SPheRIO manual

(Dated: April 2010)

SUMMARY

This document is the programing manual for the FORTRAN77 version of SPheRIO which implements entropy representation of the Smoothed Particle Hydrodynamics (SPH) method for relativistic high-energy collisions. The code has been investigated and developed within the São Paulo - Rio de Janeiro Collaboration. SPheRIO is the shorthand of Smoothed Particle hydrodynamical evolution of Relativistic heavy-IOn collisions.

Section I gives a bird's-eye view of the program stucture of SPheRIO. A complete explanation of option file OPTNS can be found in section II. Section III is devoted to explain the conventions and interfaces used in EOS (equation of state) tables. Section IV focuses on the details of functions and algorithms of each subroutine. In Section V, some simple but practical examples are given to show how one may connect his own IC (initial conditions) or EOS with SPheRIO by complying with the code's interface standard.

I. MAIN PROGRAM STRUCTURE OF SPHERIO

SPheRIO is an implementation of hydrodynamic model of nucleus-nucleus collisons based on SPH method. The main reference for SPheRIO is

Y. Hama, T. Kodama and O. Socolowski Jr., Braz. J. Phys. 35 (2005) 24.

and the reference for SPH method is

C.E. Aguiar, T. Kodama, T. Osada, Y. Hama, J. Phys. G27 (2001) 75.

The present release of SPheRIO (v4.01) was built in such a way that it can be easily utilized and managed by someone who is unfamiliar with the source code. On the other hand, to further develope the code or to incorporate SPheRIO into another program can also be achieved by an experienced programmer without much effort. SPheRIO in its present form consists of several blocks as shown in Fig.1. Each block possesses its specific functionalities, meanwhile it can be seen as almost independent from the rest of the code. In fact they can be compiled independently and only communicate with other parts of the code through well defined interfaces. Therefore, debugging one of the blocks in principle will not affect the remaining part. Several blocks were implemented so that one may exchange them for his own codes with little effort by simply implementing some interface protocols explained in this document. Concerning the output format of the code, we decided that it should be determined in a way as to be sufficient and convenient to connect to other open source codes on the market which deals with particle decay and rescattering. In this spirit, we provide several different output formats. This section is devoted to explain the structure of SPheRIO and the basic idea of its communication interface.

SPheRIO is a program simulates hydrodynamic evolution in nucleus-nucleus collisions. Its EOM is based on Smoothed Particles Dynamics. For pratical reasons, the master routine of SPheRIO, namely, spherio() is embodied in a small shell script, namely, aamain(). It is the

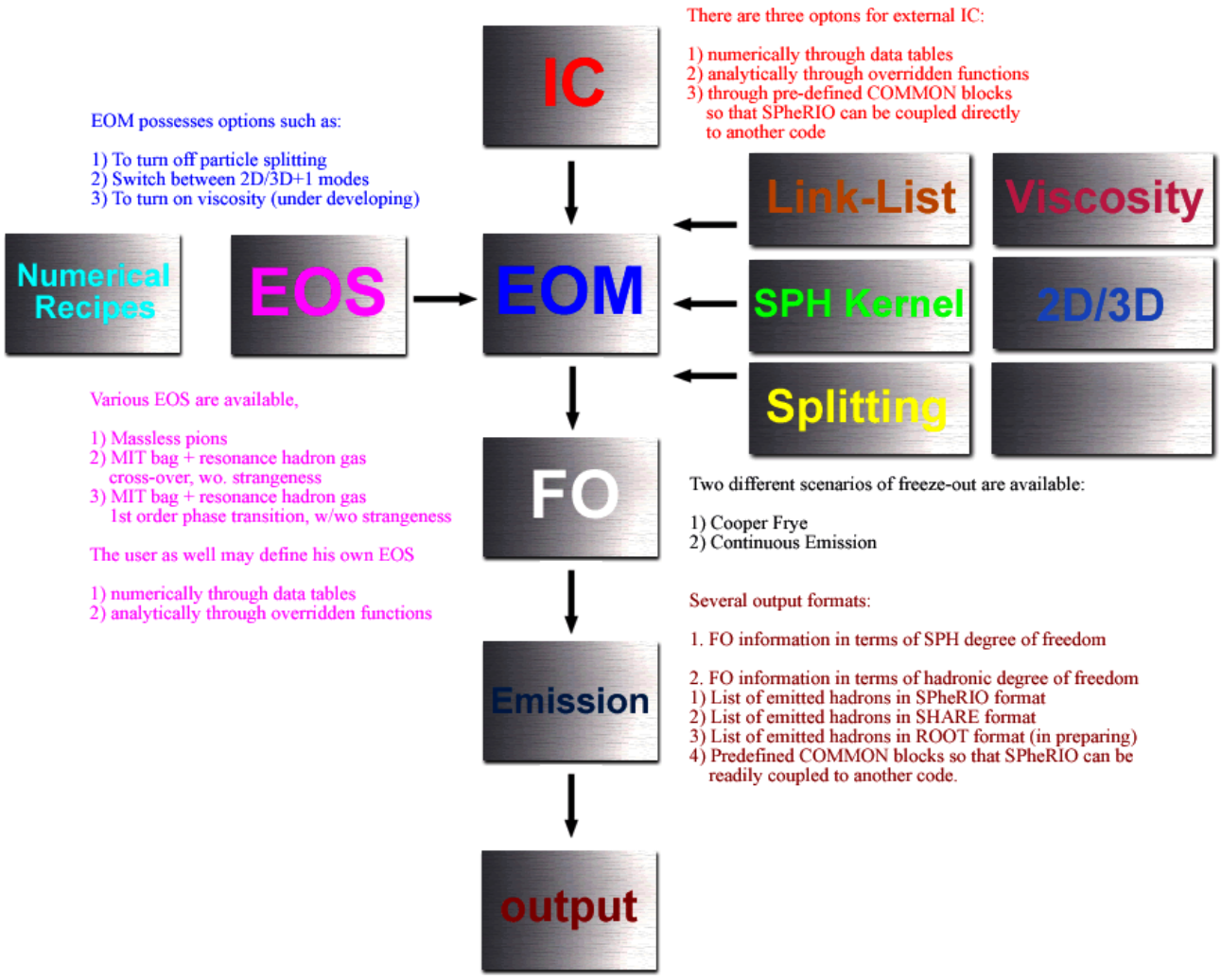

FIG. 1: Block diagram of SPheRIO programming structure.

place where most of the initializations are done and the parameters are read and interpreted by the aread()[1]. From the shell, SPheRIO is called not once but *nevent*[2] times, where $nevent = nfreeze \times nfull$, $nfull$ is the number of full event, and $nfreeze$ is the number of freeze-out Monte Carlo freeze-out procedure which emulates the hadron emission. When the code is run in an event by event fashion, $nfull$ should be defined as the number of total events. Effectively, one may consider there are two loops, the outer loop goes from 1 to $nfull$, for each outer loop, the inner loop goes from 1 to $nfreeze$, i.e., each full collision event possesses $nfreeze$ Monte Carlo processes. To be more specific, each time one begins an outer loop, spherio() is evoked, it calculates the full hydrodynamic evolution from scratch and stores all the information on the freeze-out surface, then the hadron emission is carried out. The results are directed to the output file or to other means through xellwrite().

[1] The subroutine aread() to a great extent adopts NEXUS/EPOS' input technique, we are very thankful for the helps from Prof. Klaus Werner.

