The IDEFIX radiative transfer module

Nicolas Scepi
IPAG, ERC MHDiscs

European Research Council
Established by the European Commission

19th of September 2024

MHDiscs

Outline

- Introduction on radiative transfer
- Radiative transfer module in IDEFIX and tests
- User guide
- Some technical details

What is radiative transfer?

Study of the propagation of light and its interaction with matter

Three kind of interactions:

- absorption
- emission
- scattering

Why do we need radiative transfer?

1. Compute observational signature of an object

- Light curve
- Spectrum
- Polarisation

Why do we need radiative transfer?

1. Compute observational signature of an object

   - **Light curve**
   - Spectrum
   - Polarisation

Why do we need radiative transfer?

1. Compute observational signature of an object

- **Light curve**
- Spectrum
- Polarisation

2. Compute the dynamics when radiative effects are dominant

- High luminosity objects
- Irradiated disks
- Objects with lots of absorption lines
- “High-energy” plasmas (pair-creation basically)

Why do we need radiative transfer?

1. Compute observational signature of an object

    - **Light curve**
    - Spectrum
    - Polarisation

2. Compute the dynamics when radiative effects are dominant

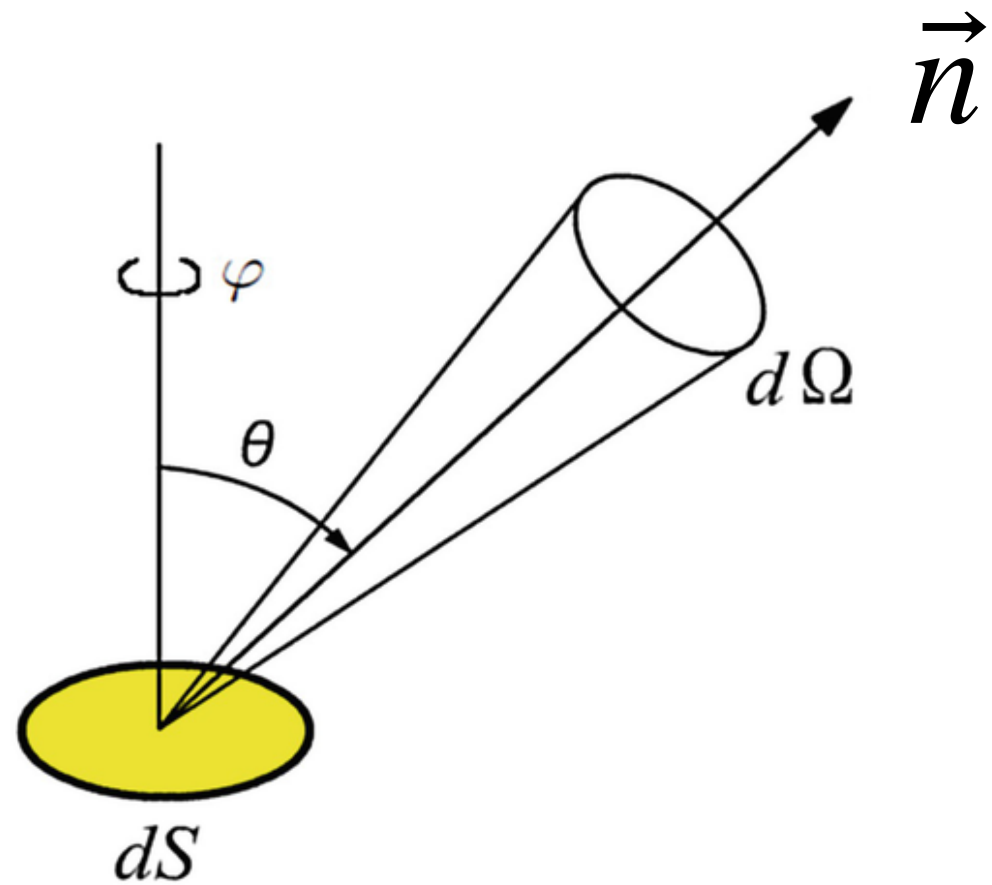
    - **High luminosity objects**
    - **Irradiated disks**
    - Objects with lots of absorption lines
    - "High-energy" plasmas (pair-creation basically)

How do we do radiative transfer?

We solve the radiative transfer equation!

$$\frac{1}{c}\frac{\partial}{\partial t}I_\nu + \vec{n}\cdot\vec{\nabla}I_\nu = \eta_\nu - (\kappa_\nu + \sigma_\nu)I_\nu$$

- I_ν is the specific intensity
- η_ν is the emissivity
- κ_ν is the absorption opacity
- σ_ν is the scattering opacity



How do we do radiative transfer?

We solve the radiative transfer equation!

$$\frac{1}{c}\frac{\partial}{\partial t}I_\nu + \vec{n}\cdot\overrightarrow{\nabla}I_\nu = \eta_\nu - (\kappa_\nu + \sigma_\nu)I_\nu$$

Basically means “light get transported along $\vec{n}$ with absorption/emission/scattering events along the way”

So why is radiative transfer difficult?

Why is radiative transfer difficult?

