyan-release` will will show a usage message.

Currently NOT WORKING(tested on Ubuntu 22.04):

```
 5. To make sure that this version is used from within an R session,
execute
    the following commands **before** loading the `RSimpactCyan`
package:


Sys.setenv(PATH=paste("/home/me/simpactcyan/build",Sys.getenv("PATH"),se
p=":"))
        Sys.setenv(PYTHONPATH="/home/me/simpactcyan/python")
        Sys.setenv(SIMPACT_DATA_DIR="/home/me/simpactcyan/data/")

    The first line makes sure that the `RSimpactCyan` will be able to
locate the
    compiled executables. The second line makes sure that the Python
module
    that's stored in that directory will be found. The last line tells
the
    library where the data files can be found. Note that the data
directory
    needs to end with a single `/`.

    Only after these lines have been set can you load the library:

        library("RSimpactCyan")
```

PROF

```
    If you then use e.g. the `simpact.run` command, the newly compiled
versions
    are used.
```

6. The previous step allows you to run the new executables from within R, but to debug the code this is not the most convenient way to run the program. Instead, first create a configuration file using e.g. the `dryrun` option of `simpact.run`. You can then just run the executable from the command line, as described in the manual, for example:

   ```
   ./simpact-cyan-debug /path/to/config.txt 0 opt
   ```

Compiling Simpact Cyan on MS-Windows

Make sure you use Visual Studio 2015. In the following I'm assuming a 32-bit build (as this also works on 64-bit systems), but a 64-bit build should work as well, with some obvious changes to the commands.

1. The source code needs GSL and TIFF libraries to build; to speed things up pre-compiled versions can be downloaded from: simpact_vs2015_deps.rar.
   Extract these somewhere, below I'll assume that after extraction you have a directory called `c:\simpact_vs2015_deps\32bit`. If you've extracted the RAR archive somewhere else you'll need to change some paths in the instructions below.

2. To be sure that the DLL files for the GSL and TIFF libraries can be found when executing the generated Simpact Cyan executable later on, add the `c:\simpact_vs2015_deps\32bit\bin` directory to the PATH environment variable (see e.g. this link if you don't know how to do this).

3. Download the Simpact Cyan source code and extract it somewhere, or clone the GitHub repository somewhere. Below, I'll assume that this has created a path called `c:\projects\simpactcyan`.

4. In that path, there's a file called `CMakeLists.txt`. Using a tool called CMake, this file will be analyzed and a Visual Studio project will be created. Make sure CMake is installed and start the GUI.

5. In the top part of the window, you need to fill in where the source code is, and where to build the binaries. In the source code field, fill in `c:\projects\simpactcyan`. The build directory is arbitrary, that's just

PROF

where the compiled files and executables will be stored, I typically use something like `c:\projects\simpactcyan\build`.

6. We want to make sure that CMake can find the GSL and TIFF libraries, which is why we'll do the following: click the `Add Entry` button, enter `CMAKE_INSTALL_PREFIX` where it says `Name`, specity `PATH` as the `Type`, set the `Value` field to `c:\simpact_vs2015_deps\32bit` and click `Ok`. In the central part of the window, you'll see this `CMAKE_INSTALL_PREFIX` appear.

7. In the main CMake GUI, press the button `Configure` and you'll get a window that asks to specify the generator for the project. There, you should be able to select `Visual Studio 14 2015` (the version number of Visual Studio 2015 is 14).

8. If everything works correctly, you'll see a bunch of red lines appear in the same part of the window where the `CMAKE_INSTALL_PREFIX` was shown. If you click `Configure` again, they should become white, and the `Generate` button should be possible to click. If there are still some red lines, it's possible that you have to press `Configure` once more. After clicking `Generate`, a file called `c:\projects\simpact\build\simpact-cyan.sln` should be created.

9. Open this `.sln` file (the 'solution' file, which is the name for a project in Visual Studio) with Visual Studio 2015. When this has been started, we'll be able to actually compile the source code and build the executables.

10. By default, you'll see that `Debug` option is selected in the bar below the menu bar. This means that the debug version of the code will be compiled. You can also change this to `Release` (and even some other things), which will compile the optimized version of the code. For now, we'll just leave this at `Debug`.

11. In the solution explorer, probably at the right of the Visual Studio window, you'll see the `ALL_BUILD` subproject. If you right-click this, you'll get a pop-up menu where the top item is `Build`. Click this to start the compilation process.

12. If all went well, the folder `c:\projects\simpactcyan\build\Debug` should now contain the executable called `simpact-cyan-debug.exe`. Since this is a command line program, we'll start it from the command line: open the 'Command Prompt' and go to the specified directory. If you then type `simpact-cyan-debug`, the program should output some usage information.

13. To compile the release version, simply change the the selection from step 10, right click `ALL_BUILD` again and select `Build`.

14. To run the version you're working on from within R, you need to run the following commands **before** loading `RSimpactCyan`:

Sys.setenv(PATH=paste("c:\projects\simpactcyan\build\Debug;c:\projects\simpactcyan\build\Release",Sys.getenv("PATH"),sep=";"))
Sys.setenv(PYTHONPATH="c:\projects\simpactcyan\python")
Sys.setenv(SIMPACT_DATA_DIR="c:\projects\simpactcyan\data\")

```
The first line makes sure that the `RSimpactCyan` will be able to locate
the
compiled executables. Note that you need to have both the debug and
release
executables. The second line makes sure that the Python module that's
stored
in that directory will be found. The last line tells the library where
the
data files can be found. Note that double backslashes need to be used
(or
a single normal slash), and that the data directory needs to end with
`\\`
(or with a single `/`).

Only after these lines have been set can you load the library:

    library("RSimpactCyan")

If you then use e.g. the `simpact.run` command, the newly compiled
versions
are used.
```

15. The previous step allows you to run the new executables from within R, but to debug the code this is not the most convenient way to run the program. Instead, first create a configuration file using e.g. the `dryrun` option of `simpact.run`.

16. In Visual Studio, right click the `simpact-cyan` subproject and select `Set at StartUp Project`. Right click the `simpact-cyan` subproject again and select `Properties` at the bottom of the menu. In this properties dialog, make sure that the drop-down list next to `Configuration:` (at the top-left of the dialog) is showing `Debug`. In the tree view on the left, you should see a `Debugging` option in the `Configuration Properties`, which you need to select.

17. In the option on the right that appear then, you'll see an empty input field named `Command Arguments`. There, you'll need to fill in the arguments to the executable, as described in the manual. So this would be something like

    c:\path\to\generated\configfile.txt 0 opt

PROF

```
After pressing `Ok` to store the settings and close the dialog, make
sure
you set the active version to `Debug` (and not `Release`) in the bar
below
the menu bar. Pressing F5 then not only starts the program with the
specified
configuration, but also starts the Visual Studio debugger, allowing you
to
set breakpoints at various locations.
```