[2] Variable names and keywords are denoted by italic letters when it appears for the first time in the text

Starting from the second inner loop, SPheRIO only computes hadron emission, based on the results on hydrodynamic evolution it has already obtained and saved. The successive inner loops repeat the process of the second inner loop until a new outer loop starts, where some (presumably) different IC are put into action. The double loop structure ends when it finally enumerates all outer loops.

To discuss the structure of SPheRIO, a good place to start is the master routine spherio(), since lots of important subroutines of SPheRIO are invoked here. These subroutines appear in the code in the same order as the run-time calculations develop. It contains the following major blocks, as shown in Fig.1:

- IC(initial conditions)
- EOM (equation of motion)
- FO (freeze-out)
- hadron emission
- FO info output
- HBT radius

There are some other blocks which have distinct features or independent functionalities, however they are coded in such a way that they are either partially integrated within or frequently invoked from different parts of the code. Therefore it is a good idea to separate them from the rest of the code and treat them individually. Those blocks are:

- link-list method
- particle splitting (it is literally called particle decay in the code)
- kernel functions
- EOS (equation of state)
- pure numerical recipes

Last but not least, to debug SPheRIO, one also has to know the following aspects

- input interface between SPheRIO and end-users
- SPheRIO programming style

Master routine

spherio()

The master routine spherio() obtains parameters handed down from the shell script through the optns file as well as through spherio.set and spherio.inc. The values of some parameters are defined for a second time according to the setting of $ibugflg$. It is a convention in the code that those values determined via $ibugflg$ settings have a high priority than those defined by optns file or spherio.set. The reason is that it provides a way to set up a group of parameters in a single step as a specific scenarios for debugging or other practical purpose, therefore one doesn't have to go through every single parameter contained in the option file which might be error prone. For example, ibugflg = 08 defines an environment to run the code with EOS of massless pions in the 2D+1 fashion, namely, the EOM will be solved only on the x-y plane, and using the Bjorken scaling solution for the η direction, and the code will print out the profile of hydrodynamic evolution of the system at pre-defined time instants $\tau = 1, 3, 5, 10$. Since this setting is frequently used, it is more convenient to simply set ibugflg = 08 to activate above set of pre-defined parameters instead of set by hand their values one by one in the optns file, namely, set ieos 04 set itrobmode 01 set idimension 2 et. al. The master routine spherio() creates and controls the main surveillance file ztr.data, which prints the de facto value of all the major parameters at run-time and monitors the execution of program with necessary debugging outputs and warning messages. Depending on the parameter $ieos$, spherio() loads correspondingly the EOS tables, and some preliminary tests are made. An energy density rescaling file might be applied at this point accroding to certain parameters. Typically, config() or config_pion() is called to initialize the lattices, SPH particles are consequently created and assigned to the grids with their respective velocity, energy and entropy values. Then the time loop begins to calculate the hydrodynamic evolution. Here we come across two important subroutines: solvediff() and lssf(). They are the meat and bread of SPheRIO. solvediff() deals with equation of motion (EOM), using 2nd order Adams-Moulton method. lssf() handles freeze-out. These two subroutines have to be invoked at the same time, and each one depends on the result of the other. This is because freeze-out need the information on EOM, and EOM might be affected by freeze-out when frozen or low temperature SPH particles are taken away from the fluid. At the end of hydro evolution, setevg() are evgen() are invoked to treat hadron emission and pass the information back to the shell script. HBT radius is computed by evghbt() as needed.

Initialization

rfactor_*(), config(), config_pion(), latt(), loadeos(), machconeini(), modeprint(), newvel(), rfactor(), rfactorprint(), rfactorset(), seteosmode(), set_flagce(), set_flagxn(), setin(), setin_kodama(), setin_pion(), setkappa(), setlog(), settautrans(), settempsph(), settempsphdenovo(), settrinimode()

EOM tables are loaded by the subroutine loadeos(). The space-time are divided into discrete lattices by config() or config_pion(), each of which further calls latt(). latt() has within itself a loop enumerating all the lattice sites, this is when setin() or setin_kodama() or setin_pion() is executed to assign dynamic quantities (four velocity) and thermodynamic quantities (entropy density, energy density, volumn and etc) to the corresponding SPH particle sitting on the lattice site. There are other subroutines handle the initilization of the parameters such as freeze-out temperatures, rescaling factor, initial transverse velocity and

various flags.

SPheRIO provides three different methods for user defined IC. The first one is to make use of the current lattice setting. The user may define $nxico, nyiconzico$ for the number of sites, $xminico, xmaxico, yminico, ymaxico, zminico, zmaxico$ for the initial size of the system, and *tempoico* for initial time τ_0 of hydrodynamic evolution. Next, one should provide a file which contains the information on energy density ($IcoE$), flavor density ($IcoF$) and velocity distribution ($IcoV$) on the lattice sites. The name of the file can be passed to the code by the keywords: "fname ico $desired\_filename$". The structure of the file can be easily inferred from the following subroutine which reads it.
open(97,file=fnio(1:nfnio),status='old')

read(97,*) iversn !Version of the file, irrelevant
read(97,*) laprojx,maprojx,latargx,matargx !Proton number and Mass of the incident nucleus and the target
read(97,*) engyx !Incident energy
read(97,*) bminimx,bmaximx,ikolmnx,ikolmxx !Maximal and minimal value of impact parameter
read(97,*) tauicox !Initial time for hydrodynamic Evolution
read(97,*) iabs_ninicon !Irrelevent
read(97,*) nxicox,nyicox,nzicox !Number of sites in x y z directions
read(97,*) xminicox,xmaxicox,yminicox,ymaxicox,zminicox,zmaxicox !Initial size of the system
read(97,*) (((IcoE(ix,iy,iz),ix=1,nxico),iy=1,nyico),iz=1,nzico),((((IcoV(i,ix,iy,iz),i=1,3),ix=1,nxico),iy
close(97,file=fnio(1:nfnio),status='old')

To explicitly tell SPheRIO know that the IC file is provided, use the keyword "set icomode 01". If nothing is ever assigned to *icomode*, the default value is 01.

The second method is to pass the variables through COMMON blocks defined in *epos.incico*. To make use of this "dangerous" option, user should compile their own code which implements the COMMON block and link it to SPheRIO (through Makefile). This alternative makes it easier to connect SPheRIO directly to an event generator. In this way, one does not need to write IC into a file and later tell SheRIO to read it. It provides efficiency especially when one uses event by event IC. To tell SPheRIO that IC is passed using this method, use keyword "set icomode 00". Warning: if *icomode* is set to 00, but nothing is implemented for the COMMON block, there is no way for SPheRIO to figure this out by itself, in this case no error message will be triggered directly.

The third method is to override the subroutine config_override() of SPheRIO. It gives user the biggest freedom to define IC from scratch. For instance, SPH particles don't have to be initially distributed uniformly in space. As a price to pay, the user must explicitly define how to divide the lattice sites and also initialize and distribute SPH particles by hand. Since the purpose of this subroutine is to be overridden by the user, the only communication between the subroutine and the rest of the code are the interface of the subroutine, it is not recommend to utilize COMMON blocks to pass data. The present form of the subroutine gives an example of what quantities are expectly to be defined in it. To tell SPheRIO that IC is provided by overriding config_override(), use keyword "set icomode 02".

