Space Chrage Tracking Package

Hung-Chun Chao

8/7/2015

This is a package based on the SimTrack's framework, which facilitates the lattice definition, multiple particle tracking, and twiss optics calculation.

I. Packages:

1. SimTrack
   - ➓ It is a C++ library for simple tracking code. It comes with a form of header file simtrack.h, which is included in the tar file. See the manual for instructions (BNL-91006-2010-IR).
   - ➓ The compiler g++ version 4.4 is needed.
   - ➓ I have made some modifications. See the README file for details.
   - ➓ You have to make sure GSL library is installed in your system and modify the path in the second line in the Makefile accordingly.
   - ➓ To build it, switch to the directory and type the commands:
     make clean
     make
     Six .o object files and the archive file will be generated. The whole simtrack directory should be placed in the source file folder.
1. Faddeeva
   - ➓ A C++ library to calculate the complementary error functions, which is necessary for space charge calculation (myroutine1.cpp).
   - ➓ These two files Faddeeva.cc and Faddeeva.hh, written by Steven G. Johnson in MIT, can be downloaded via http://ab-initio.mit.edu/wiki/index.php/Faddeeva_Package.
1. Myroutine files
   - ➓ myroutine.h, myroutine.cpp:
     Dealing with basic math, random number and the distribution generation.
   - ➓ myroutine1.h, myroutine1.cpp:
     There are two functions calculate the space charge kicks. Fsc use Faddeeva::erfc and Fsc2 uses Faddeeva::w. The second one is scaled so it is recommended.
   - ➓ myroutine2.h, myroutine2.cpp:
     A collection of functions dealing with the calculation of sextupole harmonics, response matrix, save/load quadrupole strengths, etc. It is custom-made for the specific lattice.
1. Armadillo

➓ A highly developed C++ linear algebra library, which facilitates the calculation of statistics and matrix input/output.

➓ It can be downloaded via http://arma.sourceforge.net/. To build it, cmake is required in the Linux system. See README.txt for the installation guide upon unzipping the tar file.

➓ The current version (2015-06-24) is 5.200.2, which can make use GPUs if it is properly installed.

1. NLOpt (optional)
   - ➓ A nonlinear optimization package. The current version is 2.4.2. It can be downloaded via

http://ab-initio.mit.edu/wiki/index.php/NLopt. The installation guide can be found inside.

1. BOOST lib (optional)
   - Check http://www.boost.org/ for download and installation.
1. GNU Parallel
- version >= 3 is recommended.

We have to make sure the library paths is in the environment variable LD_LIBRARY_PATH.

The files are organized as follows. The source files are in the source directory. Within it there is simtrack dir, where the library are needed to be built first. The dependency of files are written in the Makefile in source dir. Once the program is compiled, it is copied to the individual project folder. The compiled program, output data, graphics, along with data analysis scripts are in the individual folder.

II. Example programs

This is a project doing stopband harmonics corrections by nlopt (nlopt2/) and the tracking.
Files:

- source files: source/nlopt2.cpp, nlopt_obj1.h, nlopt_obj2.cpp, loadQ_track.cpp, FODO4.h, FODO4.cpp
- loadQ_track: This program is produced from the code loadQ_track.cpp, which loads the quadrupole setting files into FODO4 lattice and perform multiple tracking.
- nlopt2: This program is produced from nlopt2.cpp and performs the nonlinear optimization. The objective is defined in nlopt_obj1.h and nlopt_obj1.cpp. Read the first few lines in the source code to see the usage input/output.
- viewer: A program to read and display the calculated results graphically and the setting files according to the labels. One has to edit run.sh to precede # to the case to be plotted. The data has to be pre-made and put into the directory data/. It is written by wxWidget 3.0 library.
- runQ.sh produces the quadrupole setting files by the compiled program Qerrgen. The input data is included in the end of the script. It produces Q_0.dat, Q_1.dat, Q_5.dat in this case.
- parallel_tracking.sh is the script to do parallel tracking in multi-core CPUs workstation. It arranges the arguments to be fed into the program loadQ_track and use command line GNU parallel version 2 to distribute the jobs into the CPUs. The file table.in is fed into this script. The lines preceded by # are the cases to be ignored and will not be executed.
- The newer version of GNU parallel has the ability to distribute jobs in a cluster by assigning the hosts and the number of nodes. Somehow the grammar for the command line has to be changed.
- run.sh is a script controlling all the case study of harmonic correction. The lines are the input arguments to be fed into the program nlopt2 and the script plot.sh. The line preceded by # or % are ignored and will not be executed.