- **Depends on** $(t, \vec{x}, \vec{n}, \nu)$
- **Non-local effects (for example scattering)!**
- **Time scale separation between hydro and radiation**

Fluid approximation

To simplify the problem, we get rid of the directionality of radiation

Define the moments of the radiation distribution and solve for “fluid” equations

Fluid approximation

To simplify the problem, we get rid of the directionality of radiation

Define the moments of the radiation distribution and solve for "fluid" equations

0th order: $E_{r,\nu} = \frac{1}{c}\int_{\Omega} I_\nu d\Omega$ is the radiative energy density

1st order: $F^i_{r,\nu} = \int_{\Omega} I_\nu n^i d\Omega$ is the radiation energy flux vector

2nd order: $P^{ij}_{r,\nu} = \int_{\Omega} I_\nu n^i n^j d\Omega$ is the radiation energy pressure tensor

Fluid approximation

Conservation of radiation energy density:

$$\frac{1}{c}\frac{\partial E_r}{\partial t} + \overrightarrow{\nabla} \cdot \overrightarrow{F}_r = G^0$$

Conservation of radiation energy flux:

$$\frac{1}{c}\frac{\partial \overrightarrow{F}_r}{\partial t} + \overrightarrow{\nabla} \cdot \bar{P}_r = \overrightarrow{G}$$

$$\text{where } G^\mu = \int_\Omega ((\kappa_\nu + \sigma_\nu)I_\nu - \eta_\nu)n^\mu d\Omega$$

Equation of moment of order *m* requires information on moment *m+1*

Need a closure somewhere!

Diffusion approximation

Simplest case is to say that $\overrightarrow{F}_r \propto \overrightarrow{\nabla} E_r$

$$\frac{1}{c}\frac{\partial E_r}{\partial t} + \overrightarrow{\nabla} \cdot \left(\frac{c\lambda}{\kappa\rho}\overrightarrow{\nabla} E_r\right) = G^0$$

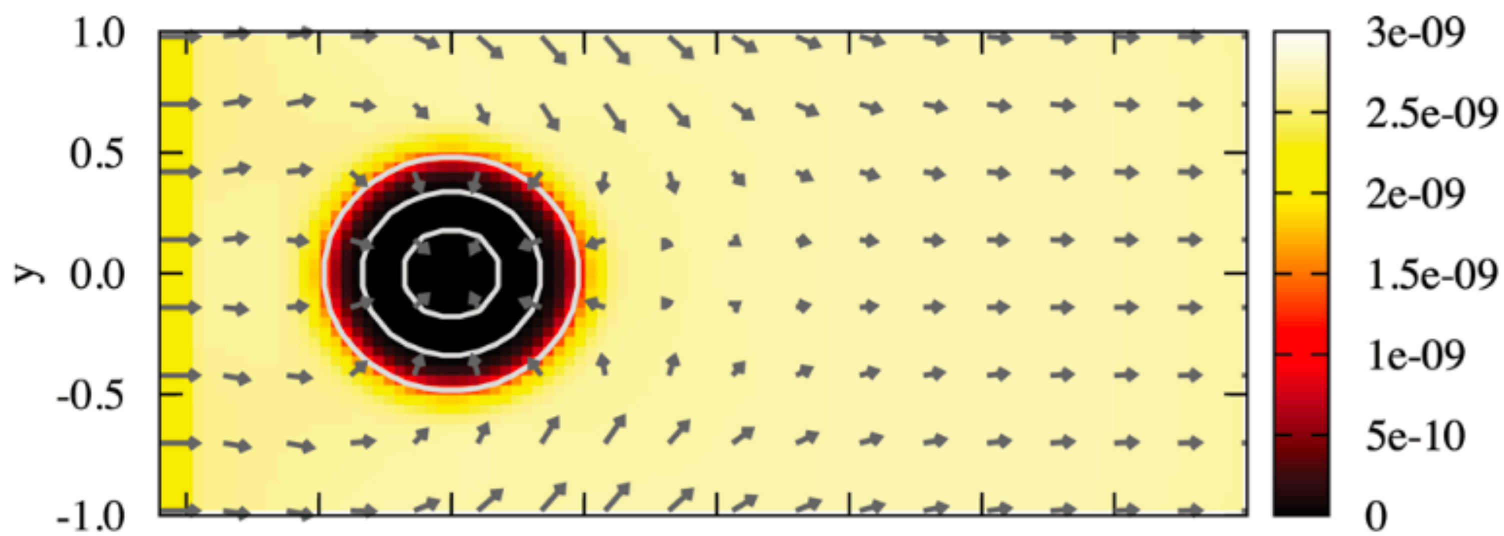
This is called the diffusion approximation!

Diffusion approximation

Simplest case is to say that $\vec{F}_r \propto \vec{\nabla} E_r$

$$\frac{1}{c}\frac{\partial E_r}{\partial t} + \vec{\nabla} \cdot \left(\frac{c\lambda}{\kappa\rho}\vec{\nabla} E_r\right) = G^0$$

This is called the diffusion approximation!



Sadowski et al. 2013

Cannot handle shadows, bad in optically thin regions

Not suited for irradiated disks or radiation-driven outflows…

M1 approximation

Assume that $\bar{P} = f(E_r, \overrightarrow{F}_r)$ for the closure

$$\frac{1}{c}\frac{\partial E_r}{\partial t} + \overrightarrow{\nabla} \cdot \overrightarrow{F}_r = G^0$$

$$\frac{1}{c}\frac{\partial \overrightarrow{F}_r}{\partial t} + \overrightarrow{\nabla} \cdot \bar{P}_r = \overrightarrow{G}$$

$f(E_r, \overrightarrow{F}_r)$ is chosen to recover free-streaming and diffusion limit but might fail in between