Hydrodynamic evolution

dqfn(), dqfn_xy(), findbt(), findbt_xy(), firststep(), fldobsrv(), fldobsrv_fin(), outp_raw(), rung(), solvediff(), tranisotropy(), trobsrv(), trobsrv_old(), trobsrv_pion()

This block serves two purposes. Firstly, to solve the hydrodynamic equation Eq.(100) on Yogiro's paper "Topics on hydrodynamic model of nucleus-nucleus collisions" (hep-ph/0407264). Secondly, maintenance of SPH particle data for itself and other blocks such as freeze-out, particle splitting, link-list and etc. The code employs 2nd order Adams-Moulton method to solve the second order ordinary differential equation. A nonlinear equation is solved to obtain velocity from momentum (This is not necessary since one may derive the EOM in terms of μ_T). The variable chosen to be solve is $\beta = ln(\sqrt{1+\mu_T^2} + \mu_T) = -ln(\sqrt{1+\mu_T^2} - \mu_T)$. Since ν_i cancels on both sides of EOM, for convenience, the canonical momentum q is defined as pi_i/ν_i. The subroutine dqfn() is used to calculate the r.h.s. of Eq.(100). rung() implements the 2nd order Adams-Moulton method (though it is named after Runge-Kutta by a mistake), which makes use of numerical recipe rtflsp1(). At first time step, firststep() is invoked to prep the loop. And solvediff() controls overhead. To sum up, solvediff() invokes firststep(), rung(), dqfin() to solve the EOM, while maintenance of data is taken care of by itself. Two important subroutines dectrl() and dectrl_off() are embodied in solvediff(), which we will come to later when discussing particles splitting block.

Freeze-out

checkce(), checknu(), checkxn(), flintp(), lssf(), normal(), normal_ce(), probce(), probce_gen(), recordxn(), recordxn_inspector()

lssf() is the meat and potatoes of freeze-out block. lssf() treats Cooper Frye freeze-out scenario within itself, it calls checkce() to handle continuous emission, and checkxn() to implement chemical freeze-out.

Hadron emission

checkodd(), checkomg(), checkv2(), density(), evgen(), evgen_ic(), lorenz(), lorenz_ic(), pdmax(), pdmax_ic(), peso(), pesotest(), pesotrials(), prodis(), prodis_ic(), rot(), rot_ic(), setevg(), setevg_ic(), setfac(), setnex(), settot(), settot_ic(), setus(), weigpr(), weigpr_ic()

The hadron emission block involves two parts. The first part uses statistical distribution function to estimate number of each species of hadrons. The subroutines responsable for this are peso(), peso_ic(), weigpr(), weigpr_ic(), setevg() and setevg_ic(). The second part employs random number generator to produce the hadrons, and the data are sent back to NEXUS. These subroutines are prodis(), prodis_ic(), pdmax(), pdmax_ic(), evgen(), evgen_ic() and setnex(). One thing woth mentioning is that when HBT radius is calculated, the second part is swtiched off to save CPU time, since this part of code will repeatedly be executed many times when nfreeze is big. HBT radius has its own method to deal with hadron emission.

FO infomation output

SPheRIO provides several different methods to output the information on freeze-out surface. The first catalogue involves to output the freeze-out surface itself. For the time being, SPheRIO only provides to output the information in terms of SPH degrees of freedom. To active this method, use "write fzoutTable" in the OPTNS file, and utilize "fname fzout $the\_desired\_filename$" to define the filename. In contrary, use "read fzoutTable" to read freeze-out surface information from a file, in this case, the hydrodynamic evolution will be skipped by SPheRIO.

The second catalogue is of greater useness, it prints out a list of hadrons emitted on the freeze-out surface. Since there are seveval conventions for programatic particle identification, SPheRIO makes some effords to implement most popular conventions. Firstly, use "switch xell on" to swith on output. use "set ifzlistfmt 01" for SPheRIO particle talble conventions, "set ifzlistfmt 02" for NEXUS/EPOS particle talble conventions, "set ifzlistfmt 03" for THERMINATOR/SHARE particle talble conventions, "set ifzlistfmt 04" for ROOT particle talble conventions (under development). Emitted hadron list will also be available through the COMMON block in the near future.

HBT radius

corr(), evghbt(), fndk(), raplist()

Eq.(120) and Eq.(122) of the paper "Topics on hydro..." are used to calculate the coefficient C2. A link-list method in rapidity space is implemented in the subroutine raplist().

Link-list method

bsqd(), bsqd_xy(), fins()fins_xy(), finsb_gen(), finsb_gen_xy(), mklt(), prcn()

The method can be found in Numerical Recipe book. The goal of this block is provide a efficient way to calculate any dynamic or thermodynamic quantity at any given point. subroutine mklt() is to build the link-list, prcn() is to pick up nearby SPH particles using the link-list, while finsb_gen() among others is to calculate the quantities at a given point.

Particle splitting

decay(), dectrl(), dectrl_off()

Particle splitting is introduced due to the fact that SPH particle might not be evenly distributed in space due to the hydro evolution. Note that the kernel function we employed expands 4hxy. If in a cube of $(4hxy)^2(4het)$, the number of SPH particles is smaller than 64 (this estimation is not exact though), the smoothness of SPH formulation is questionable, especially when some SPH particles carry large entropy. (However, it is arguable whether it is the right way to heal the problem. An alternative option is that every SPH carries the same amount of conserved quantity eg. entropy and there is no decay. Initially SPH particles are not uniformly distributed, that the number density of SPH particles is proportional to entropy density. In this case, sometimes a very big SPH particle number is needed to achieve desired resolution.) dectrl() determines when SPH particle splitting takes place,

decay() implements the splitting, dectrl_off() is a subroutine to skip particle splitting for debugging purpose.

Kernel functions

krln()

The present kernel functions adopted by SPheRIO are based on cubic spline. They can be found along with a few discussions in Eq.(39) in the article "A pedagogical tool using SPH to model fluid flow past a system of cylinders" by Brain Schlatter.

Equation of state

bdy0read(), bdyread(), digread(), eosph(), eosph_original(), eosph_pion1(), eosph_pion2(), eosph_pion(), eosph_pion_bag1(), eosph_pion_bag2(), eosph_strangeness(), eosphdbg(), eosphtrials(), esxread(), geteosesx(), geteosmix(), getesx(), gettmunew(), ingues(), mixdphs(), mixread(), nxpd(), pdtable(), phsjdg1(), phsjdg2(), phsjdg1_original(), phsjdg1dbg(), phsjdg2dbg(), phsjdgtrials(), rftzer(), readstb(), readtb(), thintp(), thrread(), tmevol(), tmintp(), tmuread(), usrfun()

See Section III for details.

Several sets of pre-defined EOS are at disposal. They can be chosen through the setting "set ieos *an_eos_number*", where *ieos* is the keyword for EOS setting, and *an_eos_number* is the corresponding EOS index. The first set of EOS makes use of hadronic resonance model with finite volume corrections to describe the matter on the hadronic side, and MIT bag model for quark gluon plasma (QGP) phase. The main part of observed resonances in Particle Data Tables are included in the hadronic phase. Use "set ieos 01" to utilize this EOS. The second set of EOS is the same as the first set of EOS except that local strangeness neutrality is assumed to find strangeness chemical potential. Use "set ieos 02" to activate this EOS. "set ieos 03" is a set of analytic EOS with uses massless pion gas for hadronic phase and MIT bag model for QGP phase. "set ieos 04" gives the EOS of massless pions (π^+, π^-, π^0) for debugging purpose.

"set ieos 07" activates a set of EOS which was inspired by lattice QCD. EOS defined by "set ieos 08" is essentially the same as "set ieos 07" except that local strangeness neutrality is assumed. *an_eos_number*07 and 08 are still under development and will be available soon.