- get_table_out.sh is a script to arrange to plot.
- plot.sh and plot1.sh are the scripts to arrange the data and generate the postscript plots. Gnuplot is needed to run this script. They are readable and details are needed be modified accordingly in order to make good plots.
- The dependency of the file is listed in this Makefile. One can change the paths accordingly. BOOST library is required for the nlopt_obj2.o

Procedures:
0. prepare:
1. Install the libraries.
2. Go to source, edit Makefile, build the three executables Qerrgen, nlopt2, loadQ_track and copy it to the directory nlopt2.

I. Run the corrections and generate quadrupole error files and plots before and after correction.
3. In nlopt2, execute the script runQ.sh to generate Q_0.dat, Q_1.dat, Q_1.dat, rename them to Q0.in, Q1.in, and Q5.in to feed to nlopt2. Q0.in is no-error, Q5.in is the 5 times scaled quadrupole error.
4. Modify Trm_xx.in which assigns the trim quadrupoles to be used. The first column is the index, second column is min boundary, and the third column is max boundary.
5. Modify Obj_x.in which assigns the weights for the object. This file is self-explained and easy to understand.
6. Modify and execute run.sh. The line preceded by # or % are ignored and will not be executed. For each case, nlopt2 output 7 files. These files will be moved to data/. The resulting quadrupole setting file is Q2_(label).dat. plot.sh produces 3 files in eps/.

II. Do the tracking to check the emittance growth
7. Write a table like table.in to be fed into the script parallel_tracking1.sh. Configurate parallel_tracking1.sh and run it. Now wait. The results will be tbt_x.dat, dnu_x.dat, and env_x.dat and will be moved to data_track/.
8. Plot tbt_x.dat to check the emittance evolution.

```
#Makefile for nlopt2 project
CC = g++
ARCH = $(shell getconf LONG_BIT)
OPTS = -O3

ARMA_INCLUDE=-I$(HOME)/usr/include
ARMA_LIBRARY=-L$(HOME)/usr/lib64 -larmadillo
SIMTRACK_INC=-I./simtrack
SIMTRACK_LIB=-lm -lgsl -lgslcblas -L./simtrack -lsimtrack$(ARCH)
GA_INCLUDE=-I$(HOME)/usr/local/include
GA_LIBRARY=-L$(HOME)/usr/local/lib -lm -lga
NLOPT_INCLUDE=-I$(HOME)/usr/local/include
NLOPT_LIBRARY=-L$(HOME)/usr/local/lib -lm -lnlopt
BOOST_INCLUDE=-I$(HOME)/usr/local/include
BOOST_LIBRARY=-L$(HOME)/usr/local/lib
```

```
myroutine.o: myroutine.cpp myroutine.h
   $(CC) $(OPTS) -c $<
myroutine1.o: myroutine1.cpp myroutine1.h myroutine.cpp myroutine.h Faddeeva.cc Faddeeva.hh
   $(CC) $(OPTS) -c $<
myroutine2.o: myroutine2.cpp myroutine2.h myroutine.cpp myroutine.h
   $(CC) $(OPTS) -c $< $(ARMA_INCLUDE)
Faddeeva.o: Faddeeva.cc Faddeeva.hh
   $(CC) $(OPTS) -c $<

FODO4.o: FODO4.cpp FODO4.h
   $(CC) $(OPTS) -c $< $(SIMTRACK_INC)

loadQ_track.o: loadQ_track.cpp
   $(CC) $(OPTS) -c $< $(SIMTRACK_INC)
loadQ_track: loadQ_track.o FODO4.o myroutine.o myroutine1.o myroutine2.o Faddeeva.o
simtrack/libsimtrack$(ARCH).a
   $(CC) $^ $(SIMTRACK_INC) $(SIMTRACK_LIB) $(ARMA_INCLUDE) $(ARMA_LIBRARY) -
o $@

nlopt_obj1.o: nlopt_obj1.cpp nlopt_obj1.h
  $(CC) $(OPTS) -c $< $(ARMA_INCLUDE) $(SIMTRACK_INC) $(BOOST_INCLUDE)
        nlopt2.cpp
nlopt2.o: nlopt2.cpp
   $(CC) $(OPTS) -c $< $(SIMTRACK_INC) $(ARMA_INCLUDE) $(NLOPT_INCLUDE)
nlopt2: nlopt2.o nlopt_obj1.o FODO4.o myroutine.o myroutine2.o simtrack/libsimtrack$(ARCH).a
   $(CC) $^ $(SIMTRACK_INC) $(SIMTRACK_LIB) $(NLOPT_INCLUDE) $(NLOPT_LIBRARY)
$(ARMA_INCLUDE) $(ARMA_LIBRARY) -lboost_regex -o $@

all: nlopt2 loadQ_track

nlopt2.cpp
        usage: ./nlopt2 label Q0.dat Q1.dat var_params.dat obj_params.dat method > logfile
        input:
                label
                Q0.dat: quadrupole setting files, no error
                Q1.dat: quadrupole setting files, before correction
                var_params.dat:
                        The file of the info of trim quad used. The first row is the index of trip
                        quadrupole used. The index starts from 0. The second and third rows are the
                        lower and upper limit.
                obj_params.dat:
                        The file of the info parameters for objective, including weightings. Comments
                        are preceded by #. The file is self-explanatory.
                method:
                        method options provided by NLOpt. Check NLOpt manual for the methods
                        available.
        output:
                dqba_(label).dat: change of quadrupole strength
                harm_(label).dat: harmonics
```

