CFBC Package - User Guide v1.0

Peter H. Heins

Email: `p.heins@sheffield.ac.uk`

February 18, 2016

Contents

1 Introduction

The CFBC package allows a user to implement inhomogeneous wall boundary conditions into the open-source spectral CFD code *Channelflow* written by J. F. Gibson [1]. The package also includes a C++ *controller class* to aid in the simulation of linear time-invariant (LTI) output-feedback controllers in feedback with turbulent channel flow. All programs included in the package have been validated and used by the author for published research [2].

2 Installation

The CFBC package only includes files which have been created or modified by the author. Therefore, if you do not already have a version of *Channelflow*, you are required to download one from **http://channelflow.org**. The CFBC package was designed to be installed with version: channelflow-1.4.2. Ensure you have all the prerequisites listed on the website installed on your system. Additional to those listed on the website, CLAPACK and CBLAS are also required if installing on a Linux machine. If installing on Mac OSX, the Accelerate framework already includes these libraries. In the following, step by step instructions will be given for installation of *Channelflow* with the CFBC package on Linux and Mac OSX.

2.1 Linux

1. Download `channeflow-1.4.2.tar.gz` from http://channelflow.org.
2. Unzip tarball: `tar xvfpz channeflow-1.4.2.tar.gz`
3. `cd channelflow-1.4.2`
4. `./configure --prefix=/home/channelflow-1.4.2`
5. `cd channelflow`
6. `rm dns.* tausolver.*`
7. Now move `dns.cpp`, `dns.h`, `tausolver.cpp`, `tausolver.h`, `controller.cpp` and `controller.h` from `CFBC/Linux` into directory:
   `/home/channelflow-1.4.2/channelflow`
8. In the directory `/home/channel-1.4.2/channelflow`, open the file `Makefile` in a text editor.

   - For variable "`am_libchflow_la_OBJECTS=`" add `controller.lo` to the list.

   - Change variable "`LIBS`" from:
   "`LIBS = -lhdf5_cpp -lhdf5_hl -lhdf5 -lz -lfftw3`"
   to
   "`LIBS = -lhdf5_cpp -lhdf5_hl -lhdf5 -lz -lfftw3 -L/usr/lib64/atlas -lclapack -lcblas`".

- For variable "`libchflow_la_SOURCES=`" add `controller.cpp` to the list.
- For variable "`pkginclude_HEADERS=`" add `controller.h` to the list.

9. In the directory `/home/channel-1.4.2/tests`, open the file `Makefile` in a text editor.

   - Change variable "`LIBS`" from:
   "`LIBS = -lhdf5_cpp -lhdf5_hl -lhdf5 -lz -lfftw3`"
   to
   "`LIBS = -lhdf5_cpp -lhdf5_hl -lhdf5 -lz -lfftw3 -L/usr/lib64/atlas -lclapack -lcblas`".

10. Move to directory `/home/channel-1.4.2`

11. `make`

12. `make install`

13. `make test` (optional)

14. Move `CFBC_Examples` into `/home/channel-1.4.2`

15. Copy `/home/channel-1.4.2/examples/Makefile` to `/home/channel-1.4.2/CFBC_Examples`

16. Open `Makefile` in a text editor. Edit `CHANNELDIR` to `CHANNELDIR = $(HOME)/channelflow-1.4.2`

17. To compile and run the included example programs:

    ```
    make example.x
    ./example.x
    ```

2.2 Mac OSX

1. Download `channeflow-1.4.2.tar.gz` from http://channelflow.org.

2. Unzip tarball: `tar xvfpz channeflow-1.4.2.tar.gz`

3. `cd channelflow-1.4.2`

4. `./configure --prefix=/home/channelflow-1.4.2`

5. `cd channelflow`

6. `rm dns.* tausolver.*`

7. Now move `dns.cpp`, `dns.h`, `tausolver.cpp`, `tausolver.h`, `controller.cpp` and `controller.h` from `CFBC/Mac` into directory: `/home/channelflow-1.4.2/channelflow`

8. Using a text editor, change `<fftw3.h>` to `"/sw/include/fftw3.h"` in `chebyshev.h, dns.h, flowfield.h, periodicfunc.h, symmetry.h`

9. Uncomment "`#include <Accelerate/Accelerate.h>`" in `dns.h`
10. Uncomment "`typedef unsigned int uint`" in `mathdefs.h`
11. In the directory `/home/channel-1.4.2/channelflow`, open the file `Makefile` in a text editor.

    - For variable "`am_libchflow_la_OBJECTS=`" add `controller.lo` to the list.

    - Change variable "`LIBS`" from:
    "`LIBS = -lhdf5_cpp -lhdf5_hl -lhdf5 -lz -lfftw3`"
    to
    "`LIBS = -lhdf5_cpp -lhdf5_hl -lhdf5 -lz -lfftw3 -framework Accelerate`".