To supply a user-defined EOS, override "virtual" functions eosph_override(), phsjdg1_override() and phsjdg2_override() and apply "set ieos 99".

Pure numerical recipes

findph(), findth(), gammln(), gasdev(), indexx(), itrp2d(), locate1(), locate(), mnewt(), mnewtgetmu(), newton(), phif(), poidev(), polint11(), polint(), ran1(), ran3(), rtflsp1(), rtflsp2(), rtnewt(), velequ()

In the following Section IV, I will try to discuss each subroutine in detail.

II. THE OPTION FILE INTERFACE

The option file *.optns serves as an interface to pass parameters to the code at run-time. Due to historical reasons, SPheRIO adopted the option file of NEXUS and went ahead to add more features of its own.

In the following, I will list all the keywords provided by SPheRIO[3].

#define bim03 2.55 #define bim05 3.32 #define bim06 3.64 #define bim10 4.73 #define bim15 5.81 #define bim20 6.68 #define bim25 7.46 #define bim30 8.17 #define bim35 8.83 #define bim40 9.42 #define bim45 10.00 #define bim50 10.55 #define bim60 11.57 #define bim70 12.48 #define bim80 13.37 #define bim92 14.57

set bminim 0.0 set bmaxim bim06

!!above definitions set the collision's impact parameters at the first PHOBOS centrality window: 0parameters of PHOBOS and STAR centrality windows, it will automatically find and utilize correct rescaling factor (to adjust the pseudo-rapidity distribution of all charged particles) as well as the freeze-out temperatures, using linear interpolation, so just un-remark one of the lines in the option file I sent. For more information read comment of ibugflg

set laproj 79 set maproj 197 set latarg 79 set matarg 197

!!above statement set, from left to right, number of projectile protons to 79, number of projectile nucleons to 197, number of target protons to 79 and number of target nucleons to 197. Therefore it literally reads, Au-Au collisions.

set ecms 200.0

!!it sets collision energy at 200A GeV

set nfreeze 10

!!it sets the number of Monte Carlos freeze-out procedures to 10, corresponding to one full hydrodynamic evolution.

set tempoico 1.0

!!initial time τ of the IC when hydro takens over

set nxico 25 set nyico 25 set nzico 25

!!it sets the number of lattice sites in x, y and η direction respectively

set xminico -8 set xmaxico 8 set yminico -8 set ymaxico 8 set zminico -6 set zmaxico 6

!!the initial size of the system in question

fname ico ../optns/auau-epos-pj1.ico

!!ico file to read as an input (not the output file)

switch mprcsion off

!!run the code with the highest possible precision but very very slow

set icomode 00

!!00 - IC information is passed to SPheRIO through COMMON blocks

!!01 - default value, IC information is passed to SPheRIO through ICO file (filemane defined by fname ico)

!!02 - IC information is passed to SPheRIO through overriding of config_override()

set hxy 1.0 set het 1.0

!!set hxy and het, the size of SPH particles

set dtau 0.25

!!*dtau*, time step interval

[3] Here some features are adopted or inspired by NEXUS/EPOS, credits are given to Klaus Werner and co-authors

set ibugflg 20

!!debugging/run-time mode switcher, ibugflg has the highest priority, it may override any other parameters

!!00 - normal mode, optns file determines all the parameters, inital time, thermal/chemical freeze-out temperatures and etc

!!01 - run the code with original EOS without strangeness and corrected equation of motion

!!02 - run the code with new EOS with strangeness and corrected equation of motion

!!03 - reserved

!!07 - reserved

!!08 - run the code with EOS of massless pions, the EOM will be solved only on the x-y plane, and using the scaling solution for the eta direction, one may also use the switch "skip icotable" in the option file, since one must provide a 2-dimensional IC

!!09 - run the code with EOS of massless pions with bag model containing a first order phase transition, using the Gaussian-like IC

!!10 - run the code with EOS of massless pions with bag model containing a first order phase transition, using NEXUS IC

!!11 - run the code with new EOS with strangeness and using the Gaussian-like IC

!!12 - run the code with new EOS with strangeness and using the NEXUS IC, it outputs some informations on fluid evolution

!!13 - run the code with new EOS with strangeness and using the NEXUS IC, it outputs more informations on fluid evolution

!!15 - run the code with EOS of massless pions, the EOM will be solved only in two dimensions

!!20 - run the code with new EOS with strangeness and using the NEXUS IC, the thermal/chemical freeze-out temperatures are automatically chosen by the code, according to the impact parameters

!!21 - the same as 20, except that it is used to calculate v2 and the yields of some particles are multiplied by a factor of 10

!!30 - run the code with original EOS without strangeness and original equation of motion and core fuction

set iprtflg 9

!!iprtflg determines the debugging message output level, bigger number implies more detailed info

set tempqmout 0.145

!!freezeout temperature for chemical freezeout

set tempfzout 0.135

!!it sets the temperature of thermal freeze-out, this value can be overridden by ibugflg

set tempfinal 0.090

!!it sets the lowest temperature the fluid can reach, this value is essentially determined by the boundary of EOS table

switch emcontinua off

!!run the code using thermal freeze-out scenario, this switcher has been modified from Otavio's version to match the style of the rest of the program

!switch emcontinua on

!!run the code using continuous emission scenario

set kappa 0.30

!!parameter kappa determines how easily a particle can escape from the fluid, see eq.(123) of Yogiro's paper: Topics on Hydro...

set idecomode 01

!!01 - isotropic original formulae for continous emission

!!02 - anisotropic momentum dependent formulae, momentum dependent formulae, it determines how hadrons emit from the frozen-out SPH particles the first choice "isotropic" uses regular isotropic formula, which takes into account fluid velocity, momentum of emitted particle, isentropic surface and free Fermi/Boson distribution, see eq.(111) of Yogiro's paper: Topics on Hydro... the second choice takes into account that even the free Fermi/Boson distribution is not isotropic, it can be expressed in terms of interaction distribution as $f_{free} = CP/(1-P)f_{int}$, where P is the probability of emission and it is anisotropic, and C is a constant regarding normalization, for details see comments of the code

set itrinimode 00

!!this swtcher provides user with some predefined IC to choose from, it has four options, pure NEXUS IC, NEXUS plus some initial transverse velocity using two different formulae and Mach Cone IC

!!00 - pure NeXus initial transverse boost

!!09 - NeXus+tanh() initial transverse boost

!!11 - a way of transverse boost using rapfactor

!!15 - Mach Cone initial conditions

switch sphsplit off

!!turn off SPH particle splitting

switch freezeout on

!!to evoke the freeze-out process or simply skip it

set ifzlistfmt 3

!!print out format of hadron emission (1)spherio (2)NEXUS/EPOS (3)THERMINATOR/SHARE (4)ROOT

