1 6/8/21: Solving the avective equation

Consider the wave equation:

$$\frac{\partial^2\phi}{\partial r^2} - \frac{1}{v^2}\frac{\partial^2\phi}{\partial t^2} = 0, \tag{1}$$

which describes traveling waves. The solution to the wave equation, $\phi(t, r)$, is a function of both time, t, and position, r. There are a few different ways to describe what is a wave. Here is a useful one for us now: A wave holds its shape and travels at the constant velocity v. If we were to solve this partial differential equation (PDE) on the computer, then, we would expect the shape we give it as its initial condition to not change as it travels left or right at the speed v.

We could solve the wave equation, but let's do something simpler to start off with. The wave equation factors:

$$\left(\frac{\partial}{\partial r} + \frac{1}{v}\frac{\partial}{\partial t}\right)\left(\frac{\partial}{\partial r} - \frac{1}{v}\frac{\partial}{\partial t}\right)\phi = 0. \tag{2}$$

Thus, if either of

$$\left(\frac{\partial}{\partial r} + \frac{1}{v}\frac{\partial}{\partial t}\right)\phi = 0, \qquad \left(\frac{\partial}{\partial r} - \frac{1}{v}\frac{\partial}{\partial t}\right) = 0 \tag{3}$$

are satisfied, the wave equation is satisfied. It can be shown that the solutions to the equation on the left represent *right-moving* waves and solutions to the equation on the right represent *left-moving* waves. We will solve for left-moving waves by solving

$$\frac{\partial\phi}{\partial t} = v\frac{\partial\phi}{\partial r}, \tag{4}$$

which is called the avective equation. Importantly, I isolated the time-derivative on the left hand side, which is the form we want for the solution method we are going to use.

To solve PDEs we will use *finite-differencing* to handle *spatial*-derivatives. A different technique will be used for the time-derivatives. This means we can write the avective equation as

$$\frac{\partial\phi(t, r_i)}{\partial t} = \frac{v}{r_{i+1} - r_{i-1}}\left[\phi(t, r_{i+1}) - \phi(t, r_{i-1})\right]. \tag{5}$$

This finite differencing is said to be second order accurate.

To handle the time derivative we treat the differential equation in (5) as *the* differential equation we are solving (instead of the one in (4)) (this is called the *method of lines*). We have a first-order ODE, then, and we use Runge-Kutta to solve the time direction.

Coding wise, the first thing to do is set up a grid *that does not include* $r = 0$. We often divide by r, so we don't want to actually include $r = 0$. This is called a staggered grid. We create a large 2D array. We picture it as the horizontal direction is the spatial direction and the vertical direction is the temporal direction. The spatial points are set up as

$$r_i = (i + 1/2 - 3)\Delta r, \tag{6}$$

where Δr is the *grid spacing*. Note the factor of 3. It's important we include a few points to the left of $r = 0$. These are not physical and are called *virtual grid points*. They are necessary for computing the finite-differenced spatial derivatives. We have then

$$r_0 = -2.5\Delta r, \qquad r_1 = -1.5\Delta r, \qquad r_2 = -0.5\Delta r, \qquad r_3 = 0.5\Delta r, \qquad \cdots. \tag{7}$$

Note that $i = 3$ corresponds to the first *positive* grid point. It's important to keep this in mind. Let's start with, say, 1000 spatial points and a grid spacing of $\Delta r = 0.01$.

You also have the temporal points, which can include $t = 0$:

$$t_n = n\Delta t, \qquad n = 0, 1, 2 \ldots. \tag{8}$$

A rule of thumb is that you need

$$\gamma = \frac{\Delta t}{\Delta x} \leq 0.5 \tag{9}$$

for your code to retain stability while solving the ODE. Set $\gamma = 0.5$ and again use 1000 temporal points to start off.

We need an initial condition, which is the initial shape of the wave. We'll use a Gaussian:

$$\phi(0, r) = \alpha e^{-\beta(r-a)^2}. \tag{10}$$

Note that a is the center of the wave, so that the Gaussian is centered at $r = a$. You want to place the center away from the origin at first, so use something like $r = 5$. You need to *discretize this* and load it into the $n = 0$ row of your array. That is, the individual components of the $n = 0$ row of your array should have the values

$$\phi(0, r_i) = \alpha e^{-\beta(r_i-a)^2}, \qquad r_i = (i + 1/2 - 3)\Delta r. \tag{11}$$

The values of the virtual grid points should be set up as follows:

$$\phi(0, r_{-i}) = \phi(0, r_i). \tag{12}$$

This is because the initial condition should be an *even* function.

Now comes the challenging part: Solving the PDE. You begin at $n = 0$. Using a loop, you start at $i = 0$ and move through the spatial grid points. At each value of i, you compute the right hand side of the PDE in (5). You then use Runge-Kutta to evolve forward one step in time and to find $\phi(t_{n+1}, r_i)$, which is stored in the $n = 1, i$ element of the array.

2 6/9/21

I want you to try and solve a physical problem that has a single particle coupled to gravity. The particle is as simple as it gets in particle physics because it is spin-0, or has no spin. In particle physics it is called a *scalar field*.

There will be two functions that are used to describe gravity:

$$a = a(t, r), \qquad \alpha = \alpha(t, r). \tag{13}$$

Solving for these will take a few new ideas from what you've done with the wave equation (in other words, the wave equation did not have fields like these). The scalar field has three fields associated with it:

$$\phi = \phi(t, r), \qquad \Phi = \Phi(t, r), \qquad \Pi = \Pi(t, r). \tag{14}$$

The exact definitions of Φ and Π are $\Phi = \partial_r \phi$ and $\Pi = (a/\alpha)\partial_t \phi$, though you probably won't use these. But you can perhaps see from them that Φ and Π are used to turn second order differential equations into a system of first order differential equations.