Radiative transfer module in IDEFIX and tests

M1 approximation

Assume that $\bar{P} = f(E_r, \overrightarrow{F}_r)$ for the closure

$$\frac{1}{c}\frac{\partial E_r}{\partial t} + \overrightarrow{\nabla} \cdot \overrightarrow{F}_r = G^0$$

$$\frac{1}{c}\frac{\partial \overrightarrow{F}_r}{\partial t} + \overrightarrow{\nabla} \cdot \bar{P}_r = \overrightarrow{G}$$

$f(E_r, \overrightarrow{F}_r)$ is chosen to recover free-streaming and diffusion limit but might fail in between

We follow *Melon Fuksman et al. (2019, 2021)* and split the solver in **explicit** and **implicit** part

Radiation transport

Explicit step:

$$\frac{1}{c}\frac{\partial E_r}{\partial t} + \overrightarrow{\nabla} \cdot \overrightarrow{F}_r = 0$$

$$\frac{1}{c}\frac{\partial \overrightarrow{F}_r}{\partial t} + \overrightarrow{\nabla} \cdot \bar{P}_r = 0$$

Solve Riemann problem for radiation

Solver can be HLL or HLLC

Radiation transport

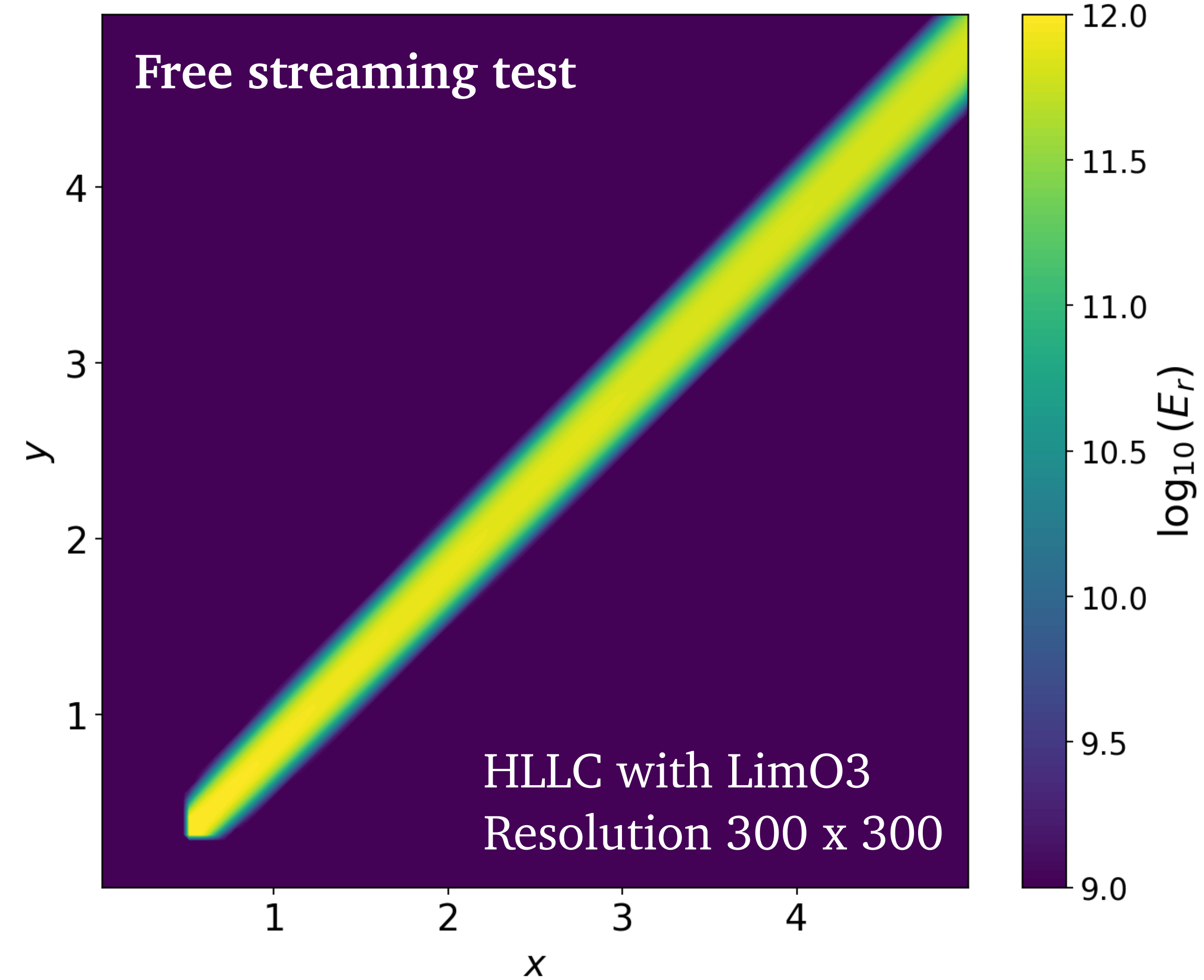
Explicit step:

