The JETSCAPE Framework

Tianyu Dai
Duke University

RHIC & AGS Annual User’s meeting
June 8th 2021
Outline

June 8th:
- Overview of JETSCAPE framework
- Run JETSCAPE using container
- Configure JETSCAPE
- Perform parton evolution in static medium
- Implement a new module in JETSCAPE*

June 9th:
- JETSCAPE for heavy ion collisions
- Configure JETSCAPE to use realistic hydrodynamic module
- Run p+p collisions
- Extend to A+A collisions and calculate RAA
Questions

Interact on Slack:

jetscape_tutorial_rhic_ags_2021

- Ask questions : # general
- Questions about installation:
  # installation
- School material: # school_material

Wiki page:

https://github.com/TianyuDai/JETSCAPE-rhic-ags/wiki
Questions

Interact on Slack:

jetscape_tutorial_rhic_ags_2021

- Ask questions: # general
- Questions about installation:
  # installation
- School material: # school_material

Wiki page:
https://github.com/TianyuDai/JETSCAPE-rhic-ags/wiki
Steps to run JETSCAPE

1. Install and build JETSCAPE using container: Singularity / Docker

   One line of code to create environment with all software pre-requisites

   - **Run only in Linux**
     - Not require root access

   **https://github.com/TianyuDai/JETSCAPE-rhic-aggs/wiki/Installing-JETSCAPE**

   ```
   singularity build --sandbox jetbox docker://jetescape/base:v1.4
   
   Shell into the newly built container.
   
   singularity shell --writable jetbox
   ````

   This last command essentially created a virtual machine where all the libraries required are available to compile and run JETSCAPE.

   - **Run in Linux, Windows, Mac**
     - Require root access

   **https://github.com/JETSCAPE/JETSCAPE/tree/master/docker**

   **Using JETSCAPE via Docker**

   Docker is a software tool that allows one to deploy an application in a portable environment. A docker “image” can be created for the application, allowing any user to run a docker “container” from this image. We have prepared a docker image for the JETSCAPE environment, which allows you to use JETSCAPE on macOS or linux without installing a long list of pre-reqs or worrying about interference with software you already have installed. Step-by-step instructions are provided below.

   Create and start a docker container that contains all of the JETSCAPE pre-reqs:

   ```
   macOS:
   
   docker run -it --rm -v ~/jetscape-docker:/home/jetscape-user --name myJetscape jetscape/base:v1
   
   linux:
   
   docker run -it --rm -v ~/jetscape-docker:/home/jetscape-user --name myJetscape --user $(id -u):$(id -g)
   ```
Steps to run JETSCAPE

Load Singularity/Docker environment

Inside container:
Building and running JETSCAPE

Outside container:
Editing text files

Instructions for this step: Find in Github wiki page!
https://github.com/TianyuDai/JETSCAPE-rhic-aggs/wiki/Installing-JETSCAPE

Tip:
Keep two terminals open: One inside the container, and one outside

When building/running JETSCAPE — inside container
When editing text files — outside container
Steps to run JETSCAPE

2. Download and build JETSCAPE
   - Create a directory called "rhic_ags_school" and CD into it. We assume /rhic-ags-school as the working directory during the school.
     mkdir /rhic-ags-school
     cd /rhic-ags-school
   - Download JETSCAPE
     git clone https://github.com/TianyuDai/JETSCAPE-rhic-ags.git
   - Enter into the cloned repository. Get external packages
     cd JETSCAPE-rhic-ags/external_packages
     bash get_music.sh
   - Create a directory called build. Enter into the build directory and build JETSCAPE.
     mkdir /rhic-ags-school/ JETSCAPE-rhic-ags/build
     cd ../build
     cmake –DUSE_MUSIC=ON ..
     make –j4
Configuring JETSCAPE

JETSCAPE is configured via two XML files

- **Master XML file** — *you don’t modify this*
  Contains default values for every possible module and parameter

- **User XML file** — *you provide this*
  List of which modules to run, and which default parameter values to override
Configuring JETSCAPE

Master XML file

you don’t modify this

A “database” of all possible modules and parameters

All possible basic settings

All possible initial state module parameters

...
Configuring JETSCAPE

User XML file
you provide this

Specify which modules to run
Specify parameter values (otherwise taken from master)

Activate modules (in order)