!read fzoutTable - fname fzout ../optns/media-auau-pj1.fzo.data

!!fzout file to read as an input file !write fzoutTable

switch quimfz on

!!employ chemical freezeout, when chemical freeze-out is on, the code further provides four type of chemical freeze-out scenarios, namely

set iquimmode 06

!!01 - chemical and thermal freeze out of hyperon at hadron phase boundary

!!02 - chemical and thermal freeze out of hyperon at qgp phase boundary

!!03 - chemical and thermal freeze out of hyperon at temperature tempqmfzout

!!04 - chemical of hyperon at tempqmfzout, thermal freeze out at tempfzout

!!05 - scenario specifically for omega

!!06 - the same scenario as 04 but for all particles

switch ihbt off

!!when this switcher is on, it calculates hbt radius

set ieos 02

!!this parameter sets which EOS is to be used, it can be overridden by bugflg

!!01 - 1st order MIT bag + hadron resonance without strangeness

!!02 - 1st order MIT bag + hadron resonance with strangeness

!!03 - 1st order MIT bag + massless pion

!!04 - massless pion gas

!!07 - Lattice inspired MIT bag + hadron resonance without strangeness
!!08 - Lattice inspired MIT bag + hadron resonance with strangeness
!!99 - User-define EOS
set ihydromode 00
!!00 - original mode, SPH decouples at Tfin
!!01 - violent mode, SPH decouples at freeze-out surface
!!This parameter gives options to run the code using two different EOM. In the first case, when it sets to zero the SPH particles continuous interacting with others even after freeze-out in the second case, when it sets to 1, frozen-out SPH particles are taken away from EOM, they remain their velocities when they decouple from the fluid, although they are counted when calculating the probability of particle emission
set itrobmode 00
!!this parameter controls the level of outputs of information on the evolution of the system the output file contains velocity, entropy density, temperature and other thermodynamic quantities of the fluid and SPH particles
!!00 - no output
!!01 - standard output at $\tau = 1, 3, 5, 10$, output on x and y axis
!!02 - standard output at $\tau = 1, 2, 3, ...$, output on x and y axis
!!08 - specific output for massless pion gas, output and raw output on transverse plane, corresponding to ibugflg 08
!!09 - specific output at $\tau = 1, 4, 7, 10$, raw output on x, y and r direction, corresponding to ibugflg 09
!!12 - specific output at $\tau = 1, 3, 5, 10$, output on transverse plane, corresponding to ibugflg 12
!!13 - specific output at $\tau = 1, 2, 3, ...$, output on x and y axis and raw output on individual SPH particle, corresponding to ibugflg 13
!!14 - specific output at $\tau = 1, 2, 3, ...$, output on transverse plane
!!15 - specific output for massless pion gas, corresponding to ibugflg 15
set idimension 3
!!3D calculation or 2D limit with specified IC
set iviscosity 01
!!00 - ideal hydro dq non-linear solve
!!01 - ideal hydro dq iteration
!!02 - bulk viscosity
!!03 - shear viscosity and bulk viscosity

III. CONVENTION OF EOS DATAFILE IN SPHERIO

The EOS data files are made more complicated than it would have been. The reason is that SPheRIO need to know in which the phase a certain SPH particle locates when dealing with the freeze-out. (For example, in QGP phase the SPH particles do not suffer freeze-out in "Continuous Emission" scenario)
The data files concerning EOS infomation are
bdy-hrg0.dat
bdy-qgp0.dat
bdy-hrg.dat
bdy-qgp.dat

eos-eq.dat
eos-er.dat
eos-sq.dat
eos-sr.dat
mix-ee.dat
mix-tp.dat
stabl1.dat
stabl2.dat
stabl3.dat

bdy-qgp.dat and bdy-hrg.dat contain the information about boundaries between QGP phase/mixed phase and hadronic phase/mixed phase. Both of them are one-dimensional. I will take bdy-qgp as an example, because the data structures in both two files are very similar. The first line contains the number of columns, eg. 200 The second line was just a comment line, however, please remain it unchanged, since the subroutine to read the file is not very smart. After that was successive 200 lines, each of them contains 15 numbers, they are temperature (xtm) – the only variable since it is one dimensional baryon chemical potential (xmub) strangeness chemical potential (xmus) energy density on the QGP side (eedstqgp) entropy density on the QGP side (tpdstqgp) baryon density on the QGP side (bbdstqgp) strangeness density on the QGP side (ssdstqgp) strangeness fraction (= strangeness density / baryon density) on the QGP side (xfsqgp) pressure on the QGP side (ppdstqgp) energy density on the hadronic side (eedsthrg) entropy density on the hadronic side (tpdsthrg) baryon density on the hadronic side (bbdsthrg) strangeness density on the hadronic side (ssdsthrg) strangeness fraction (= strangeness density / baryon density) on the hadronic side (xfshrg) pressure on the hadronic side (ppdsthrg=ppdstqgp) If one draws the phase boundary in terms of temperature vs. baryon chemical potential, there are two curves corresponding to the boundary to QGP phase and hadronic phase. This is the reason that one need two files to store the information. In the case when strangeness is not considered, the data in the two files are the same.

bdy-hrg0.dat and bdy-qgp0.dat store the information at T=0 of hadronic phase and qgp phase. Pratically, due to numerical issue, (Although one can solve strictly at T=0, but at very small T it turned out to be difficult.) the temperature was taken as 0.01GeV. Take bdy-qgp0 as an example, The first line contains the number of columns, eg. 200 The second line was just a comment line, however, please remain it unchanged, since the subroutine to read the file is not very smart. After that was successive 200 lines, each of them contains 7 numbers, they are temperature (xtm) – the only variable since it is one dimensional baryon chemical potential (xmub) strangeness chemical potential (xmus) energy density (eedstqgp) entropy density (tpdstqgp) baryon density (bbdstqgp) pressure (ppdstqgp) For QGP phase, the baryon density to start with is not necessarily zero, so I start with a number little smaller than the deconfinement transition, 0.001fm^{-3}

eos-eq.dat, eos-er.dat, eos-sq.dat, eos-sr.dat, mix-ee.dat and mix-tp.dat are the files store the information in QGP and hadronic phase in terms of energy density/baryon density or entropy density/baryon density. Taking eos-eq.dat for example, e (of eq) stands for energy density, and q stands for QGP phase, so this file contains the EOS information of qgp phase. The first line contains the numbers of two elements of the two-dimensional array, eg. 150 (the loops of baryon density) 150 (the loops of energy density) After that was successive 150×150 lines, each of them contains 7 numbers, they are energy density (edeq) – the inside(second) loop index, according to the file, it goes from 1.6GeV (phase boundary at rhob=0) to 21GeV

entropy density (sdeq) baryon density (rhobeq) – the outside(first) loop index, according to the file, it goes from 0.00001fm^{-3} to 2.5fm^{-3}, note that the pace was not equally divided pressure (ppeq) temperature (tteq) baryon chemical potential (mubeq) strangeness chemical potential (museq)

For the file eos-sq.dat, the data are orgnized in the same order, except one should understand it as energy density (edsq) entropy density (sdsq)– the inside(second) loop index baryon density (rhobsq) – the outside(first) loop index pressure (ppsq) temperature (ttsq) baryon chemical potential (mubsq) strangeness chemical potential (mussq)

stabl1.dat, stabl2.dat and stabl3.dat only concern the particle emission, so in principle they do not need to be modified when one does not concern strangeness or other conserved charges. Note that some data in the tables overlap one another, the names of the variables are adopted from the subroutine readtb(). The conventions are as follow,

temp - temperature chemb - baryonic chemical potential utb - chemical potential of proton divided by temperature utab - chemical potential of proton-bar divided by temperature denom - the correction factor on the denominate, when involving finite volume correction. See formulae in Yogiro's paper: Topics on hydro... It is worth noting that finite volume correction eq.(28-29) of the paper "topics on hydro model of nucleus-nucleus collsions" are used, that's why utb != -utab

ut - chemical potential divided by temperature mt - mass divided by temperature xintg - baryonic number density divided by temperature cube This table is used to save time in the calculation, note xintg is a quantity that stays constant when ut and mt are given

xx_T - temperature xx_ub - baryonic chemical potential xx_P - pressure The dimension of the table are determined by ntb1x and ntb1y, volume corrections were considered.