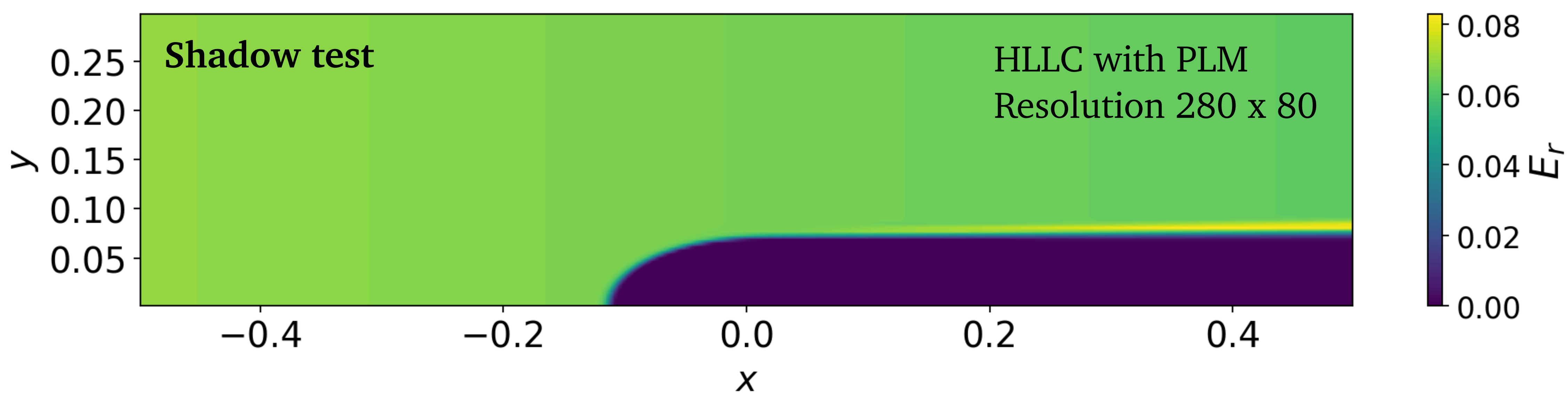
$$\frac{1}{c}\frac{\partial E_r}{\partial t} + \vec{\nabla}\cdot\vec{F}_r = 0$$

$$\frac{1}{c}\frac{\partial \vec{F}_r}{\partial t} + \vec{\nabla}\cdot\bar{P}_r = 0$$

Solve Riemann problem for radiation

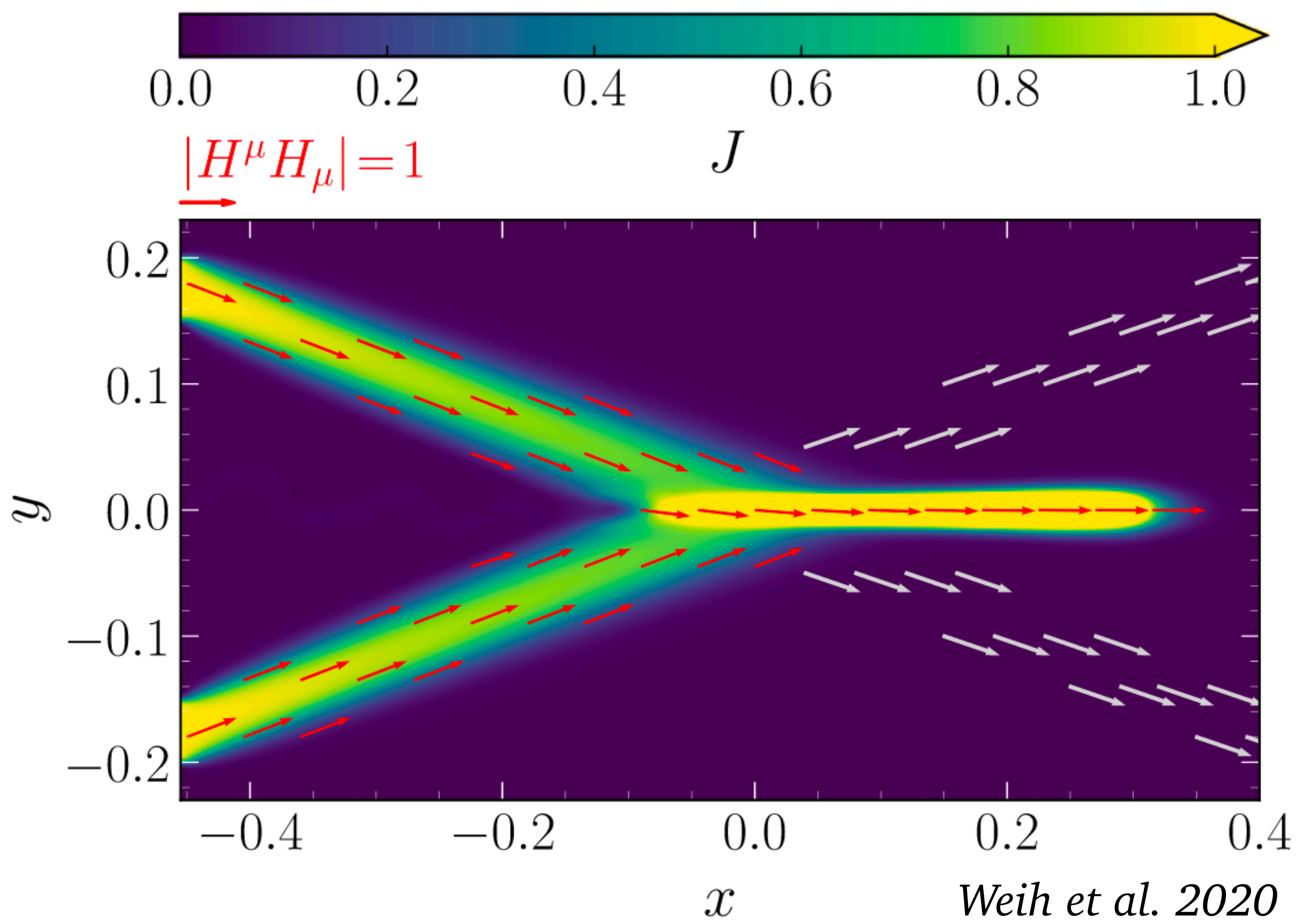
Solver can be HLL or HLLC





Crossing beams

M1 cannot handle crossing of light rays
(more generally multiple sources)