    - For variable "`libchflow_la_SOURCES=`" add `controller.cpp` to the list.

    - For variable "`pkginclude_HEADERS=`" add `controller.h` to the list.
12. In the directory `/home/channel-1.4.2/tests`, open the file `Makefile` in a text editor.

    - Change variable "`LIBS`" from:
    "`LIBS = -lhdf5_cpp -lhdf5_hl -lhdf5 -lz -lfftw3`"
    to
    "`LIBS = -lhdf5_cpp -lhdf5_hl -lhdf5 -lz -lfftw3 -framework Accelerate`".
13. Move to directory `/home/channel-1.4.2/programs`
14. In `findsymmetries.cpp` change `uint` to `int`
15. Move to directory `/home/channel-1.4.2`
16. `make`
17. `make install`
18. `make test` (optional)
19. Move to directory `/home/channel-1.4.2/lib`
20. Copy the FFTW3 libraries to this directory via: `cp /sw/lib/libfftw3* .`
21. Move `CFBC_Examples` into `/home/channel-1.4.2`
22. Copy `/home/channel-1.4.2/examples/Makefile` to `/home/channel-1.4.2/CFBC_Examples`
23. Open `Makefile` in a text editor. Edit `CHANNELDIR` to `CHANNELDIR = $(HOME)/channelflow-1.4.2`
24. To compile and run the included example programs:

    ```
    make example.x
    ./example.x
    ```

3 Implementing Boundary Conditions

This section describes how to implement simply-varying or static inhomogeneous boundary conditions in *Channelflow*. We shall use the script `BCs_Run.cpp` in the `CFBC_Examples` directory for this brief tutorial. The vast majority of this file consists of standard *Channelflow* code and it assumed that you are already familiar with this. The script sets up a low Reynolds number plane channel flow, initialised from a random initial condition. It then simulates this flow with sinusoidal wall-normal velocity boundary conditions for 10 time units.

On line 125 of the script, a boundary condition array is generated using the syntax:

```
FlowField BC(Nx,2,Nz,3,Lx,Lz,a,b,Physical,Physical);
```

This uses the in-built `FlowField` class to generate a `Nx` × `2` × `Nz` × `3` array. The dimensions of this array correspond to the three components of velocity at the upper and lower walls. This array is subsequently set to zero.

On line 174 of the script, we have the syntax:

```
if (Controlled){
dns.reset_dtIH(dt,BC);
}
```

This new time-step reset function needs to be called if starting a simulation with non-zero boundary conditions, i.e. if loading boundary conditions from a file as on line 142.

On line 190 of the script, within the time-stepping for-loop, we change the boundary conditions in the `BC` array. Here, we have opted to only change boundary conditions for wavenumber pair ($k_x = 0, k_z = 1$), at both walls. The syntax is:

```
BC.cmplx(0,0,1,1) = 0.01*sin(t);
BC.cmplx(0,1,1,1) = 0.01*sin(t);
```

The boundary condition array has already been transformed into state (x-z Fourier, y Physical) on line 138. Therefore, the first and third numbers of `BC.cmplx` correspond to the wavenumber pair. The second number selects which wall you want to assign the boundary condition to:

$0 \rightarrow$ lower wall
$1 \rightarrow$ upper wall

The fourth number corresponds to the component of velocity you wish to assign the boundary condition to:

$0 \to u$
$1 \to v$
$2 \to w$

However, you are free to assign boundary conditions in spectral or physical state, and to as many wavenumber pairs or physical locations as required. You are also free to assign boundary conditions to multiple components of velocity, although this has not been validated.

On line 309 of the script, we advance the simulation forward in time with the boundary conditions implemented. The syntax is:

```
dns.advance_inhom(u,q,BC,dt.n());
```

This is a new advance function in the `dns.cpp` class. It will incorporate the boundary conditions set in the `BC` array between `t` and `t+dT`.

You can see how the boundary conditions affect the simulation by varying the magnitude of the sinusoidal signal, currently set at `0.01`. Making it too large will cause the CFL condition of the simulation to get very large which is to be expected.

4 Using the Controller Class

In this section, the `controller.cpp` class included in the CFBC package will be outlined. We shall use the script `Controller_Run.cpp` in the `CFBC_Examples` directory in this tutorial. The script runs a simulation of low Reynolds number plane channel flow in feedback with an array of LTI output-feedback controllers. Each controller acts on one wavenumber pair. In this example, ten wavenumber pairs will be controlled simultaneously.