In the new version of SPheRIO I added data of chemical potential and modified some structure of the last three tables as well as removed some overlapped data. But the basic idea remains the same.

IV. FUNCTION AND ALGORITHM OF SUBROUTINES

In the following, we will discuss the function and algorithm of each subroutine. Some efforts are made to adopt the comment style of "Numerical Recipes": instead of telling trivial detials of what each statement line is doing, we try to focus more on how the subroutines fit and serve in the whole program and why certain algorithm is employed.

Some in-line comments can be found in SPheRIO's source files and updates are logged on top of the subroutines.

Initialization

rfactor_*(), config(), config_pion(), latt(), loadeos(), machconeini(), modeprint(), newvel(), rfactor(), rfactorprint(), rfactorset(), seteosmode(), set_flagce(), set_flagxn(), setin(), setin_kodama(), setin_pion(), setkappa(), setlog(), settautrans(), settempsph(), settempsphdenovo(), settrinimode()

config() divides the space-time into discrete lattices. The range of space-time under consideration can be readjusted in accordance with the value of bugflg, since bugflg has a higher priority than the optns settings. However, the total number of initial SPH particles is determined by nxico, nyico and nzico. These numbers are adopted by SPheRIO as parameters.

In case one wants to change them, edit NEXUS' file nexus.incico. latt() is called at the end of subroutine. The program structure between config(), latt() and setin() is to be improved.

config_pion() serves in the same way as config() except that it divides space-time with cylindrical symmetry and it implements the function of latt() and setin() within itself. For the time being, the subroutine is used in the case of massless pions, however it is easy to be modified and applied to different situations.

latt() has within itself a loop enumerating all the lattice sites, this is when setin() or setin_kodama() or setin_pion is invoked. These subroutines calculate and assign dynamic quantities (four velocity) and thermodynamic quantities (entropy density, energy density, volumn and etc) to the corresponding SPH particle sitting on the very lattice site. Then the total number of SPH particles is estimated and passed to Nsp(1). If one calcuates Mach Cone, the subroutine machconeini() is invoked to distribute an extra SPH particle on top of others.

setin() calculates dynamic quantities (four velocity) and thermodynamic quantities (entropy density, energy density, volumn and etc) at a given lattice site using the NEXUS initial conditions. Firstly, newvel() is called to solve the four velocity by the equation $T^{\mu}_{\nu}u^{\nu} = \epsilon u^{\mu}$. The resulting velocity is used to calculate flavor charge IcoF and invariant energy IcoE. Then the rescaling factor() is applied. Baryon density, electric charge density, strangeness density are evaluated. If an SPH particle stays beyond the boundary of EOS (its phase is indeterminable), it is thrown away. If the temperature of SPH particle is lower than the freeze-out temperature Tf, but higher than the EOS table limit, its properties will be stored in the "ic" arrays for future use. This particle is then excluded from EOM. The variable "nic" stores the total number of these particles. Some initial transverse boost profile is applied according to the value of "itrans".

rfactor() is an energy rescaling factor as a function of spatial coordinate eta. The function calls one of different implementations rfacor_*() according to the value of index variable "irft". When "irft" does not match any value in a pre-defined list, an error echos and the program halts.

rfactor_*() implements the rescaling factor. For instance, rfactor_bug20_t1a0_mus_js1() is used when bugflg=20, $\tau_0 = 1.0$, EOS with strangeness, in the first STAR centrality window 0 - 5rfactor_unit() is a trivial case where the rescaling factor is a constant, 1.

rfactorset() is executed before that rfactor() is invoked by setin(). Its goal is to assign a correct value to "irft" according to collision energy, impact parameters, τ_0, freeze-out scenarios and bugflg. What we intended is to make the choice of rfactor as automatic as possible, obviously, it didn't work out very well. rfactorset() and rfactor() look fat and they are difficult to maintain.

loadeos() loads EOM tables into the code according to "ieosmode".

seteosmode() is executed before loadeos(), it determines the value of "ieosmode" by bugflg.

machconeini() creates an heavy SPH particle with a different velocity and entropy on top of the IC generated by setin(). It is used to creat an initial conditions for the study of Mach Cone.

modeprint() prints the de facto value of all the major parameters in the initilization block into the main surveillance file ztr.data.

newvel() solves the four velocity by the equation $T^{\mu}_{\nu}u^{\nu} = \epsilon u^{\mu}$.

rfactorprint() prints out the function of rescaling factor vs. eta, for debugging purpose.

set_flagce() and set_flagxn() set initial "0" to freeze-out flags "iflgce" and "iflgxn".

setkappa() sets the value of "kappa", the parameter in Eq.(123) on Yogiro's paper "Topics on hydrodynamic model of nucleus-nucleus collisions" (hep-ph/0407264).

setlog() resets all the log files.

settautrans() sets τ_0, the initial time when NEXUS hands down the initial conditions, and SPheRIO starts the hydro evolution.

settempsph() sets thermal/chemical freeze-out temperatures accroding to optns and freeze-out scenarios.

settempsphdenovo() automatically readjusts the thermal/chemical freeze-out temperatures according to the value of bugflg.

settrinimode() determines the initial transverse boost velocity and new EOM scenario where SPH particles are/not taken away from fluid when they are frozen out.

Hydrodynamic evolution

dqfn(), dqfn_xy(), findbt(), findbt_xy(), firststep(), fldobsrv(), fldobsrv_fin(), outp_raw(), rung(), solvediff(), tranisotropy(), trobsrv(), trobsrv_old(), trobsrv_pion()

solvediff() solve the hydrodynamic equation Eq.(100) on Yogiro's paper "Topics on hydrodynamic model of nucleus-nucleus collisions" (hep-ph/0407264). It employs 2nd order Adams-Moulton method to solve the second order ordinary differential equation. To obtain velocity from momentum, a nonlinear equation is made use of (In fact, this is not necessary since one may derive the EOM in terms of uT). In the nonlinear equation, the variable chosen to be solve is $\beta = ln(\sqrt{1+u_T^2}+u_T) = -ln(\sqrt{1+u_T^2}-u_T)$. β goes to infinite when velocity approaches the speed of light, hence the choice of beta ensures the precision at high velocity region. Since ν_i cancels on both sides of EOM, for convenience, the canonical momentum q is defined as pi_i/ν_i. The subroutine also does maintenance of SPH particle data for itself and other blocks such as freeze-out, particle splitting, link-list and etc. In the follows,, we explain how the subroutine works. First, firststep() is invoked to do some initialization. "ihdfull" and "ihdmode" are checked here and from time to time later on to implement the new EOM. The new EOS works by making a link-list excluding those frozen-out particles. "ihdmode" determines whether new EOM optns is turned on, "ihdfull" indicates at run-time whether one should build the link-list with/not those frozen-out particles. The information on SPH particles of two time steps in the past is stored in arrays with "oflg" being -2 and -1. This is not because we make use of a 2nd method to solve EOM but that thermal freeze-out employs three point interpolation. Particle decay is treated by dectrl() or dectrl_off() before the time evolution takes place. The Adams-Moulton method is implemented in two steps corresponding to two instances τ and $\tau + d\tau$. In the first step, "oflg" is set to 1, mklt(), prcn(), fins(), bsqd() are invoked to update the link-list, and compute the thermodynamic quantities of each SPH particle. dqfn() is used to evaluate the r.h.s of Eq.(100). rung() calcuates dynamic quantities. In the second step, the code repeats itself except "oflg" is set to 2 and the time τ evolves. Then the SPH particles are sorted using indexx() in terms of temperature in decreasing order. Since each SPH particle in principle will be assigned to a different index number, the father-son relationship between SPH particles (due to particle splitting) has to be carefully taken care of. In the end, low temperature (T ¡ Tfin) SPH particles are cut away from the EOM. Throughout the subroutine, status of SPH particles as well as total entropy and energy conservation are verified, and the results are directed to the surveillance file ztr.data.