Weih et al. 2020

Source terms

Semi-implicit step:

We use implicit, iterative method called the fixed-point method

$$\frac{1}{c}\frac{\partial E_r}{\partial t} = G^0$$

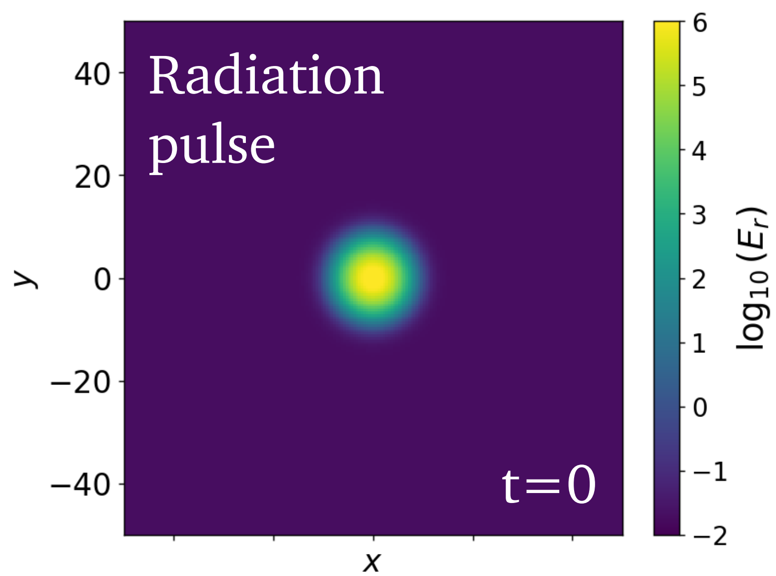
$$\frac{1}{c}\frac{\partial \overrightarrow{F}_r}{\partial t} = \overrightarrow{G}$$

