****
Instalation instructions for STIC
****

STiC is a non-LTE inversion code created by
J. de la Cruz Rodriguez, J. Leenaarts, S. Danilovic and H. Uitenbroek.

STiC support the inversion of PRD lines and Zeeman polarization signals.
The code allows applying regularization functions to the model parameters

DISCLAIMER:
==========

The code is still under development and it provided "as it is".
Due to limited man power, we will provide support within scientific
collaborations.

Limited support or none might be provided to non-collaborators.

CITING STIC:
============

Please, when using STiC add the following description (or a very similar one,
including all references)
to acknowledge all the work that has been done by different scientists over the past
years:

STiC (de la Cruz Rodriguez et al. 2018; de la Cruz et al. 2016) is a MPI-parallel
non-LTE inversion code
that utilises a modified version of RH (Uitenbroek 2001) to solve the atom population
densities assuming statistical equililibrium and plane-parallel geometry and it
allows including partial
redistribution effects of scattered photons (Leenaarts et al. 2012). The radiative
transport equation is
solved using cubic Bezier solvers (de la Cruz Rodriguez et al. 2013).

The inversion engine of STiC includes an equation of state extracted from the SME
code (Valenti & Piskunov 2016).

CODE DEPENDENCIES:
==================

Dependencies: netcdf-cxx4, Eigen-3, FFTW-3, MPI-2.
Tested compilers: gnu compilers (gcc/g++/gfortran >= 4.8), Intel compilers.
clang/clang++ can be used, but since we need a fortran compiler, we need to
have gcc/gfortran anyway...

The C++ compiler must support C++-11.

OSX installation instructions:
==============================

The simplest and most straight way of installing all dependencies to compile STIC
is to use MacPorts or homebrew. These instructions are based on the former.
https://www.macports.org

OSX includes and old version of clang/clang++, but no fortran compiler. We need
to install the GCC compilers in any case.

1) Install gcc (in this example I am installing gcc-8) and make it the default
gcc compiler in the system:

```
sudo port install gcc8
sudo port select gcc mp-gcc8
hash -r

1.b) gcc in macports used to not include AVX instruction optimization.
To get better performance, we must install the assembler compiler from clang.

sudo port install clang-7.0
sudo port select clang mp-clang-7.0
hash -r

2) Install openmpi and make it the default mpi installation in the system:

sudo port install openmpi +gcc8
sudo port select mpi openmpi-mp-fortran

3) Install FFTW-3 and Eigen-3:

sudo port install fftw-3 +gcc8
sudo port install eigen3 +gcc8


4) Install netcdf-gxx4. This part is a bit trickier because netcdf depends on
the HDF5 package. So we first install HDF5 with the right flags. Do not
install the MPI-parallel version because it is not compatible with the C++
bindings of netcdf:

sudo port install hdf5 +gcc8 +hl
sudo port install netcdf +gcc8 +netcdf4
sudo port install netcdf-cxx4 +gcc8
hash -r

Now all dependencies have been installed. You can pull the latest version
of STiC from the repository:

git clone https://github.com/jaimedelacruz/stic.git

We have prepared different makefiles for different platforms and operating systems.
If you are using a Mac, set the following environment variables (you can do this
in your $HOME/.bashrc file and start a new terminal to make the changes effective):

export OS=Darwin
export CPU=i386
export OMPI_CC = gcc
export OMPI_CXX = g++
export OMPI_FC = gfortran

source ~/.bashrc


These variables are just labels for the makefile, the code will be compiled in 64 bit
mode anyway.

STiC is based on a modified version the excellent RH code (Uitenbroek 2001).
We have encapsulated RH in a module that needs to be compiled first:

cd stic/src/rh
make clean
make

cd rh_1d/
make clean
make

cd ../../
make clean
```

```
make
```

If everything went fine you will find the binary of STiC in the main src folder.
You can try to execute it and see if it starts. You should get something like this:

```
   SSSSSSSSSSSSSS TTTTTTTTTTTTTTTTTTTTTT  iiii         CCCCCCCCCCCCC
 SS:::::::::::::ST:::::::::::::::::::::T i::::i     CCC::::::::::::C
S::::::SSSSSS::::::ST:::::::::::::::::::::T  iiii    CC:::::::::::::::C
S:::::S     SSSSSSST:::::TT:::::::TT:::::T         C:::::CCCCCCCC::::C
S:::::S            TTTTTT  T:::::T  TTTTTTiiiiiii C:::::C       CCCCCC
S:::::S                    T:::::T        i:::::iC:::::C
 S::::SSSS                 T:::::T         i::::iC:::::C
  SS::::::SSSSS            T:::::T         i::::iC:::::C
    SSS::::::::SS          T:::::T         i::::iC:::::C
       SSSSSS::::S         T:::::T         i::::iC:::::C
            S:::::S        T:::::T         i::::iC:::::C
            S:::::S        T:::::T         i::::i C:::::C       CCCCCC
SSSSSSS     S:::::S      TT:::::::TT      i::::::i C:::::CCCCCCCC::::C
S::::::SSSSSS:::::S      T:::::::::T      i::::::i  CC:::::::::::::::C
S:::::::::::::::SS       T:::::::::T      i::::::i    CCC::::::::::::C
 SSSSSSSSSSSSSS          TTTTTTTTTTT      iiiiiiii       CCCCCCCCCCCCC

STIC: Initialized with 1 process(es)
file_check: ERROR, file  does not exist!
```