Here is a list of EOM arrays used to store information on SPH particles during the hydro evolution. Wtih the exception of Nsp(oflg) being the total number of SPH particles in EOM, all the rest of them share the same form.

EOM_Array_Name():=EOM_Array_Name(oflg, EOM_SPH_particle_index),

where "oflg" is the status flag: -2 and -1 are used in interpolation of Cooper Frye, 0 is reserved for temporary storage like sorting, 1 and 2 are used in 2nd order Adams-Moulton method. xx()=xsph(1) x coordinate, yy()=xsph(2) y coordinate, et()=xsph(3) eta coordinate, bt()=rsph(1) beta, ph()=rsph(2) phi, al()=rsph(3) alpha, sn(1,) entropy, br() baryon number, sr() strangeness charge, qr() electric charge, ss() proper entropy density, bd() proper baryon density, sd() proper strangeness density, qd() proper electric charge density, mbxn() baryon chemical potential of chemical freeze-out, msxn() strangeness chemical potential of chemical freeze-out, tempxn() temperature of chemical freeze-out, uxn() contracovariant four velocity corresponding to ufcc not ufrs.

firststep() does the initialization. It sorts SPH particles in terms of temperature and prints out run-time value of some quantities in ztr.data.

dqfn() evaluates the r.h.s of Eq.(100), with dqfn_xy() being the 2D implementation of dqfn().

rung() implements the 2nd order Adams-Moulton method (though it is named after Runge-Kutta by a historical mistake). The subroutine calculates the dynamic quantities of SPH particles. It first computes the momentum q, and add to it dq obtained by dqfn(). Then rtflsp1() is employed to solve the equation of beta findbt(). With beta obtained, other dynamic quantities like vx, vy, ve, alpha, phi can be acquired easily. Here is a list of definitions of the variables envolved. $rsph(1) = \beta = ln(\sqrt{1+u_T^2} + u_T)$, $rsph(2) = \phi = atan(vy/vx)$, $rsph(3) = \alpha = 1/2ln((1+\tau v_e)/(1-\tau v_e))$. It is easy to verify that four velocity can be expressed in terms of rsph arrays: $u_0 = cosh(rsph(3) - xsph(3))cosh(rsph(1))$, $u_e = sinh(rsph(3) - xsph(3))cosh(rsph(1))/\tau$, $u_x = cos(rsph(2))sinh(rsph(1))$, $u_y = sin(rsph(2))sinh(rsph(1))$.

findbt() is the equation of beta, with findbt_xy() being its 2D version.

fldobsrv() and fldobsrv_fin() print out longitudinal velocity (alpha) distribution of SPH particles at the beginning and end of hydro evolution.

outp_raw() prints out transverse spatial distribution of SPH particles.

tranisotropy() computes momentum anisotropy as a function of time.

trobsrv(), trobsrv_old(), trobsrv_pion() print out transverse distribution of thermodynamic quantities. The difference between the three subroutines is how to evaluate the quantity at a given spatial point.

Freeze-out

checkce(), checknu(), checkxn(), flintp(), lssf(), normal(), normal_ce(), probce(), probce_gen(), recordxn(), recordxn_inspector()

lssf() is the meat and potatoes of freeze-out block. lssf() treats Cooper Frye freeze-out scenario within itself, it calls checkce() to handle continuous emission, checkxn() and recordxn() to observe chemical freeze-out. Here is how it works. checkxn() and checkce() are called first off. The two subroutines check SPH particles one by one to see if they meet the criteria of freeze-out. If so, the freeze-out flags ("iflgce" and "iflgxn") of those particles switch from "0" to "1" to signal for further treatment. After that, it is a check point of "ihdmode" and "ihdfull" for the new EOM. Then it comes to two successive blocks to handle

continuous emission and Cooper Frye. Since the structures of the two blocks are very similar, we will only focus on the first block of continuous emission. First, the freeze-out flag "iflgce" switches from "1" to "2" to indicate that SPH particle has been taken care of. Information on dynamic quantities (coordinate, velocity) and thermodynamic quantities (entropy, baryon number and their densities, as well as temperature) on freeze-out surface is copied from EOM arrays to freeze-out arrays. The copies are made first for continuous emission then for chemical freeze-out, if the operations are legitimate. The covariant normal vector of freeze-out surface is evaluated by normal_ce() and stored, as well as covariant four vector "sfrs" (It is the term in the summation on the r.h.s of Eq.(111) excluding contravovariant momentum p and statistical distribution f, we will refer to it as normal flow thereafter) . The difference between the block of kinematic freeze-out and continuous emission is that it employs three point to calculate the quantities on freeze-out surface. To meet the kinematic freeze-out criteria, the SPH particle has to stay below freeze-out temperature Tf for two successive time steps. The subroutine comes to end with another check point of "ihdmode" and "ihdfull" for the new EOM.

Here is a list of freeze-out arrays used to store information on freeze-out surface. All of them share similar form.

FO_Array_Name(*):=FO_Array_Name(*, FO_SPH_particle_index).

xfrs(0) τ, xfrs(1) x coordinate, xfrs(2) y coordinate, xfrs(3) eta coordinate, rfrs(1) beta, rfrs(2) phi, rfrs(3) alpha, ssfz() entropy density, snfz() entropy, bnfz() baryon number, mfrs() baryon chemical potential, msfrs() strangeness chemical potential, edfz() energy density, bbfz() baryon density, tempfz() temperature, nrfz(0-3) normal vector, ppfz() probability of escape via continuous emission, sfrs(0-3) normal flow, mfrsxn() baryon chemical potential of chemical freeze-out, msfrsxn() strangeness chemical potential of chemical freeze-out, tempfrsxn() temperature of chemical freeze-out, ufrsxn() uxn of chemical freeze-out.

checkce() invokes probce() to calculate the escape probability of SPH particle and make use of Monte Carlo to decide whether an SPH particle freezes-out. The new EOM are also implemented in this subroutine. When calculation the escape probability, all particles should be taken into account, thus "ihdfull" is set to 1, after that, "ihdfull" is switched back to 0.

probce() evaluates the escape probability of a given SPH particle. It first calculates the percentage of escaping particles pp using a momentum independent approximation Eq.(123). Its first order derivative measures the rate of SPH particle freezing. Taking into consideration that we only select from the free SPH particles in EOM, a factor 1/(1-pp) is included.

probce_gen() returns the escape probability of a given coordinate, in stead of a SPH particle, in this sense this subroutine has more general usage.

checkxn() verifies if a SPH particle suffers chemical freeze-out. If it does, cft flag "iflgxn" is set to 1 and recordxn() is invoked to store the information. There are several senarios of chemical freeze-out, chemical freeze-out may take place on phase boundaries or at a fixed temperatures.

recordxn() makes copy of quantities of chemical freeze-out particles into the arrays with keyword "xn". It uses two point interpolation if chemical freeze-out takes place on phase boundaries, and three point interpolation is employed if chemical freeze-out temperature is adopted. The latter case uses the same algorithm as flintp(). "iflgxn" is set to 2.

normal_ce() evaluates the covariant normal vector of freeze-out surface, it is obtained by evaluting the gradient of isentropic surface. It is straightforward to verify the formulae used in the code, we only note here that derivative of $\gamma = cosh(al - eta)cosh(bt)$ on the denominate results in an extra term which presents itself at the end of the subroutine.

normal() is similar to normal_ce() except that it calculates the normal vector of isotherm.

flintp() does a three point interpolation to extract the value of thermodynamic quantities at the thermal freeze-out temperature.

recordxn_inspector() is a surveillance script check whether recordxn() gives consistent results.