There are no parameters that need to be specified in this model.

The equations for ϕ, Φ, and Π are

$$\begin{aligned}\partial_t \phi &= \frac{\alpha}{a}\Pi \\ \partial_t \Phi &= \partial_r \left(\frac{\alpha \Pi}{a}\right) \\ \partial_t \Pi &= \frac{1}{r^2}\partial_r \left(\frac{r^2 \alpha}{a}\Phi\right).\end{aligned} \tag{15}$$

As you can see these are PDEs and we solve them using the method of lines, i.e. by using Runge-Kutta *in the time direction only* and finite-differencing for the spatial derivatives.

The equations for α and a are different. They are ODEs only:

$$\begin{aligned}\partial_r \alpha &= 4\pi G r a^2 \alpha S^r{}_r + \frac{\alpha(a^2-1)}{2r} \\ \partial_r a &= 4\pi G r a^3 \rho - \frac{a(a^2-1)}{2r}\end{aligned} \tag{16}$$

where

$$\begin{aligned}\rho &= \frac{\Phi^2 + \Pi^2}{2a^2} \\ S^r{}_r &= \frac{\Phi^2 + \Pi^2}{2a^2}\end{aligned} \tag{17}$$

(in this system $\rho = S^r{}_r$). In your code, simply set $G = 1$.

2.1 Initial Conditions

Use the initial condition

$$\phi(t=0, r) = C r^2 e^{-(r-r_0)^2}, \tag{18}$$

for some parameters C and r_0. You will need to take the analytical derivative of this (you can use Mathematica if you like, or do it by hand) to obtain the initial condition for Φ, since $\Phi = \partial_r \phi$. Use at first the values $C = 0.001$ and $r_0 = 5$.

We'll take $\Pi(t=0, r) = 0$, which, since $\Pi \propto \partial_t \phi$, means the scalar field is initially stationary.

2.2 Boundary Conditions

In your previous programs you used *periodic boundary conditions*. They are simpler and easier to code, but not very physical. Here we need to do something else.

At the inner boundary $r = 0$ we take ϕ to be even, Φ to be odd, and Π to be even. This determines their values at the inner virtual grid points.

For the outer boundary, we use conditions that allow the fields to leave the computational domain. They are

$$\begin{aligned}\partial_t \phi &= -\frac{\phi}{r} - \Phi \\ \partial_t \Pi &= -\frac{\Pi}{r} - \partial_r \Pi \\ \Phi &= -\frac{\phi}{r} - \Pi.\end{aligned} \tag{19}$$

Since these are only used at the outer grid point, the spatial derivative $\partial_r \Pi$ will have to be finite-differenced using *one-sided* finite differencing. Finite differencing coefficients can be found on Wikipedia here:

https://en.wikipedia.org/wiki/Finite_difference_coefficient

The boundary conditions for a and α are slightly different. The first grid point for positive r (i.e. the first non-virtual grid point) is called the *inner-most grid point*. At the inner-most grid point, we have

$$a = 1, \qquad \alpha = 1. \tag{20}$$

Later, we will change these slightly, but start with these.

2.3 Procedure

Set up a grid that *does not include* $r = 0$ and that has a few negative grid points.

Input the initial conditions for ϕ and Φ.

On a given time step, solve for a and α using Runge-Kutta in the r direction.

Then use Runge-Kutta to solve for ϕ, Φ, and Π on the next time step using Runge-Kutta in the time direction, as you did in your previous programs.

Now repeat.

3 6/13/21: Finite differencing the $\partial_t \Pi$ equation

In equation (15) the PDE for Π is

$$\partial_t \Pi = \frac{1}{r^2} \partial_r \left(\frac{r^2 \alpha}{a} \Phi \right). \tag{21}$$

A straight forward finite differencing of the right hand side is:

$$\partial_t \Pi = \frac{1}{r^2} \frac{1}{(r_{i+1} - r_{i-1})} \left[\left(\frac{r^2 \alpha}{a} \Phi \right)_{i+1} - \left(\frac{r^2 \alpha}{a} \Phi \right)_{i-1} \right], \tag{22}$$

where $r_{i+1} - r_{i-1} = 2\Delta r$. Unfortunately, this finite differencing doesn't work around the origin and causes problems. A common fix is to finite difference as

$$\partial_t \Pi = \frac{3}{r_{i+1}^3 - r_{i-1}^3} \left[\left(\frac{r^2 \alpha}{a} \Phi \right)_{i+1} - \left(\frac{r^2 \alpha}{a} \Phi \right)_{i-1} \right]. \tag{23}$$

Note that there is no longer a $1/r^2$ out front. This right hand side ends up being exactly the same, but handles the origin just fine. Further, it gives the same answer whether used at the origin or anywhere else and can be used over the entire computational domain.

4 6/14/21: Hydrodynamics

We are going to learn how to simulate a neutron star. The code to do this is similar in strategy to the code you have written, but is much more complicated. It is also somewhat new to me, making it even more complicated, since I'm probably going to be confused on some of the finer details as we put this together. Still, once we've put this together, we will be able to simulate a truly physical system, a neutron star.

4.1 Modified-Euler

Lately you've been using fourth order Runge-Kutta (RK4). We need all the speed we can get, so we are going to use a less accurate, but faster algorithm called modified-Euler. For differential equation

$$y' = f(y), \tag{24}$$

the modified-Euler algorithm is

$$\begin{aligned} k_1 &= hf(y) \\ k_2 &= hf(y + k_1) \\ y(t+h) &= y(t) + \frac{1}{2}(k_1 + k_2). \end{aligned} \tag{25}$$

4.2 Primitive variables, conservative variables, and the equation of state

There are two different sets of variables that we will use to describe a neutron star. The first set are called the *primitive variables*,

$$\rho(t,r), \qquad P(t,r), \qquad v(t,r). \tag{26}$$

We describe a neutron star in terms of a *fluid*, and ρ is the energy density of the fluid, P is the pressure, and v is the velocity. The variables are functions of time and position. We are assuming spherical symmetry, so position is given by r, the distance from the center of the star. The energy density and pressure are not independent. They are connected through an *equation of state*. The equation of state is where different models for a neutron star come into the theory. We will start out with an equation of state called the *ultrarelativistic equation of state*,

$$P = (\Gamma - 1)\rho, \tag{27}$$

where $\Gamma > 1$ is a constant. This is not the equation of state we will be interested in, in the long run, but it has some convenient properties, making it the best one to start with.