Linux installation instructions
==========

It is hard to provide instructions that work in all Linux distributions.
The important part is to identify the correct package names in your distribution.
All dependencies are standard open-source packages and they are included in all
Linux distributions that I have encountered. In this example I will use debian
Stretch.
Ubuntu should have very similar (if not identical) package names since it is based on
Debian:

1) Install gcc/g++/gfortran:

```
sudo apt install build-essential
```

2) Install openmpi and the header files:

```
sudo apt install libopenmpi2 libopenmpi-dev
```

3) Install Eigen-3 and fftw-3:

```
sudo apt install libfftw3-dev libfftw3-bin
sudo apt install libeigen3-dev
```

4) Install netcdf/netcdf-cxx4:

```
sudo apt install libnetcdf-c++4-dev libnetcdf-c++4 libnetcdf-dev
```

Now all dependencies have been installed. You can pull the latest version
of STiC from the repository:

```
git clone https://github.com/jaimedelacruz/stic.git
```

We have prepared different makefiles for different platforms and operating systems.
If you are using a Mac, set the following environment variables (you can do this
in your $HOME/.bashrc file and start a new terminal to make the changes effective):

```
export OS=Linux
export CPU=x86_64
```

STiC is based on a modified version the excellent RH code (Uitenbroek 2001).
We have encapsulated RH in a module that needs to be compiled first:

```
cd stic/src/rh
make clean
make

cd rh_1d/
make clean
make

cd ../../
make clean
make
```

If everything went fine you will find the binary of STiC in the main src folder.
You can try to execute it and see if it starts. You should get something like this:

```
   SSSSSSSSSSSSSS TTTTTTTTTTTTTTTTTTTTT  iiii         CCCCCCCCCCCCC
 SS:::::::::::::ST:::::::::::::::::::::T i::::i     CCC::::::::::::C
S::::::SSSSSS::::::ST:::::::::::::::::::::T  iiii    CC:::::::::::::::C
S::::::S     SSSSSSST:::::TT:::::::TT:::::T         C:::::CCCCCCCC::::C
S::::::S            TTTTTT  T:::::T  TTTTTTiiiiiii C:::::C       CCCCCC
S::::::S                    T:::::T        i:::::iC:::::C
 S::::SSSS                  T:::::T         i::::iC:::::C
  SS::::::SSSSS             T:::::T         i::::iC:::::C
    SSS::::::::SS           T:::::T         i::::iC:::::C
       SSSSSS::::S          T:::::T         i::::iC:::::C
            S:::::S         T:::::T         i::::iC:::::C
            S:::::S         T:::::T         i::::i C:::::C       CCCCCC
SSSSSSS     S:::::S       TT:::::::TT      i::::::i C:::::CCCCCCCC::::C
S::::::SSSSSS:::::S       T:::::::::T      i::::::i  CC:::::::::::::::C
S:::::::::::::::SS        T:::::::::T      i::::::i    CCC::::::::::::C
 SSSSSSSSSSSSSS           TTTTTTTTTTT      iiiiiiii       CCCCCCCCCCCCC

STIC: Initialized with 1 process(es)
file_check: ERROR, file  does not exist!
```

Windows installation instructions

You must be kidding ... right? :-)
Jokes appart, Windows is not supported.

Python tools for reading the input/output files

You will need a python distribution with numpy, netCDF4 and scipy.
The easiest way to get install anaconda python, which is freely available.

In stic/pythontools there is a set of tools that should allow you to calibrate
Your data and write the input files. The main package is called "sparsetools".
These tools were originally written for another code (hence the name).\

There are 2 basic classes in sparsetools: profile and model.

The model class

If you want to create an empty model so you can fill in the variables you can do something like this:

```
import sparsetools as sp
m = sp.model(nx = 10, ny=10, ndep = 42, nt = 1)
```

Now you will see that m contains all thermodynamical variables needed for STiC:
temp, vlos, vturb, Bln, Bho, azi, pgas, nne, rho, ltau, z, cmass
these variables have dimension [nt, ny, nx, ndep], where ndep is the fast axis.

Not all of them need to be filled. You will need at least one vertical depth scale (z or ltau or cmass), temp and at least one pressure/density scale (pgas, rho, nne).

Once you have populated the variables, you can write the model to disk:

```
m.write('name.nc')
```

If you have the result from an inversion an you want to read the model, simply use:

```
m = sp.model('result.nc')
```

and the package will populate all the variables with the content of the file.

The profile class

Similarly we have a profile class with the following variables:
wav[nwav], dat[nt, ny, nx, nwav, nstokes], weights[nwav, nstokes].

To initialize an empty object:

```
p = sp.profile(nx=10, ny=10, nwav=200, ns=4, nt=1)
```

Or to read one from HD:

```
p = sp.profile('profiles.nc')
```