dnxz_(label).dat: dnu
FODO0_(label).twiss, FODO1_(label).twiss, FODO2_(label).twiss
Q2_(label).dat: quadrupole setting files, after correction

nlopt_obj1.h, nlopt_obj2.cpp

The object class and function the the NLOpt algorithm are defined here. To compile this, BOOST library is needed, in order to deal with the two quadrupoles. The members of the class include the quadrupole index and strength, the BPM index, the stopband harmonics, the twiss functions, tunes, etc. Not all are used in each case.

FODO4.h, FODO4.cpp:

The lattice is defined here. It is a 18 superpeiod, 360 FODO ring. An initial quadrupole strengths are given but they can be changed in the main program. The starting point is set in a drift section. The space charge kickers are the element type MARKER named SPKICK. They are placed exactly every 3 meter along the ring. Eighteen BPMs are placed in every superpeiod. Because the quadrupoles are split into two pieces, many routines dealing with the quadrupole strengths I/O has to be carefully managed.

loadQ_track.cpp

This program loads the quadrupole setting file and sets it into FODO4 lattice and performs multiple tracking. The 17 input arguments are explained as follows:

1. label
2. epsx: initial horizontal rms emittance
3. epsz: initial vertical rms emittance
4. N0: line density
5. N_particle: number of macro particles
6. N_TURN: number of turn to be tracked
7. N_INJECTION: injection turn numbers
8. apx: horizontal aperture
9. apz: vertical aperture
10. Qsetting0: Quadrupole setting withoug error. This is used to generate the particle distribution.
11. Qsetting1: Quadrupole setting for the tracking.
12. TBT_OUT: a flag to write the turn-by-turn data ouput tbt_(label).dat.
13. RAW_OUT: a flag to write the raw data ouput raw_(label).dat. The file is big.
14. SBL_OUT: a flag to write the stable flags ouput sbl_(label).dat.
15. EBE_OUT: a flag to write the element-by-element output ebe_(label).dat. This is for the FFT analysis and the file is very huge. We usually don't save this file after the frequency info was analyzed and extracted.
16. DNU_OUT: a flag to write the tune shift parameters ouput dnu_(label).dat.
17. ENV_OUT: a flag to write the envelope coordinates output env_(label).dat.

The last six arguments are either 1 or 0 in order to switch on/off the outputs for post data process. This particle spice and energy are assigned in the program. The initial distribution are truncated Gaussian distributed and scaled with the beta functions of the bare lattice. The other tricks include:

1. dynamically estimate the beam size at SPKICKs
2. the smoothization of the emittance according to the previous 108 SPKICKs.
3. Zero mean at every SPKICK to keep the beam centered.