The second set of variables are called the *conservative variables*. They are

$$\Pi(t,r), \qquad \Phi(t,r) \tag{28}$$

and are defined by

$$\Pi = \frac{\rho + P}{1 - v} - P, \qquad \Phi = \frac{\rho + P}{1 + v} - P \tag{29}$$

This definition allows us to determine the conservative variables from the primitive variables.

We will need to know how to determine the primitive variables from the conservative variables. Unfortunately, this is not, in general, straightforward. Trying to invert (29) leads to

$$(\Pi - \Phi)^2 = (\Pi + \Phi - 2\rho)(\Pi + \Phi + 2P), \qquad v = \frac{\Pi - \Phi}{\Pi + \Phi + 2P}. \tag{30}$$

The way in which these are solved for the primitive variables is as follows. Using the equation state, which tells us that P can be determined from ρ, the only unknown primitive variable in the left equation is ρ. We thus solve the left equation numerically for ρ. Having found ρ, we immediately have P from the equation of state and v using the right equation.

The value in the equation of state in (27) is that the left equation in (30) can be solved analytically, so that we don't have to solve it numerically. For a general equation of state, however,

the left equation in (30) will have to be solved numerically. For the equation of state in (27), the solution to the left equation in (30) is

$$P = -(\Pi + \Phi)\left(\frac{2-\Gamma}{4}\right) + \sqrt{(\Pi+\Phi)^2\left(\frac{2-\Gamma}{4}\right)^2 + (\Gamma-1)\Pi\Phi}. \tag{31}$$

Using (27), the analytical expression for ρ is

$$\rho = -(\Pi + \Phi)\frac{2-\Gamma}{4(\Gamma-1)} + \frac{1}{\Gamma-1}\sqrt{(\Pi+\Phi)^2\left(\frac{2-\Gamma}{4}\right)^2 + (\Gamma-1)\Pi\Phi}. \tag{32}$$

When using the relativistic equation of state, I would like you to write your code so that you use (32), i.e. the first primitive variable you find is ρ. Then use the equation of state in (27) to find P (instead of using the analytical formula in (31)). The reason is that this generalizes better to an upcoming equation of state that I will have you use.

4.3 Fluid equations

The differential equations that describe the fluid are

$$\partial_t \mathbf{u} = -\frac{1}{r^2}\partial_r\left(r^2\frac{\alpha}{a}\mathbf{f}\right) + \mathbf{s}, \tag{33}$$

where

$$\mathbf{u} = \begin{pmatrix}\Pi\\ \Phi\end{pmatrix} \qquad \mathbf{f} = \begin{pmatrix}\frac{1}{2}(\Pi-\Phi)(1+v)+P\\ \frac{1}{2}(\Pi-\Phi)(1-v)-P\end{pmatrix} \qquad \mathbf{s} = \begin{pmatrix}+\Sigma\\ -\Sigma\end{pmatrix} \tag{34}$$

and

$$\Sigma = \Theta + \frac{2\alpha}{ra}P, \qquad \Theta = \frac{1}{2}\left[(\Pi-\Phi)v - (\Pi+\Phi)\right]\left[8\pi ra\alpha P + \frac{a\alpha}{2r}\left(1-\frac{1}{a^2}\right)\right] + \frac{a\alpha}{2r}\left(1-\frac{1}{a^2}\right)P. \tag{35}$$

Note that (33) is actually two equations, because it is written in terms of two-component vectors. Throughout, you should think of $\mathbf{u}$ as representing either Π or Φ, the two conservative variables.

One of the difficulties in simulating a neutron star is that the right hand side of (33) is written in terms of a mixture of conservative and primitive variables. The solution to (33), which is $\mathbf{u}$, gives the conservative variables. And thus, at each time step, we will have solved for the conservative variables. But to then try to solve for the next time step, we will need to evaluate the right hand side of (33), which requires also knowing the primitive variables. This is the reason why we will need to solve the equations in (30), which give the primitive variables from the conservative variables. For the ultrarelativistic equation of state, we can instead use the analytical solution in (31).

4.3.1 Alternative form

There is an alternative way to write the equation in (33) that helps with solving it. Defining,

$$\mathbf{f}^{(1)} = \begin{pmatrix}\frac{1}{2}(\Pi-\Phi)(1+v)\\ \frac{1}{2}(\Pi-\Phi)(1-v)\end{pmatrix} \qquad \mathbf{f}^{(2)} = \begin{pmatrix}+P\\ -P\end{pmatrix}, \tag{36}$$

we have that

$$\mathbf{f} = \mathbf{f}^{(1)} + \mathbf{f}^{(2)}. \tag{37}$$

When we plug this into (33), we have

$$\begin{aligned} -\frac{1}{r^2}\partial_r\left(r^2\frac{\alpha}{a}\mathbf{f}\right) &= -\frac{1}{r^2}\partial_r\left(r^2\frac{\alpha}{a}\mathbf{f}^{(1)}\right) - \frac{1}{r^2}\partial_r\left(r^2\frac{\alpha}{a}\mathbf{f}^{(2)}\right) \\ &= -\frac{1}{r^2}\partial_r\left(r^2\frac{\alpha}{a}\mathbf{f}^{(1)}\right) - \partial_r\left(\frac{\alpha}{a}\mathbf{f}^{(2)}\right) - \frac{2}{r}\frac{\alpha}{a}\mathbf{f}^{(2)}. \end{aligned} \tag{38}$$