Set nEvents

Set initial condition

Set medium

Set energy loss module

Set Writer

config/jetscape_user_brick.xml
Running JETSCAPE

There is one central executable: runJetscape.cc

- Modules are automatically added according to User XML
- You don’t ever need to re-compile this executable

Pass your user configuration file as a command line argument:

cd /rhic-ags-school/JETSCAPE-rhic-ags/build
./runJetscape ../config/jetscape_user_brick.xml
JETSCAPE Output

JETSCAPE output contains:
- Final state hadrons
- Final state partons
- Full parton-shower history

You can produce JETSCAPE output in two formats:

Ascii

Custom JETSCAPE format
Executables are provided to extract final-state hadrons/partons
Can also directly write gzipped ascii

HepMC3

Standard event format (larger size)
Compatible with Rivet

Ascii output: build/brick_out.dat

```
2 Event
# sigmaGen 1
# sigmaErr 0
# weight 1
# HardProcess Parton List: PGun
0 21 0 120 0 0 120 0 0 0 0
# Energy loss Shower Initiating Parton: JetEnergyLoss
0 21 0 120 0 0 120 0 0 0 0
# Parton Shower in JetScape format to be used later by GTL graph:
[0] V 0 0 0 0
[1] V 0 0 0 0
[2] V 0 0 0 0.7
[3] V 0 0 0 0.7
[0]=>[1] P 0 21 0 120 0 0 120 0 0 0 0
[1]=>[2] P 3 21 0 118.743 0 0 118.743 0.71 0 0 0.71
[1]=>[3] P 4 21 0 1.27016 0 0 1.27016 0.71 0 0 0.71
```
Generate final state partons using Ascii output:

```
cd /rhic-ags-school/JETSCAPE-rhic-ags/build
mkdir .././JETSCAPE-output
./FinalStatePartons brick_out.dat .././JETSCAPE-output/my_final_state_partons.txt
```

JETSCAPE-rhic-ags/examples/FinalStatePartons.cc has two input parameters:

- `brick_out.dat` is the filename of the Ascii output file
- `my_final_state_partons.txt` is the file to save the final state parton information

JETSCAPE Output

Generate final state partons using Ascii output:

```bash
cd /rhic-ags-school/JETSCAPE-rhic-ags/build
makedir ../../../JETSCAPE-output
./FinalStatePartons brick_out.dat ../../../JETSCAPE-output/my_final_state_partons.txt
```

**JETSCAPE-rhic-ags/examples/FinalStatePartons** has two input parameters:

- `brick_out.dat` is the filename of the Ascii output file
- `my_final_state_partons.txt` is the file to save the final state parton information

Tutorial for this practice: Find in Github wiki page!  
Calculating parton energy distribution

Run the python script to plot the final parton energy distribution:

```
run python3 plot_energy_distribution.py
```

- **Initial energy distribution:**
a Dirac-delta function at initial energy

- **Final energy distribution:**
a distribution with a wider range

Tutorial for this practice:
Find in Github wiki page!
Calculating parton energy distribution

Run the python script to plot the final parton energy distribution:

```
cd /rhic-ags-school/JETSCAPE-rhic-ags/analysis/
python3 plot_energy_distribution
```

Tutorial for this practice:
Find in Github wiki page!
Implement a new module

```cpp
#include "JetEnergyLossModule.h"

using namespace Jetscape;

class MyJEL : public JetEnergyLossModule<MyJEL>
{
    public:
        MyJEL();
        virtual ~MyJEL();

        void Init();
        void DoEnergyLoss(double deltaT, double time, double Q2,
                            vector<Parton>& pIn, vector<Parton>& pOut);
        void WriteTask(weak_ptr<JetScapeWriter> w);

    private:
        // Allows the registration of the module so that it is available
        // to be used by the Jetscape framework.
        static RegisterJetScapeModule<MyJEL> reg;
};

#endif // MyJEL
```

Implement your physics in these standard functions, which will be called by the framework.