The controller class in the CFBC package accommodates **continuous-time** controllers of the form:

$$\dot{x}(t) = Ax(t) + By(t), \tag{1a}$$

$$u(t) = Cx(t) + Du(t), \tag{1b}$$

where x is the controller state vector, u is the control signal vector, and y is the flow measurement vector calculated from $y = C_s X_{CF}$, where X_{CF} are the primitive variables in the *Channelflow* simulation, and C_s is a "sensor" matrix. You are required to generate files for matrices A, B, C, D, C_s for each controller in the correct format. To help with this, the Matlab script `K_File_Gen.m` is included in the `CFBC_Examples` directory. The script should not need much explaining. All you are required to do is form each controller using Matlab's `ss(A,B,C,D)` function using the naming style defined at the top of the script. The sensor matrix C_s is generated using a function in the script, you would have to write your own. **Note: controllers must be real-valued.** Once run, a directory named `Mult_Control_Mat` should be created, containing binary files:

`Ka_mat_kx(n)kz(n).bin`	A controller matrix
`Kb_mat_kx(n)kz(n).bin`	B controller matrix
`Kc_mat_kx(n)kz(n).bin`	C controller matrix
`Kd_mat_kx(n)kz(n).bin`	D controller matrix
`Cs_mat_kx(n)kz(n).bin`	sensor matrix

for each controlled wavenumber pair. In order to run `Controller_Run.cpp`, a `Mult_Control_Mat` directory containing the controller matrices for ten wavenumber pairs has been included.

Turning our attention to `Controller_Run.cpp`, we will now see how the controllers are implemented. Lines 148-151 have the syntax:

```
int minKx = 0;
int maxKx = 0;
int minKz = 0;
int maxKz = 10;
```

This allows you to select which wavenumber pairs you wish to control. In this case, $(k_x = 0, k_z \leq 10)$. Currently, you are restricted to controlling adjacent wavenumber pairs.

On line 156 of the script, we have the syntax:

```
int uvw = 1;
```

This variable allows you to select which component of velocity at the walls you wish to use as actuation - $0 = u$, $1 = v$, $2 = w$.

On line 157 of the script, we have the syntax:

```
int NumInputs = 8;
```

This variable selects the number of real-valued inputs to the controllers, i.e. measurements of the flow. In this example, we measure streamwise and spanwise wall-shear stress at both walls. However, as these measurements are complex-valued, the real and imaginary parts are treated separately. Therefore, the number of inputs to the controllers are: 2(upper/lower wall)$\times$2(streamwise/spanwise shear-stress)$\times$2(real/imaginary part)$= 8$.

On line 160 of the script, we have the syntax:

```
double tau = 0.010;
```

The variable `tau` is the actuator time-constant. The controller class models actuator dynamics using a first-order low-pass filter of the form:

$$\dot{q}(t) = -\frac{1}{\tau}q(t) + \frac{1}{\tau}u(t), \tag{2}$$

where q are the actual boundary conditions set in the simulation. If low-pass filtering is not required, just set $\tau \ll 1$.

On line 162 of the script, we have the syntax:

```
bool Spectral_states = false;
```

This indicates whether the controller states are physical (`false`) or Chebyshev spectral coefficients (`true`).

On line 164 of the script, an object called `Hinf` is created using the controller class.

On line 455, we have the syntax:

```
dns.advance_inhom_CON(Hinf,u,q,BC,F,dt.n(),IO,CStateMat);
```

This newly written advance function, advances the simulation forward in time in feedback with the continuous-time controllers. Because the controllers are continuous-time, the time-step `dt` can be varied. The controllers are integrated forward in time at each time sub-step using an implicit Crank-Nicolson scheme, ensuring accuracy and stability. `F` is a variable of class `FlowField` which outputs the forcing from the nonlinearity. `IO` is an array created on lines 171-187, and stores inputs and outputs from the controller at each time-step for printing to files. This array has been set up for the current number of inputs and outputs and may need to be changed for your set up (both in the run file, and `controller.cpp`). `CStateMat` is a 3D array created on line 166, and stores the states (x) of each controller for the current time-step.

Try running simulations controlled and uncontrolled by setting `bool Controlled = true/false` on line 29. Use the same initial flow field for both and see how the energy and drag data outputted from the simulations differ.

5 Citation

If using the CFBC package for published research, please use the following citation:

P. Heins. *Modelling, Simulation and Control of Turbulent Flows.* PhD thesis, University of Sheffield, 2015.

References

[1] J. F. Gibson. Channelflow: A spectral Navier-Stokes simulator in C++. Technical report, U. New Hampshire, 2012.

[2] B. Ll. Jones, P. H. Heins, E. C. Kerrigan, J. F. Morrison, and A. S. Sharma. Modelling for robust feedback control of fluid flows. *Journal of Fluid Mechanics*, 769:687–722, 2015.