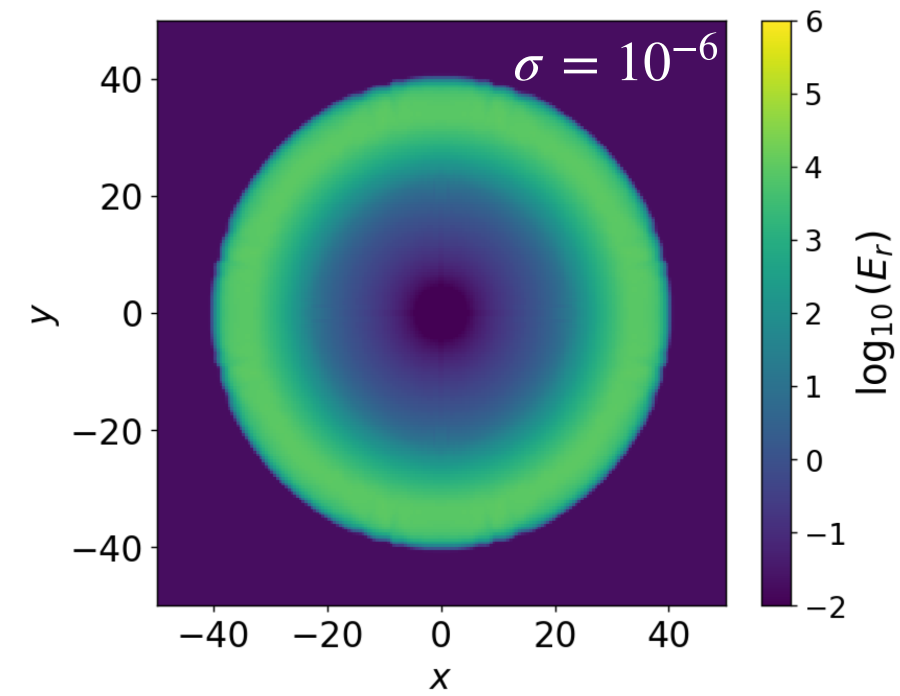
In the limit $v/c \rightarrow 0$

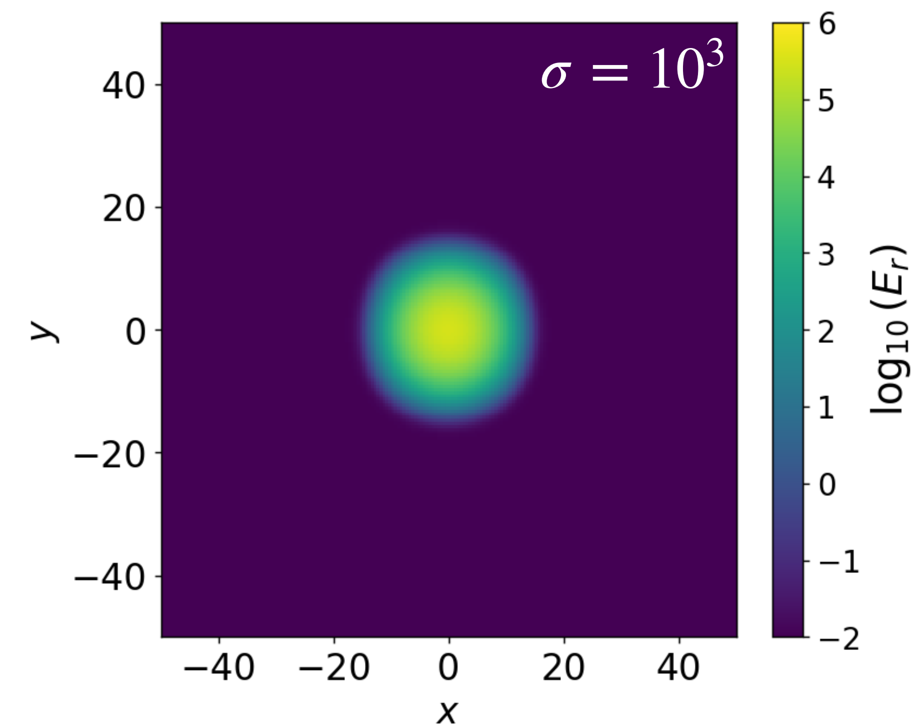
$$G^0 = \rho\kappa(E_r - a_R T^4)$$

$$\overrightarrow{G} = \rho(\kappa + \sigma)\overrightarrow{F}_r$$

Opacities can be constant, Kramers-type or imported from table







Radiation and MHD together

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \boldsymbol{v}) = 0$$

$$\frac{\partial (\rho \boldsymbol{v})}{\partial t} + \nabla \cdot (\rho \boldsymbol{v}\boldsymbol{v}) + \nabla p_g = \boldsymbol{G} + \boldsymbol{S_m} - \rho \nabla \Phi$$

$$\frac{\partial (E + \rho \Phi)}{\partial t} + \nabla \cdot [(E + p_g + \rho \Phi)\boldsymbol{v}] = c\, G^0 + S_{\mathrm{E}} - \nabla \cdot \boldsymbol{F}_{\mathrm{Irr}}$$

$$\frac{1}{\hat{c}} \frac{\partial E_r}{\partial t} + \nabla \cdot \boldsymbol{F}_r = -G^0$$

$$\frac{1}{\hat{c}} \frac{\partial \boldsymbol{F}_r}{\partial t} + \nabla \cdot \mathbb{P}_r = -\boldsymbol{G},$$

Radiation and MHD together

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \boldsymbol{v}) = 0$$

$$\frac{\partial (\rho \boldsymbol{v})}{\partial t} + \nabla \cdot (\rho \boldsymbol{v}\boldsymbol{v}) + \nabla p_g = \boldsymbol{G} + \boldsymbol{S_m} - \rho \nabla \Phi$$

$$\frac{\partial (E + \rho \Phi)}{\partial t} + \nabla \cdot [(E + p_g + \rho \Phi)\boldsymbol{v}] = c\, G^0 + S_{\mathrm{E}} - \nabla \cdot \boldsymbol{F}_{\mathrm{Irr}}$$

$$\frac{1}{\hat{c}} \frac{\partial E_r}{\partial t} + \nabla \cdot \boldsymbol{F}_r = -G^0$$

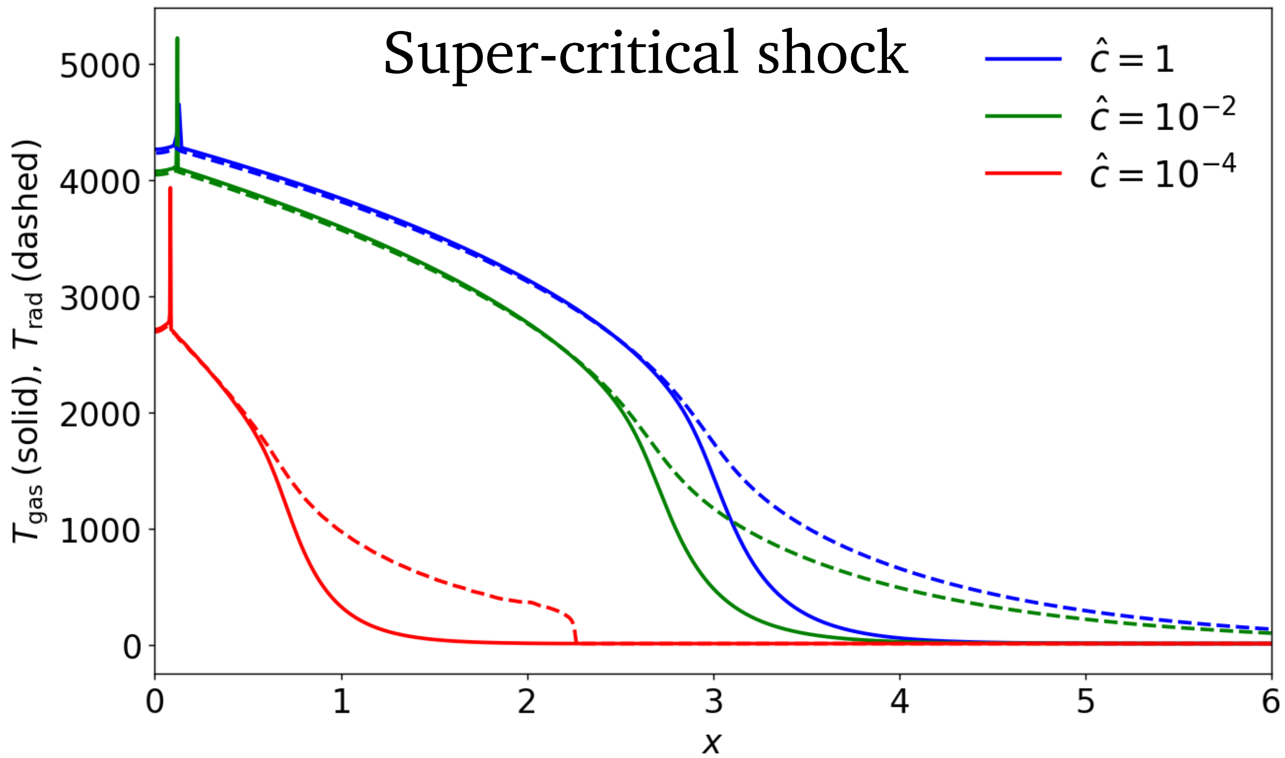
$$\frac{1}{\hat{c}} \frac{\partial \boldsymbol{F}_r}{\partial t} + \nabla \cdot \mathbb{P}_r = -\boldsymbol{G},$$