The last term on the bottom line exactly cancels the $2\alpha P/ra$ term in Σ in (35). Equation (33) can therefore be written

$$\partial_t\mathbf{u} = -\frac{1}{r^2}\partial_r\left(r^2\frac{\alpha}{a}\mathbf{f}^{(1)}\right) - \partial_r\left(\frac{\alpha}{a}\mathbf{f}^{(2)}\right) + \mathbf{s}', \tag{39}$$

where

$$\mathbf{u} = \begin{pmatrix}\Pi\\ \Phi\end{pmatrix} \qquad \mathbf{f}^{(1)} = \begin{pmatrix}\frac{1}{2}(\Pi-\Phi)(1+v)\\ \frac{1}{2}(\Pi-\Phi)(1-v)\end{pmatrix} \qquad \mathbf{f}^{(2)} = \begin{pmatrix}+P\\ -P\end{pmatrix}, \qquad \mathbf{s}' = \begin{pmatrix}+\Theta\\ -\Theta\end{pmatrix} \tag{40}$$

and

$$\Theta = \frac{1}{2}\left[(\Pi-\Phi)v - (\Pi+\Phi)\right]\left[8\pi ra\alpha P + \frac{a\alpha}{2r}\left(1-\frac{1}{a^2}\right)\right] + \frac{a\alpha}{2r}\left(1-\frac{1}{a^2}\right)P. \tag{41}$$

The important takeaway is that if make the decomposition of the flux $\mathbf{f}$ in (37), then the source is given by $\mathbf{s}'$, with the $2\alpha P/ra$ term canceling out.

4.4 Gravity equations

The differential equation in (33) describes the fluid, or matter, that makes up a neutron star. We also need equations that describe gravity. Consider first the equation for α, which you have seen before:

$$\partial_r\alpha = 4\pi ra^2\alpha\left[\frac{1}{2}(\Pi-\Phi)v + P\right] + \frac{\alpha(a^2-1)}{2r}. \tag{42}$$

This equation can be rewritten using

$$\partial_r(\ln\alpha) = \frac{\partial_r\alpha}{\alpha}, \tag{43}$$

which gives

$$\partial_r(\ln\alpha) = 4\pi ra^2\left[\frac{1}{2}(\Pi-\Phi)v + P\right] + \frac{a^2-1}{2r}. \tag{44}$$

The convenience of this form is that we do not need to know α to evaluate the right hand side. This equation can be solved using second order Runge-Kutta (RK2):

$$(\ln\alpha)^n_{i+1/2} = (\ln\alpha)^n_{i-1/2} + \Delta r\left\{4\pi ra^2\left[\frac{1}{2}(\Pi-\Phi)v + P\right] + \frac{a^2-1}{2r}\right\}^n_i. \tag{45}$$

The actual way we will do this is to start at the outer boundary and work our way in:

$$(\ln\alpha)^n_{i-1/2} = (\ln\alpha)^n_{i+1/2} - \Delta r\left\{4\pi ra^2\left[\frac{1}{2}(\Pi-\Phi)v + P\right] + \frac{a^2-1}{2r}\right\}^n_i. \tag{46}$$

Exponentiating, we have an equation for α:

$$\alpha^n_{i-1/2} = \alpha^n_{i+1/2}\exp\left(-\Delta r\left\{4\pi ra^2\left[\frac{1}{2}(\Pi-\Phi)v + P\right] + \frac{a^2-1}{2r}\right\}^n_i\right). \tag{47}$$

There are actually two equations for a. The one that you have seen before is

$$\partial_r a = 4\pi r a^3 \frac{\Pi + \Phi}{2} - \frac{a(a^2-1)}{2r}. \tag{48}$$

You will only use this one to handle the initial data. The one that you will mostly use is

$$\partial_t a = -4\pi r \alpha a^2 \frac{\Pi - \Phi}{2}. \tag{49}$$

Note that this has a time derivative.

4.5 High resolution shock capturing (HRSC) methods

The methods we will use to solve the fluid equations in (33) are called HRSC methods.

4.5.1 Grid

As usual, we discretize space and time. For the spatial part of our grid, we will think about it a little differently than we did previously. First, we define our grid as

$$r_i = (i - 1 + 0)\Delta r, \qquad t_n = n\Delta t. \tag{50}$$

For the spatial grid, include *one* virtual grid point and *include zero*. The first few spatial grid points are

$$r_0 = -\Delta r, \qquad r_1 = 0, \qquad r_2 = \Delta r, \qquad r_3 = 2\Delta r, \qquad \ldots. \tag{51}$$

The discretization of time is the same as before.

For the discretization of space, each grid point, r_i, is at the center of a *cell*, $\mathcal{C}_i$, with domain $[r_{i-1/2}, r_{i+1/2}]$ and length Δr. Take special note that the r_i are at the center of a cell and are our grid points in our code. And that the cell boundaries are $r_{i-1/2}$ and $r_{i+1/2}$, but are not included as grid points in our code.

4.5.2 Finite volumes

We want to solve the fluid equations in (33), which are

$$\partial_t \mathbf{u} + \frac{1}{r^2}\partial_r \left[r^2 \frac{\alpha}{a} \mathbf{f}(\mathbf{u}) \right] = \mathbf{s}. \tag{52}$$

Recall that $\mathbf{u}$ represents the conservative variables Π and Φ. $\mathbf{f}$ is called the flux, and in the equation above I've written it has $\mathbf{f}(\mathbf{u})$, indicating that it depends on the conservative variables, as can easily be seen from (34). $\mathbf{f}$ also depends on the primitive variables, but since the primitive variables can be determined from the conservative variables, in this section we shall think of $\mathbf{f}$ as depending on the conservative variables only.

