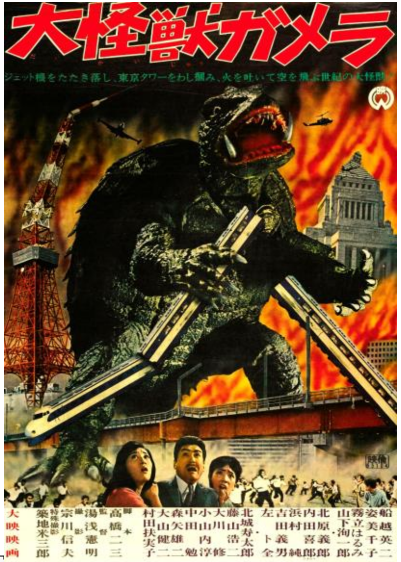
The GAMERA toolkit

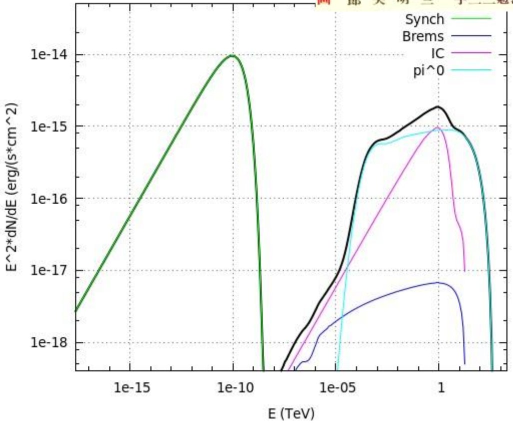
More than a kaiju

Carlo Romoli (MPI-K)

What is GAMERA?

- A Japanese Kaiju
- A toolkit to model evolution of relativistic particles and their emission in various astrophysical scenarios





Original creator and developer **Dr. Joachim Hahn**

[Hahn J., ICRC2015]

Where to find GAMERA?

- New Github repository
  - Previous one in Joachim's personal space
  - Moved to something more maintainable after Joachim left science
  - New organization to host the repository:
    - libgamera (documentation being moved to other location)
    - For more info you can ask Mischa or me (with the task of maintaining this nice code)

How is GAMERA structured?

- GAMERA is a set of C++ libraries wrapped in python using Swig
  - Efficient computing underneath
  - Nice user friendliness in the usage

What does the **PARTICLES** class do?

- Solves the time evolution of a generic distribution of particles (electrons or protons)

Cooling term: (dE/dt)

Source term

$$\frac{\partial N}{\partial t} = \frac{\partial}{\partial \gamma}(PN) - \frac{N}{\tau} + Q$$

Time evolution of differential particle count at energy E

Particle energy

Particle escape term

How does it do it?

- 2 cases:
  - General with all possible time dependencies
  - Time independent losses without escape of particles

Fully numerical solution donor-cell advection algorithm

$$q_i^{n+1} = q_i^n + \frac{\Delta t}{\Delta x}(f_{i-1/2}^{n+1/2} - f_{i+1/2}^{n+1/2})$$

Semi-analytical solution from Atoyan&Aharonian, 1999

- with constant losses (method 1)
- without losses (method 2)

Losses

Electrons: synchrotron, inverse Compton scattering (in different flavours), bremsstrahlung, adiabatic losses

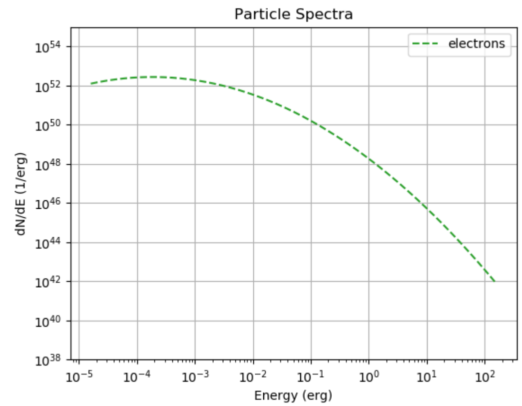
Protons: escape only

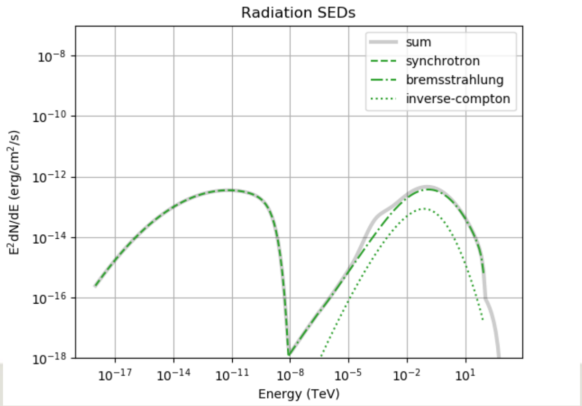
What does the **RADIATION** class do?