Note: Which function(s) to implement depends on the type of the module. For details, see arXiv:1903.07706.
Implement a new module

To register a module with the framework, we should do with the following steps:

- Note the following in your `CustomModule.h`
  ```cpp
  private:
    // Allows the registration of the module so that it is available to be used by the Jetscape framework
    static RegisterJetScapeModule<CustomModule> reg;
  ```

- Note the following in your `CustomModule.cc`
  ```cpp
  // Register the module with the base class
  RegisterJetScapeModule<CustomModule> CustomModule::reg("CustomModule");
  ```

- Note this part in `src/framework/JetScape.cc`
  ```cpp
  if ((int)childElementName.find("CustomModule") == 0) {
    auto customModule = JetScapeModuleFactory::createInstance(childElementName);
    if (customModule) {
      loss->Add(customModule);
      JSINFO << "JetScape::DetermineTaskList() -- Eloss: Added "
         << childElementName << " to Eloss list."
      ;
    }
  }
  ```

Set your new module’s name as CustomModule, or modify `/src/framework/JetScape.cc`
Implement a new module

Practice: implement a diffusion process in the energy loss module

Interactions between the parton and the medium can be described using a Boltzmann equation

\[
\left( \frac{\partial}{\partial t} + \frac{p}{p} \cdot \nabla \right) \delta f_a = -C[\delta f_a, n_a]
\]

Soft and frequent interactions can be treated stochastically as a diffusion process

\[
C_a^{\text{diff}}[\delta f] \equiv -\frac{\partial}{\partial p^i} \left[ \eta_0(p) p^i \delta f_a(p) \right] - \frac{1}{2} \frac{\partial^2}{\partial p^i \partial p^j} \left[ \left( \hat{p}^i \hat{p}^j \hat{q}_L(p) + \frac{1}{2} (\delta^{ij} - \hat{p}^i \hat{p}^j) \hat{q}(p) \right) \delta f_a(p) \right].
\]

Put a new module performing diffusion process in JETSCAPE

Implement a new module

Practice: implement a diffusion process in the energy loss module

Use analysis/new_modules/CustomModule.cc to perform diffusion process

Initialization:

```cpp
void CustomModule::Init()
{
    JSINFO<<"Initialize CustomModule ...";
    double deltaT = 0.8;
    deltaT = GetXMLElementDouble("Eloss", "deltaT");
    string s = GetXMLElementText("Eloss", "CustomModule", "name");
    JSDEBUG << s << " to be initialized ...";
    alpha_s = 0.3;
    alpha_s = GetXMLElementDouble("Eloss", "CustomModule", "alpha_s");
    pcut = 1.;
    pcut = GetXMLElementDouble("Eloss", "CustomModule", "pcut");
    mu = 1.;
    mu = GetXMLElementDouble("Eloss", "CustomModule", "cutoff");
    g = sqrt(4.*M_PI*alpha_s);
    std::normal_distribution<Double> white_noise(0., 1.);
}
```

Read information from hydro module:

```cpp
std::unique_ptr<FluidCellInfo> check_fluid_info_ptr;
GetHydroCellSignal(Time, xx, yy, zz, check_fluid_info_ptr);
VERBOSE(5)< MAGENTA<< Temperature from Brick (Signal) = "
<<check_fluid_info_ptr->temperature;
```

Add particles after energy loss back to parton list:

```cpp
if (pVecNew.t() > pcut)
{
    pOut.push_back(Parton(0, Id, 0, pVecNew, xVec));
    pOut[pOut.size()-1].set_form_time(0.);
}
return;
```

Do energy loss:

```cpp
void CustomModule::DoEnergyLoss(double deltaT, double Time, double Q2, vector<Parton>& pIn, vector<Parton>& pOut)
```
Implement a new module

Practice: implement a diffusion process in the energy loss module

- Find `CustomModule.cc`, `CustomModule.h`, `Lorenz.h` in `analysis/new_modules`, and copy them to `src/jet/
  ```
  cp /rhic-ags-school/JETSCAPE-rhic-ags/analysis/* ../src/jet/
  ```

- Re-compile JETSCAPE
  ```
  cd /rhic-ags-school/JETSCAPE-rhic-ags/build
  cmake -DUSE_MUSIC=ON ..
  make -j
  ```

- Create a new XML file, `config/jetscape_user_custom.xml`, add the new module into it

- Run JETSCAPE with the new energy loss module
  ```
  ./runJetscape ../config/jetscape_user_custom.xml
  ```

- Run `FinalStatePartons` to process Ascii output file, and plot the final parton energy distribution
  ```
  ./FinalStatePartons custom_brick_out.dat ../JETSCAPE-output/custom_final_state_partons.txt
  cd ../analysis
  python3 plot_energy_distribution.py
  ```

Remember to modify the output file name

Tutorial for this practice: Find in Github wiki page!
Implement a new module

Practice: implement a diffusion process in the energy loss module

Energy distribution of final partons using the new added custom energy loss module

Tutorial for this practice:
Find in Github wiki page!