The problem with trying to solve (52) directly is that fluids have discontinuities and derivatives are problematic at discontinuities. On the other hand, integrals handle discontinuities just fine. The idea, then, is to rewrite (52) in terms of integrals.

Multiplying by r^2 and integrating both sides over an arbitrary cell gives

$$\frac{d}{dt}\int_{r_i-1/2}^{r_i+1/2} \mathbf{u}(t,r) r^2 dr + \left[r^2 \frac{\alpha(t,r)}{a(t,r)} \mathbf{f}(\mathbf{u}) \right]_{r-1/2}^{r+1/2} = \int_{r_i-1/2}^{r_i+1/2} \mathbf{s}(t,r) r^2 dr. \tag{53}$$

Defining the cell averaged conservative variables, $\bar{\mathbf{u}}_i(t)$, and the cell averaged sources, $\bar{\mathbf{s}}_i(t)$,

$$\bar{\mathbf{u}}_i(t) \equiv \frac{1}{r_i^2 \Delta r} \int_{r_i-1/2}^{r_i+1/2} \mathbf{u}(t,r) r^2 \, dr, \qquad \bar{\mathbf{s}}_i(t) \equiv \frac{1}{r_i^2 \Delta r} \int_{r_i-1/2}^{r_i+1/2} \mathbf{s}(t,r) r^2 \, dr, \tag{54}$$

as well as

$$A_{i\pm1/2}(t) \equiv r^2 \frac{\alpha(t,r)}{a(t,r)}\bigg|_{r=r_i\pm1/2}, \qquad \mathbf{f}_{i\pm1/2}(t) \equiv \mathbf{f}(\mathbf{u}(t,r))\bigg|_{r=r_i\pm1/2}, \tag{55}$$

the conservation equation becomes

$$\frac{d\bar{\mathbf{u}}_i}{dt} = -\frac{1}{r_i^2 \Delta r}\left[A_{i+1/2}\mathbf{f}_{i+1/2} - A_{i-1/2}\mathbf{f}_{i-1/2}\right] + \bar{\mathbf{s}}_i. \tag{56}$$

This form of the equation is written in terms of the cell averaged quantities and the cell averaged quantities can handle discontinuities.

There are two immediate challenges to solving (56). Let's say we know the solution $\bar{\mathbf{u}}_i^n$ at time t_n. To find the solution $\bar{\mathbf{u}}_i^{n+1}$ at the next time step we will use modified-Euler (just as you used Runge-Kutta in the previous code you've written), which requires evaluating the right hand side of (56). But we know the *cell averaged* quantities, $\bar{\mathbf{u}}_i^n$, which does not allow use to determine the fluxes, $f_{i+1/2}$, which are evaluated at the cell boundaries, nor the cell averaged sources $\bar{\mathbf{s}}_i$, which require integrating over the entire cell. The sources are an easy fix. We use the approximation

$$\bar{\mathbf{s}}_i(t_n) \approx \mathbf{s}(\bar{\mathbf{u}}_i^n(t)), \tag{57}$$

that is, we simply take the functions that define the sources and evaluate them with the cell averaged $\bar{\mathbf{u}}_i^n$.

Determining the fluxes is much more complicated. As already mentioned, the fluxes are a function of the conservation variables $\mathbf{u}$ and, at each time step, we know the cell averaged values $\bar{\mathbf{u}}_i$. These cell averaged values are *discontinuous* at the cell boundaries, precisely where we need to know the fluxes. How then are we going to determine the value of $\mathbf{u}$ at the cell boundaries, when what we do know is discontinuous there? It turns out, this is a *Riemann problem* and by solving the Riemann problem at each cell boundary, we can determine the fluxes there. This method is called a *Godunov method*.

4.5.3 Alternative form

Before considering the Riemann problem and the Godunov method, I want to write (56) in an alternative form. In section 4.3.1, I showed that the original differential equation in (33) can be written in the alternative form (39), by decomposing the fluxes as in (37). The purpose of doing this is that it leads to a cancellation of the term $2\alpha P/ra$, which makes solving the equations easier. We therefore do the same with the cell averaged version in (56), by writing it as

$$\frac{d\bar{\mathbf{u}}_i}{dt} = -\frac{1}{r_i^2 \Delta r}\left[\left(r^2\frac{\alpha}{a}\mathbf{f}^{(1)}\right)_{i+1/2} - \left(r^2\frac{\alpha}{a}\mathbf{f}^{(1)}\right)_{i-1/2}\right] - \frac{1}{\Delta r}\left[\left(\frac{\alpha}{a}\mathbf{f}^{(2)}\right)_{i+1/2} - \left(\frac{\alpha}{a}\mathbf{f}^{(2)}\right)_{i-1/2}\right] + \bar{\mathbf{s}}_i'. \tag{58}$$

The first term on the right hand side of (58) has the appearance of finite differencing. Moreover, this finite differencing is precisely of the form that we found problematic (see eq. (21)). The fix we used was

$$\frac{1}{r^2\Delta r} \to \frac{3}{r_{i+1/2}^3 - r_{i-1/2}^3}. \tag{59}$$

Finally, I want to change the notation from $\mathbf{f}$ to $\mathbf{F}$. This change in notation indicates that we must use some method, called a Rimeann solver, to determine the actual value of the $\mathbf{f}$'s, which we then use in the following equation:

$$\begin{aligned}\frac{d\bar{\mathbf{u}}_i}{dt} = &-\frac{3}{r_{i+1/2}^3 - r_{i-1/2}^3}\left[\left(r^2\frac{\alpha}{a}\mathbf{F}^{(1)}\right)_{i+1/2} - \left(r^2\frac{\alpha}{a}\mathbf{F}^{(1)}\right)_{i-1/2}\right] \\ &- \frac{1}{\Delta r}\left[\left(\frac{\alpha}{a}\mathbf{F}^{(2)}\right)_{i+1/2} - \left(\frac{\alpha}{a}\mathbf{F}^{(2)}\right)_{i-1/2}\right] + \bar{\mathbf{s}}_i'.\end{aligned} \tag{60}$$