Hadron emission

checkodd(), checkomg(), checkv2(), density(), evgen(), evgen_ic(), lorenz(), lorenz_ic(), pdmax(), pdmax_ic(), peso(), pesotest(), pesotrials(), prodis(), prodis_ic(), rot(), rot_ic(), setevg(), setevg_ic(), setfac(), setnex(), settot(), settot_ic(), setus(), weigpr(), weigpr_ic()

setevg() uses statistical distribution function to estimate emission number of each species of hadrons, energy conservation is not explicitly taken into account here. The subroutine starts a loop to enumerate all the SPH particles, and evaluates the emission number wg() for each species using peso() or weigpr() times a volume correction factor obtained by setfac(). Then the emission numbers are summed up with respect to all SPH particles. If a frozen-out SPH particle is not in the hadronic phase, a correction factor is applied. The results are stored in ntot() array.

setevg_ic() is similar to setevg() but treats the initicial frozen "ic" particles stored by setin().

evgen() employs random number generator to produce the hadrons, and the data are sent back to NEXUS. One thing worth mentioning is that when HBT radius is calculated, the evgen() is skipped to save CPU time, since this part of code will repeatedly be executed many times when nfreeze is big. HBT radius has its own method to deal with hadron emission. In the first place, settot() is invoked to transverse ntot() into an integer taking into account statistical deviations. prodis() is used to calculate the emission probability of a hadron with a certain momentum, where rot() and lorenz() help to transfer the momentum into laboratory frame. If the hadron survives the random draw ran1(), it is accepted and setnex() passes its information to NEXUS.

evgen_ic() has similar function as evgen() except it treats the initicial frozen "ic" particles stored by setin().

setnex() sends the data of emitted hadrons to NEXUS through common arrays.

checkodd(), checkomg(), checkv2() check whether a hadron satisfies certain properties. checkodd() checks if the hadron is a strange particle, checkomg() checks if it is Omega and checkv2() checks if the particle is proton or anti-proton.

rot() and rot_ic() calculate the momentum of hadron in proper frame, but along the directions of axes in laboratory frame, which eases the job of lorenz() and lorenz_ic()

lorenz() and lorenz_ic() carry out the Lorentz transformation from proper frame to laboratory frame of a hadron emitted from a given SPH particle. The input four momentum is ppr(), the output variable is plb().

prodis() and prodis_ic() evaluate the emission probability of a hadron. Anisotropic formulae are adopted in the present version.

pdmax() and pdmax_ic() evalute the maximum of probability of hadrons emitting from a given SPH particle. They do their work in a rough way, the present algorithm is to be improved.

peso() estimates the emitting hadron number using approximating statistical distribution. The momentum integral is carried out using extended Simpson's rule (see Numerical Recipe).

pesotest() and pesotrial() are debugging subroutines.

density() calculates the density of hadron by its chemical potential mass and temperature using Gaussian Quadratures (See Numerical Recipes)

setfac() evalutes the volume correction factor of emission number of hadrons.

weigpr() and weigpr_ic() evaluate the emitting hadron number using Eq.(111), where a part of r.h.s. of equation is already stored in the array sfrs() and made use of. The momentum integral is carried out by extended Simpson's rule.

HBT radius

corr(), evghbt(), fndk(), raplist()

Eq.(120) and Eq.(122) of the paper "Topics on hydro..." are used to calculate the coefficient C2. A link-list method in rapidity space is implemented in the subroutine raplist().

Link-list method

bsqd(), bsqd_xy(), fins()fins_xy(), finsb_gen(), finsb_gen_xy(), mklt(), prcn()

The method can be found in Numerical Recipe book. The goal of this block is provide a efficient way to calculate any dynamic or thermodynamic quantity at any given point. subroutine mklt() is to build the link-list, prcn() is to pick up nearby SPH particles using the link-list, while finsb_gen() among others is to calculate the quantities at a given point.

Particle splitting

decay(), dectrl(), dectrl_off()

Particle splitting is introduced due to the fact that SPH particle might not be evenly distributed in space due to the hydro evolution. Note that the kernel function we employed expands 4hxy. If in a cube of $(4hxy)^2(4het)$, the number of SPH particles is smaller than 64 (this estimation is not exact though), the smoothness of SPH formulation is questionable, especially when some SPH particles carry large entropy. (It is arguable whether particle splitting is the right way to heal the problem. An alternative option is that every SPH carries the same amount of conserved quantity eg. entropy and there is no decay. Initially SPH particles are not uniformly distributed, that the number density of SPH particles is proportional to entropy density. In this case, sometimes a very big SPH particle number is needed to achieve desired resolution.) dectrl() determines when SPH particle splitting takes place, decay() implements the splitting, dectrl_off() is a subroutine to skip particle splitting for debugging purpose.

Kernel functions

krln()

The present kernel functions adopted by SPheRIO are based on cubic spline. They can be found along with a few discussions in Eq.(39) in the article "A pedagogical tool using SPH to model fluid flow past a system of cylinders" by Brain Schlatter.

Equation of state

bdy0read(), bdyread(), digread(), eosph(), eosph_original(), eosph_pion1(), eosph_pion2(), eosph_pion(), eosph_pion_bag1(), eosph_pion_bag2(), eosph_strangeness(), eosphdbg(), eosphtrials(), esxread(), geteosesx(), geteosmix(), getesx(), gettmunew(), ingues(), mixdphs(), mixread(), nxpd(), pdtable(), phsjdg1(), phsjdg2(), phsjdg1_original(), phsjdg1dbg(), phsjdg2dbg(), phsjdgtrials(), rftzer(), readstb(), readtb(), thintp(), thrread(), tmevol(), tmintp(), tmuread(), usrfun()

See Section III.

Pure numerical recipes

findph(), findth(), gammln(), gasdev(), indexx(), itrp2d(), locate1(), locate(), mnewt(), mnewtgetmu(), newton(), phif(), poidev(), polint11(), polint(), ran1(), ran3(), rtflsp1(), rtflsp2(), rtnewt(), velequ()

There are remarks on top of each subroutine, please read corresponding chapter in Numerical Recipe book for details.

SPheRIO programming style and convention

The version number of SPheRIO is defined in its main program, eg. PARAMETER (version='Version 200806021200BR') Each subroutine starts with a brief log on its functions and updates, it looks like the following

CCC ##################...
CCC eosph.f controls everything for EoS in SPheRIO
CCC gf: 1- energy density, 2- entropy density
CCC eos(1)= energy density...
CCC eos(2)= entropy density...
CCC eos(3)= baryon density...
CCC eos(4)= pressure...
CCC eos(5)= temperature...
CCC eos(6)= baryon chemical potential.
CCC eos(7)= strangeness chemicalpotential –¿ to incorporate strangeness
CCC ##################...
CCC last updated 06122007
CCC incorporated flag "ieosmode"
CCC
CCC last updated 03052007
CCC incorporated some debugging statements
CCC
CCC last updated 09122006
CCC ##################...

It is noting that a separating line CCC ####... starts with CCC and three blanks and 65 successive #s