- Solves the radiation emission of a given particle distribution for different mechanisms
- Several emission mechanisms implemented for both protons and leptons
  - Radiation output computed even though the PARTICLES class does not take the process into account in the evolution

Radiation from electrons

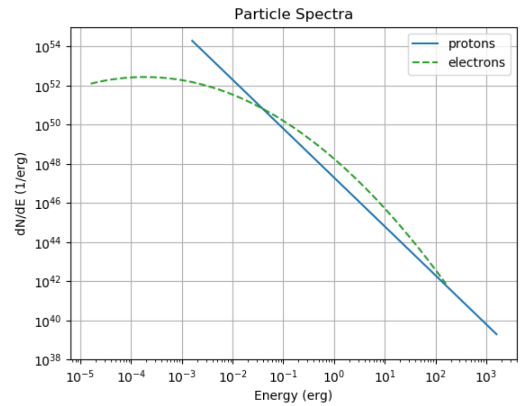
- *Synchrotron* -> Computed when we add a B field (Blumenthal&Gould1970, Ghisellini1988)
- *Inverse Compton scattering* -> Computed in presence of a target photon field
  - Isotropic (Blumenthal&Gould1970)
  - Anisotropic case (Moskalenko&Strong2000, Aharonian&Atoyan1981)
- *Bremsstrahlung* -> As soon as we add an ambient density (Baring1999)

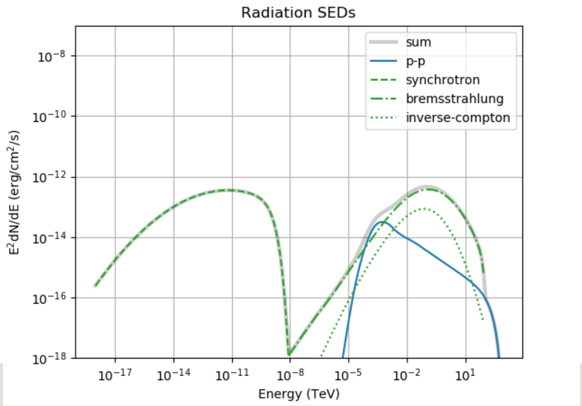




Radiation from protons

- gamma ray emission from *pp* interactions due to neutral pion decay
  - using analytical parametrization developed in Kafexhiu2014





What does the **ASTRO** class do?

- Holds astrophysical models for source modelling and population syntheses
  - Galactic structures
  - Magnetic field models
  - VHE-source progenitor model functions
  - VHE-source dynamical models (for SNRs)

What does the **UTILS** class do?

- Miscellanea of useful functions and values
  - Constants, distributions of random numbers, integrations, interpolations (2d_interp dependency)..
- Use of the 'gnu scientific library' (*gsl*) for the calculations
  - More efficient implementation

Some more technical details…

- Interface between GAMERA and Python (or I wouldn't be at a PyGAMMA workshop)
- Interface done with SWIG
  - Good interaction between the lists and *numpy* arrays and the 1D-2D vectors used in the C++ code

.i file for the swig interface

```
%module gappa
%{
#include "../include/Radiation.h"
#include "../include/Particles.h"
#include "../include/Utils.h"
#include "../include/Astro.h"
#include "../include/2D_interp/interp2d_spline.h"
#include "../include/2D_interp/interp2d.h"
%}

%include "typemaps.i"
%include "std_vector.i"
%include "std_string.i"
%include "std_iostream.i"

namespace std
{
%template(OneDVector) vector<double>;
%template(TwoDVector) vector< vector<double> >;
}

%include "../include/Radiation.h"
%include "../include/Particles.h"
%include "../include/Utils.h"
%include "../include/Astro.h"
%include "../include/2D_interp/interp2d_spline.h"
%include "../include/2D_interp/interp2d.h"
```

Python interface

Import the *gappa* module in a python script

```
import sys
sys.path.append(<gamera>/lib/)
import gappa as gp
```

Classes initializated in standard way:

```
fr = gp.Radiation()
fp = gp.Particles()
```

Initial particle distribution must be an array of tuple **(E,dN/dE)** passed as np.array

```
power_law =
  np.array(zip(ene,dnde_values))
```

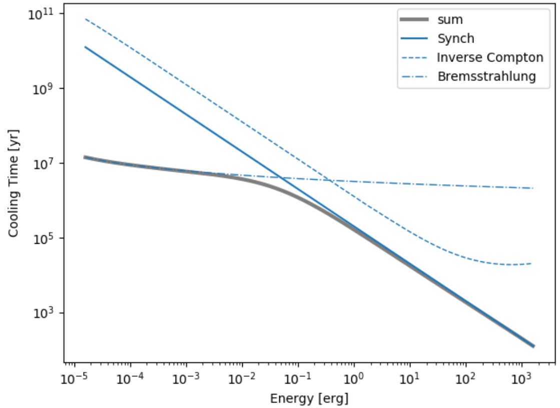
Pass the particle distribution to the class