4.5.4 Godunov methods

We have partitioned space into cells $\mathcal{C}_i$ with domain $[r_{i-1/2}, r_{i+1/2}]$ and length Δr. At each time step we know the cell averaged quantities $\bar{\mathbf{u}}_i^n$, which is not the same as the true value $\mathbf{u}^n(r)$. The simplest thing we can do is to approximate the true value with the cell averaged value:

$$\mathbf{u}^n(r) = \bar{\mathbf{u}}_i^n \quad \text{for } r_{i-1/2} < r < r_{i+1/2}. \tag{61}$$

This is actually too strong of an approximation and we are not going to do this in our code. But, let's begin here to understand some of the relevant concepts.

Consider the specific cell boundary at $r = r_{i+1/2}$ between cells $\mathcal{C}_i$ and $\mathcal{C}_{i+1}$. At a given time step, we have the following solution on either side of the boundary:

$$\mathbf{u}^n(r) = \begin{cases} \bar{\mathbf{u}}_i^n & \text{for } r_{i-1/2} < r < r_{i+1/2} \\ \bar{\mathbf{u}}_{i+1}^n & \text{for } r_{i+1/2} < r < r_{i+3/2}. \end{cases} \tag{62}$$

At the boundary, then, we have a discontinuous jump from $\bar{\mathbf{u}}_i^n$ to $\bar{\mathbf{u}}_{i+1}^n$. This is the definition of a Riemann problem.

Godunov's idea is to solve the Riemann problem. The solution to the Riemann problem gives the conservation variables at the cell boundary: $\mathbf{u}_{i+1/2}^n$. And this is precisely what we need to know to determine $f_{i\pm1/2}$ in (58). The Godunov method, then, will allow us to evaluate the right hand side of (58).

I mentioned that approximating the true value $\mathbf{u}^n(r)$ with the cell averaged value $\bar{\mathbf{u}}_i^n$, as written in (61), is too strong of an approximation. We can use a better approximation by using a polynomial representation of the solution. This is called *cell reconstruction*.

4.5.5 Cell reconstruction

We have partitioned space into cells $\mathcal{C}_i$ with domain $[r_{i-1/2}, r_{i+1/2}]$ and length Δr. At each time step we know the cell averaged quantities $\bar{\mathbf{u}}_i^n$. From these, we will define the functions $\tilde{\mathbf{u}}^n(r)$, which are said to be *reconstructed* from the cell averaged $\bar{\mathbf{u}}_i^n$. We will then use the reconstructions $\tilde{\mathbf{u}}_i^n$ to approximate the true value,

$$\mathbf{u}^n(r) = \tilde{\mathbf{u}}^n(r). \tag{63}$$

There are many different cell reconstruction methods, i.e. there are many different methods for defining the $\tilde{\mathbf{u}}^n(r)$. We will use so-called *slope-limiter methods*. Slope-limiter methods use a piecewise-linear reconstruction:

$$\tilde{\mathbf{u}}^n(r) = \bar{\mathbf{u}}_i^n + \sigma_i^n(r - r_i) \quad \text{for } r_{i-1/2} < r < r_{i+1/2}. \tag{64}$$

This reconstruction is linear in r with σ_i^n being a slope. There are many different methods for how to define the slope. We will use the *minmod slope limiter*. Defining

$$\mathbf{s}_{i+1/2}^n \equiv \frac{\bar{\mathbf{u}}_{i+1}^n - \bar{\mathbf{u}}_i^n}{r_{i+1} - r_i}, \tag{65}$$

which is the slope or derivative of $\bar{\mathbf{u}}_i^n$ across the cell boundary at $r_{i+1/2}$. Then,

$$\vec{\sigma}_i^n = \text{minmod}(\mathbf{s}_{i-1/2}^n, \mathbf{s}_{i+1/2}^n), \tag{66}$$

where

$$\text{minmod}(a,b) = \begin{cases} a & \text{if } |a| < |b| \text{ and } ab > 0 \\ b & \text{if } |b| < |a| \text{ and } ab > 0 \\ 0 & \text{if } ab \leq 0. \end{cases} \tag{67}$$

Equation (64) gives $\tilde{\mathbf{u}}^n(r)$ for any r inside a cell. We use this quantity to approximate the true value in the cell:

$$\mathbf{u}^n(r) = \tilde{\mathbf{u}}^n(r) = \bar{\mathbf{u}}_i^n + \vec{\sigma}_i^n(r - r_i) \quad \text{for } r_{i-1/2} < r < r_{i+1/2}. \tag{68}$$

Consider two adjacent cells $\mathcal{C}_i$ and $\mathcal{C}_{i+1}$, with common boundary at $r_{i+1/2}$. Each of these cells has a $\tilde{\mathbf{u}}^n(r)$ valid inside it and each of these $\tilde{\mathbf{u}}^n(r)$ gives a value at the common cell boundary. Specifically, we have the following two values at the common cell boundary at $r_{i+1/2}$:

$$\begin{aligned} \tilde{\mathbf{u}}_{i+1/2}^{n,L} &\equiv \bar{\mathbf{u}}_i^n + \vec{\sigma}_i^n(r_{i+1/2} - r_i) \\ \tilde{\mathbf{u}}_{i+1/2}^{n,R} &\equiv \bar{\mathbf{u}}_{i+1}^n + \vec{\sigma}_{i+1}^n(r_{i+1/2} - r_{i+1}). \end{aligned} \tag{69}$$

It is these values that are used in the local Riemann problem at $r = r_{i+1/2}$.