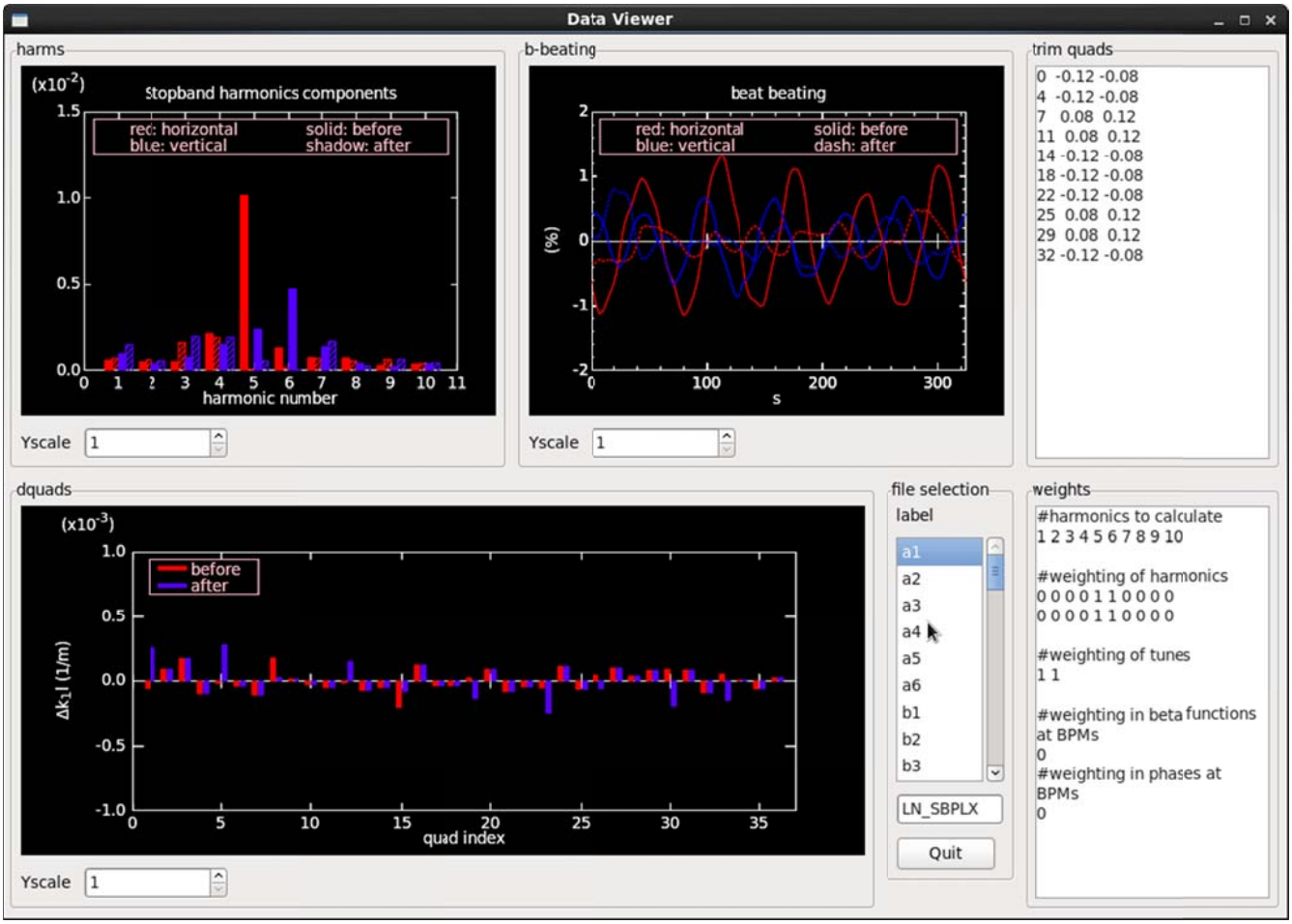
Qerrgen.cpp

usage: ./qerrgen label nux nuz Qerr scale
This program generates the quadrupole setting files for the script runQ.sh. The input arguments are:

1. label
2. seed: random number seed
3. nux: horizontal tune
4. nuz: vertical tune
5. Qerr: relative quadrupole error
6. scale: scaling wrt to Qerr

The output is Q_(label).dat and FODO_(label).twiss

Screenshot of viewer:



j