We use a reduced speed of light, $\hat{c}$, to have reasonable time steps

Ok for steady-state solutions but produce artifacts in non-steady solutions!

Reduced speed of light

HLLC with PLM, resolution 256



To avoid artifacts, we need $\hat{c} \gg \max(v_{\rm dyn}, v_{\rm diff})$

In this problem, $v_{\rm dyn} \approx 500$

$\hat{c}$ needs to be chosen with care for the problem at hand!

User guide

Using the radiative module

Only need to update two files:

- idefix.ini to define input parameters
- setup.cpp to define initial conditions and user functions

idefix.ini

```
[Rad]
nFrequencies            1
solver_rad         hll_rad
reduced_c               1.
kappa   constant        0.1
xi      userdef         1

[Units]
velocity           1.49598e10
length             1.49598e13
density            1.e-30
```

→ *Number of frequency groups* (nFrequencies)

idefix.ini

```
[Rad]
nFrequencies           1
solver_rad        hll_rad
reduced_c              1.
kappa    constant      0.1
xi       userdef       1

[Units]
velocity          1.49598e10
length            1.49598e13
density           1.e-30
```

→ *Solver for radiation (HLL or HLLC)*

idefix.ini

```
[Rad]
nFrequencies            1
solver_rad         hll_rad
reduced_c               1.
kappa    constant       0.1
xi       userdef        1

[Units]
velocity           1.49598e10
length             1.49598e13
density            1.e-30
```

→ *Value of* $\hat{c}/c$

idefix.ini

```
[Rad]
nFrequencies            1
solver_rad          hll_rad
reduced_c               1.
kappa    constant       0.1
xi       userdef        1

[Units]
velocity            1.49598e10
length              1.49598e13
density             1.e-30
```

Set absorption and scattering opacities
Can be constant, Kramers or userdef

Dimension of imported table

idefix.ini

```
[Rad]
nFrequencies            1
solver_rad         hll_rad
reduced_c               1.
kappa   constant        0.1
xi      userdef         1

[Units]
velocity           1.49598e10
length             1.49598e13
density            1.e-30
```

→ *Set units in cgs*

setup.cpp

To initialize the radiation fluid variables

```
void Setup::InitFlow(DataBlock &data) {
    // Create a host copy
    DataBlockHost d(data);
    real csiso = csisoGlob;

    for(int k = 0; k < d.np_tot[KDIR] ; k++) {
        for(int j = 0; j < d.np_tot[JDIR] ; j++) {
            for(int i = 0; i < d.np_tot[IDIR] ; i++) {

              d.Vc(RHO,k,j,i) = 1.;
              d.Vc(VX1,k,j,i) = 0.;
              d.Vc(VX2,k,j,i) = 0.;
              d.Vc(PRS,k,j,i) = d.Vc(RHO,k,j,i)*csiso*csiso;

              d.RadVc[0](ER,k,j,i) = 1.e4;
              d.RadVc[0](FR1,k,j,i) = ZERO_F;
              d.RadVc[0](FR2,k,j,i) = ZERO_F;

            }
        }
    }

    // Send it all, if needed
    d.SyncToDevice();
}
```

→ *Hydro variables*

→ *Radiation variables*

setup.cpp

To enroll userdef functions
(boundary conditions, additional source term)

```
void UserdefBoundaryRad(Fluid<RadiationPhysics> *radiation, int dir, BoundarySide side, real t) {
  IdefixArray4D<real> Vc = radiation->Vc;
  auto *data = radiation->data;

  real C_c = idfx::units.c;
  real C_ar = idfx::units.ar;
  real T0 = T0Glob;

  if(dir==IDIR) {
    int ighost,ibeg,iend;
    if(side == left) {
      ibeg = 0;
      iend = data->beg[IDIR];
      idefix_for("UserDefBoundaryRad",
        0, data->np_tot[KDIR],
        0, data->np_tot[JDIR],
        ibeg, iend,
        KOKKOS_LAMBDA (int k, int j, int i) {
          Vc(ER,k,j,i) = C_ar*std::pow(T,4);
          Vc(FR1,k,j,i) = Vc(ER,k,j,i);
        });
    }
  }
}
```

```
Setup::Setup(Input &input, Grid &grid, DataBlock &data, Output &output)
{

  T0Glob = input.Get<real>("Rad","kappa",3);

  // Set the function for userdefboundary
  if(data.haveRadiation) {
    int nFrequencies = data.radiation.size();
    for(int n = 0 ; n < nFrequencies ; n++) {
      data.radiation[n]->EnrollUserDefBoundary(&UserdefBoundaryRad);
    }
  }
  data.hydro->EnrollUserDefBoundary(&UserdefBoundary);

}
```