4.5.6 Approximate Riemann solvers

We want to solve the local Riemann problems for Godunov's method. It is possible to get numerical solutions to any desired accuracy, but at rather high numerical cost, i.e. it is slow. For this reason, approximate Riemann solvers are used that give in general very accurate answers. There are two approximate Riemann solvers that we might use. For now, we will just consider one of them. It is called the *HLLE solver*.

We assume, at a particular cell boundary, that we have the left and right reconstructed values for Π and Φ from (69). With these, we compute left and right values for P using (30), or the analytical expression in (31) when using the ultrarelativistic equation of state. We then compute left and right values for ρ using the equation of state and left and right values for v using the equation on the right in (30). We now have left and right values of all fluid variables at the cell boundary. With these, we can construct left and right version of $\mathbf{f}^{(1)}$ and $\mathbf{f}^{(2)}$ using (40). Note that each one of these has two components.

The next thing we need are the eigenvalues of a 2×2 matrix. The matrix is written

$$\mathcal{A} = \begin{pmatrix} \mathcal{A}_{11} & \mathcal{A}_{12} \\ \mathcal{A}_{21} & \mathcal{A}_{22} \end{pmatrix}, \tag{70}$$

where

$$\begin{aligned}
\mathcal{A}_{11} &= \frac{1}{2}(1+2v-v^2) + (1-v^2)\frac{\partial P}{\partial \Pi} \\
\mathcal{A}_{12} &= -\frac{1}{2}(1+v)^2 + (1-v^2)\frac{\partial P}{\partial \Phi} \\
\mathcal{A}_{21} &= \frac{1}{2}(1-v)^2 - (1-v^2)\frac{\partial P}{\partial \Pi} \\
\mathcal{A}_{22} &= \frac{1}{2}(-1+2v+v^2) - (1-v^2)\frac{\partial P}{\partial \Phi},
\end{aligned} \tag{71}$$

and

$$\frac{\partial P}{\partial \Pi} = \frac{\rho - P - 2\Phi}{(\Pi + \Phi - 2\rho) - \frac{\partial \rho}{\partial P}(\Pi + \Phi + 2P)}, \qquad \frac{\partial P}{\partial \Phi} = \frac{\rho - P - 2\Pi}{(\Pi + \Phi - 2\rho) - \frac{\partial \rho}{\partial P}(\Pi + \Phi + 2P)}. \tag{72}$$

Note that you will need to compute $\partial\rho/\partial P$ using the equation of state. The eigenvalues are

$$\lambda_1 = \frac{1}{2}\left[\operatorname{tr}\mathcal{A} + \sqrt{(\operatorname{tr}\mathcal{A})^2 - 4\det\mathcal{A}}\right], \qquad \lambda_2 = \frac{1}{2}\left[\operatorname{tr}\mathcal{A} - \sqrt{(\operatorname{tr}\mathcal{A})^2 - 4\det\mathcal{A}}\right], \tag{73}$$

where

$$\operatorname{tr}\mathcal{A} = \mathcal{A}_{11} + \mathcal{A}_{22}, \qquad \det\mathcal{A} = \mathcal{A}_{11}\mathcal{A}_{22} - \mathcal{A}_{12}\mathcal{A}_{21}. \tag{74}$$

You need to compute left and right versions of the two eigenvalues. You then compute

$$\lambda_+ \equiv \max(0, \lambda_1^L, \lambda_2^L, \lambda_1^R, \lambda_2^R), \qquad \lambda_- \equiv \min(0, \lambda_1^L, \lambda_2^L, \lambda_1^R, \lambda_2^R). \tag{75}$$

We are finally able to put everything together. We compute

$$\begin{aligned}
\mathbf{F}^{(1)} &= \frac{\lambda_+ \mathbf{f}_L^{(1)} - \lambda_- \mathbf{f}_R^{(1)} + \lambda_+\lambda_-(\mathbf{u}_R - \mathbf{u}_L)}{\lambda_+ - \lambda_-} \\
\mathbf{F}^{(2)} &= \frac{\lambda_+ \mathbf{f}_L^{(2)} - \lambda_- \mathbf{f}_R^{(2)}}{\lambda_+ - \lambda_-}.
\end{aligned} \tag{76}$$

Note that both equations are two-component vector equations, so that there is a total of four equations in all. These quantities are then used on the right hand side of (60).

4.6 Procedure

The first step is introduce initial data:

- Start by setting up the initial data for the primitive variables. Let's again use a Gaussian:

$$P = P_0 e^{-(r-r_0)^2/d^2}. \tag{77}$$

  Use numbers like $P_0 = 10^{-4}$, $R_0 = 5$, and $d = 2$. Then use the equation of state to get the initial ρ. Take $v = 0$ (i.e., the Gaussian is not initially moving). Then use (29) to get the initial data for the conservative variables.

- Our numerical scheme for solving this system doesn't work well if any fluid parameters (except v) are too small, such as equaling zero. We will need to make sure everything is at least as large as some minimum value, which we call the *floor*. Run through every Π, Φ, P, and ρ spatial point and, if it is smaller than the floor, set it equal to the floor. For now, use 10^{-12} as the floor.

- The initial data for the fluid variables is now set up. Next we need to determine α and a at time step $n = 0$. First compute a (and not α) using eq. (48), second-order Runge-Kutta (RK2), and $a(0) = 1$. Two things: First, when evaluating the right hand side of (48) at $r = 0$, the right hand side is zero, which you will need to set by hand. Second, when evaluating the right hand side at the point in the middle of two grid points, average all variables.