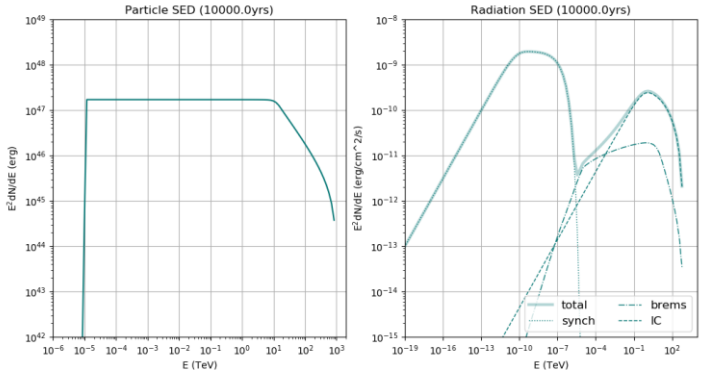
```
fr.SetElectrons(power_law)
```

…and practical examples

- Particles class

```
fp = gp.Particles()

[...]

fp.SetBField(b_field)
fp.SetAmbientDensity(density)
fp.SetRadius(radius)
fp.AddThermalTargetPhotons(t_cmb,edens_cmb,bins)
fp.SetCustomInjectionSpectrum(power_law)

tcool_tot =
    np.array(fp.GetCoolingTimeScale(energy_in_erg_pl))

fp.SetAge(age)

fp.CalculateElectronSpectrum()
sp = np.array(fp.GetParticleSpectrum())
sed = np.array(fp.GetParticleSED())
```

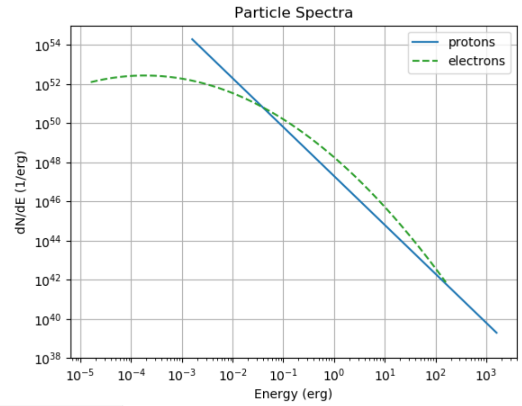


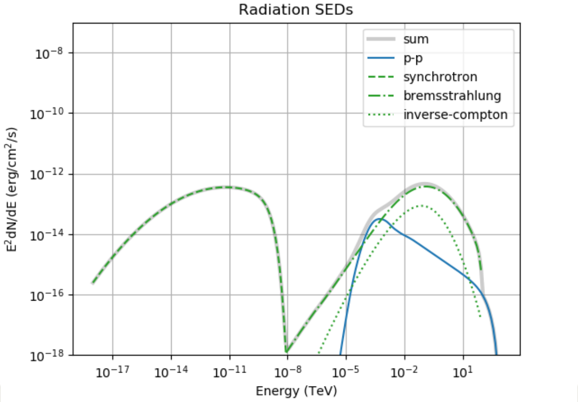


…and practical examples (pt. 2)

- Radiation class

```
fr = gp.Radiation()

[...]

fr.SetAmbientDensity(ambient_density)
fr.SetBField(b_field)
fr.AddThermalTargetPhotons(t_1,edens_1)
fr.SetDistance(distance)

fr.SetElectrons(elLogPsp)

e = np.logspace(-6,15,200) * gp.eV_to_erg
fr.CalculateDifferentialPhotonSpectrum(e)

total_sed = np.array(fr.GetTotalSED())
```





Development plans

- Some new implementations in the code
  - New version of anisotropic inverse Compton (by Mischa, added last week in the repository...documentation will come soon)
  - Implementation of $\gamma\gamma$ absorption (Carlo, will be added very soon)
  - Fixing some old pull requests... ionization losses of protons (...)
- Of course...bug fixing and improvement of the documentation
  - Beside descriptive documentation and tutorials also have a more detailed doxygen documentation of the various functions and functionalities

Conclusions

- GAMERA is a powerful tool for modelling of relativistic particles in astrophysical environments
- Aim to keep it alive and maintained
- Use it
  - libgamera

...one last thing...