Radiation data block is a vector
(for multi-fluid)

Some technical details

Radiation as a type of fluid

IDEFIX uses class templates that encapsulate arrays, fonctions, etc…

<u>Fluid class template</u>

```
template<typename Phys>
class Fluid {
 public:
  Fluid( Grid &, Input&, DataBlock *, int n = 0);
  void ConvertConsToPrim();
  void ConvertPrimToCons();
  template <int> void CalcParabolicFlux(const real);
  template <int> void CalcRightHandSide(real, real );

  // Our boundary conditions
  std::unique_ptr<Boundary<Phys>> boundary;

  // EOS
  std::unique_ptr<EquationOfState> eos;

  // Arrays required by the Hydro object
  IdefixArray4D<real> Vc;      // Main cell-centered primitive variables index
  IdefixArray4D<real> Uc;      // Main cell-centered conservative variables

  std::unique_ptr<RiemannSolver<Phys>> rSolver;
```

Radiation as a type of fluid

Each fluid has a type of physics
("Hydro" fluid, dust fluid, radiation fluid)

```
template<typename Phys>
class Fluid {
 public:
  Fluid( Grid &, Input&, DataBlock *, int n = 0);
  void ConvertConsToPrim();
  void ConvertPrimToCons();
  template <int> void CalcParabolicFlux(const real);
  template <int> void CalcRightHandSide(real, real );

  // Our boundary conditions
  std::unique_ptr<Boundary<Phys>> boundary;

  // EOS
  std::unique_ptr<EquationOfState> eos;

  // Arrays required by the Hydro object
  IdefixArray4D<real> Vc;      // Main cell-centered primitive v
  IdefixArray4D<real> Uc;      // Main cell-centered conservativ

  std::unique_ptr<RiemannSolver<Phys>> rSolver;
```

Physics class template

```
// Physics type
// The default Physics class
struct DefaultPhysics {
  static constexpr bool dust{false};
  #if HAVE_ENERGY == 1
    static constexpr bool pressure{true};
  #else
    static constexpr bool pressure{false};
  #endif
  static constexpr bool eos = pressure; // whether we have a eos
  #if MHD == YES
    static constexpr bool mhd{true};
    static constexpr int nvar{1+2*COMPONENTS  + (pressure?1:0)};
  #else
    static constexpr bool mhd{false};
    static constexpr int nvar{1+COMPONENTS  + (pressure?1:0)};
  #endif
  static constexpr bool radiation{false};
};

// Some Dust
struct DustPhysics {
  static constexpr bool dust{true};
  static constexpr bool pressure{false};
  static constexpr bool eos{false};

  static constexpr bool mhd{false};
  static constexpr int nvar{1+COMPONENTS};

  static constexpr bool radiation{false}; // No radiation for dust for now
};

// Radiation Physics
struct RadiationPhysics {
  static constexpr bool dust{false};
  static constexpr bool pressure{false};
  static constexpr bool eos{false};

  static constexpr bool mhd{false};
  static constexpr int nvar{1+COMPONENTS};

  static constexpr int radiation{true};
};
```

Radiation as a type of fluid

```
template <typename Phys>
KOKKOS_INLINE_FUNCTION void K_ConsToPrim(real Vc[], real Uc[], const EquationOfState *eos) {
  Vc[RHO] = Uc[RHO];

  if constexpr(Phys::radiation) {
      EXPAND( Vc[FR1] = Uc[FR1];  ,
              Vc[FR2] = Uc[FR2];  ,
              Vc[FR3] = Uc[FR3];  )
  } else {
      EXPAND( Vc[VX1] = Uc[MX1]/Uc[RHO];  ,
              Vc[VX2] = Uc[MX2]/Uc[RHO];  ,
              Vc[VX3] = Uc[MX3]/Uc[RHO];  )
  }
  if constexpr(Phys::mhd) {
    EXPAND( Vc[BX1] = Uc[BX1];  ,
            Vc[BX2] = Uc[BX2];  ,
            Vc[BX3] = Uc[BX3];  )
  }
  if constexpr(Phys::pressure) {
    real kin = HALF_F / Uc[RHO] * (EXPAND( Uc[MX1]*Uc[MX1]   ,
                                         + Uc[MX2]*Uc[MX2]   ,
                                         + Uc[MX3]*Uc[MX3]   ));

    if constexpr(Phys::mhd) {
      real mag = HALF_F * (EXPAND( Uc[BX1]*Uc[BX1]   ,
                                 + Uc[BX2]*Uc[BX2]   ,
                                 + Uc[BX3]*Uc[BX3]   ));

      Vc[PRS] = eos->GetPressure(Uc[ENG] - kin - mag, Uc[RHO]);

    } else { // Hydro case
      Vc[PRS] = eos->GetPressure(Uc[ENG] - kin, Uc[RHO]);

    } // MHD
  } // Have Energy
}
```

<u>ConsToPrim function template</u>

One class template for all fluid types (very easy to add a new fluid)

Choice of physics is made at instantiation of class

Conclusion

M1 radiation module of IDEFIX passed tests:

- In 1D, 2D, 3D
- In cartesian, cylindrical and spherical coordinates
- With HD (to test with MHD and/or dust yet)
- With reduced speed of light
- With analytical or imported table of opacities (1D or 2D)
- With an external source of irradiation (in progress)

Next step is to test and optimize performance on GPUs.