- There is another code that is somehow similar to GAMERA (and some of you might know it)

NAIMA

– You can find it here

Developed by **Dr. Victor Zabalza**
Zabalza V., ICRC2015

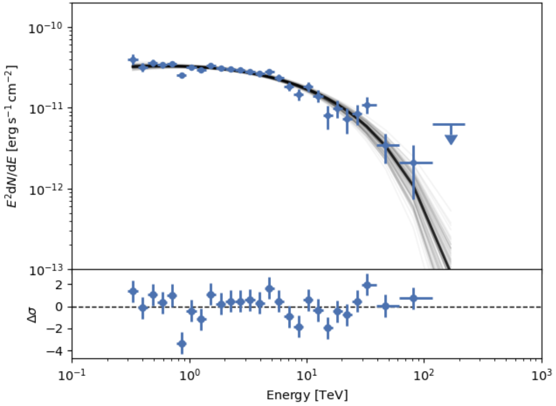
naima
Python package for computation of non-thermal radiation from relativistic particle populations and MCMC fitting to observed spectra

One slide (or 2) about NAIMA

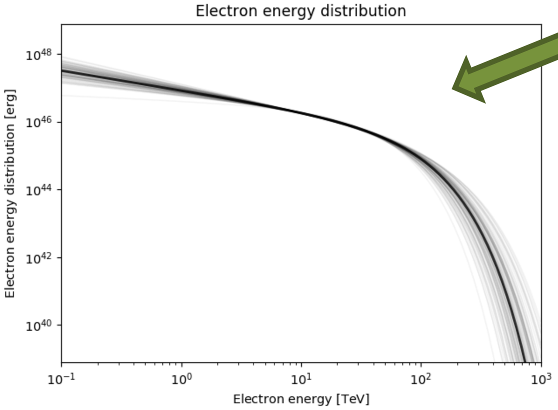
- Fully python based
  - you can install it with pip or conda
  - heavy use of analytical approximations to speed up
  - use of astropy tools (e.g. for quantities and units)
- Fit of spectral data points
  - Uses MCMC approach through the package `emcee`
- Lacks particle evolution features (snapshot spectra, no particle cooling)
- A bit more rigid with the available particle distributions

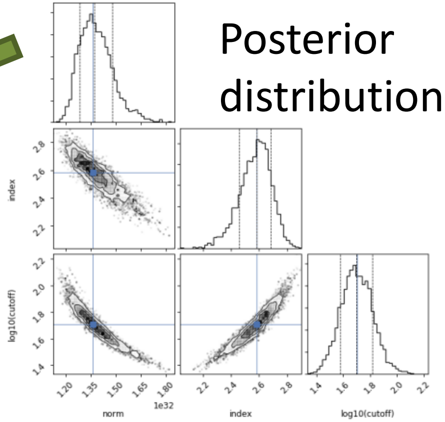
so..2 slides

- The nicest feature in NAIMA is the fitting of data
- Use of a Bayesian approach to retrieve the original parent population



MCMC fitting
Posterior distributions



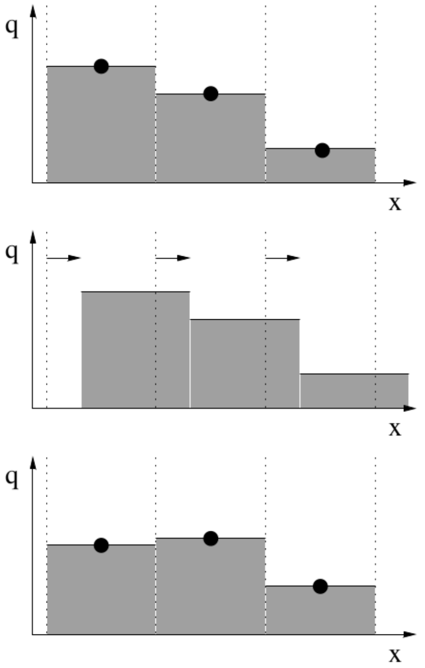


Thank you for your attention!

backup slides

In case of emergency

Donor-cell algorithm



t

t+dt

Here:
- q = N
- x = Energy

Flux = flow speed x fluid density

Here:
- Flow speed = cooling rate
- Fluid density = particle density

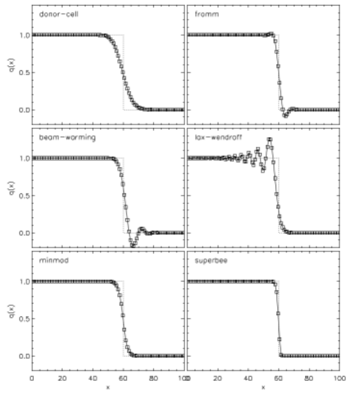
$$q_i^{n+1} = q_i^n + \frac{\Delta t}{\Delta x}\left(f_{i-1/2}^{n+1/2} - f_{i+1/2}^{n+1/2}\right)$$

time

energy

Flux (u*q) into cell i

Flux out of cell i



Discontinuity treated with slope delimiters to avoid numerical oscillations

From J.Hahn slides