- Now determine α using (47). Do this by starting at the outermost grid point, using $\alpha = 1/a$ at that grid point. There is a very important subtly here. You are going to compute α at the *cell boundary* and not at the cell center. According to eq. (47), determining α at the cell boundary requires evaluating all non-α terms at the cell center, which is perfect, because that is where they are known. Here is our convention: The value of α stored at a grid point, gives the value of α at the cell boundary to the *left* of the grid point. Around the origin, this means

$$\alpha_0 = \alpha(-\Delta r - \Delta r/2), \qquad \alpha_1 = \alpha(0 - \Delta r/2), \qquad \alpha_2 = \alpha(\Delta r - \Delta r/2). \tag{78}$$

  In other words, both α_0 and α_1 represent virtual grid points.

- Lastly, set the virtual grid points. Use that Π, Φ, P, ρ, and a are even. Don't bother with v, as it is never used at a virtual grid point. Use that α is even, but be mindful that it has two virtual grid points.

Having established all variables at $n = 0$, we now compute results for $n = 1$. I want you to use modified-Euler, evolving Π and Φ using (60) and a using (49). α is not evolved in time, but is determined using (47). This is the heart of the code and is the challenging part. There are many things to keep in mind:

- The first thing you need to do is reconstruct Π and Φ, computing the left and right quantities in (69) at the cell boundaries. Use our convention: The values stored at a grid point correspond to the values at the cell boundary to the *left* of the grid point. Then compute the fluxes using (76) and the HLLE Riemann solver. Use again our convention for cell boundaries.

- After the fluxes are computed, you can compute k_1 for modified-Euler. There are a couple subtleties in this. You are evolving at the grid points (not at the cell boundaries). Sometimes you need to evaluate quantities at the grid points (for example, with the a equation), but you only know quantities at the cell boundaries (such as α). Average the quantities to find the value at the grid point. Sometimes it's the opposite: You need to know values at the cell boundaries, but you only know them at the grid points (such as a in eq. (60)). Again, average the quantities to find the value you need. At the outermost grid point, use $\alpha = 1/a$.

- Some of the things you will need to do on on the path to computing k_2 and after finding the solution at the next time step, is determine the primitive variables (P, ρ, and v) from the conservative variables using (31), the equation of state, and the second equation in (30). You will need to solve for α using (47). You will again need to reconstruct at the cell boundaries and compute the fluxes. You will need to evaluate the virtual grid points. You will need to implement the floor for Π and Φ. The goal is to obtain the value of every variable on the $n = 1$ time step (except for v at the virtual grid point).

- We have inner boundary conditions. The inner boundary is at $r = 0$. We always have $a(0) = 1$ and thus $a_1 = 1$. For $\Phi(0)$ and $\Pi(0)$ we use

$$\Phi_1 = \Pi_1 = \frac{\Phi_2 r_3^2 - \Phi_3 r_2^2}{r_3^2 - r_2^2}. \tag{79}$$

Note that we do not solve for a_1, Φ_1, and Π_1 using modified Euler, we instead determine their values using these inner boundary conditions.

- For outer boundary conditions, we can evolve a using modified Euler at the outermost grid point. For Φ and Π, however, we do not evolve them at the outermost grid point. We simply use

$$\Phi_{\text{end}} = \Phi_{\text{end-1}}, \qquad \Pi_{\text{end}} = \Pi_{\text{end-1}}, \tag{80}$$

where Φ_{end} and Π_{end} are Φ and Π at the outermost grid point.

Make sure to use *two* outer boundary points (in a sense, we can refer to these as outer virtual points, for which we want two). Both outer boundary points are determined using (80). We need two outer virtual points because of the minmod method use in cell reconstruction.

5 6/21/21

You've been writing your code using the ultrarelativistic equation of state in (27). The main value with this equation of state is that the relationships linking conservative and primitive variables, in particular the left equation in (30), can be solved analytically. You have used the analytical solution for the energy density in (32). Unfortunately, for just about any other equation of state, the left equation in (30) cannot be solved analytically, but must be solved numerically. When you have an algebraic equation, and you want to find a value in it, it is called root finding, and thus we need a root finding routine. The one we will use is called the *Newton-Raphson method*.

Read the basics about the method in Wikipedia (Wikipedia labels it as "Newton's method"). As with any method, Newton-Raphson requires the function whose root we want to find. For us, this follows from the left equation in (30) and is

$$f(\rho) \equiv (\Pi + \Phi - 2\rho)(\Pi + \Phi + 2P) - (\Pi - \Phi)^2. \tag{81}$$

We want to find the value of ρ that gives $f = 0$. The Newton-Raphson method requires knowing the derivative of f with respect to the variable you are searching for: $f' = df/d\rho$. Now, the way that the equation f works is that Π and Φ have been determined and we are looking for the value of ρ that satisfies $f = 0$. Thus, for the sake of this root finding, Π and Φ are constants. However, P directly depends on ρ through the equation of state. Thus, our derivative is

$$f' = \frac{df}{d\rho} = -2(\Pi + \Phi + 2P) + (\Pi + \Phi - 2\rho)\left(2\frac{dP}{d\rho}\right) \tag{82}$$

For any equation of state we are using, we will need to determine $dP/d\rho$. This is very simple for the ultrarelativistic equation of state:

$$\frac{dP}{d\rho} = \frac{d}{d\rho}\left[(\Gamma - 1)\rho\right] = \Gamma - 1. \tag{83}$$

Root finding routines also begin with an initial guess. A good guess is easy for us. Just use the current value of ρ stored at the grid point of interest.

There's one issue that may crop up. It could be that the updated value of ρ, which I'll call ρ_{new}, coming out of the Newton-Raphson method is negative. But, for us, ρ is always positive. At each iteration of the method, check to see if ρ_{new} is negative. If it is, use instead $\rho_{\text{new}} = \rho_{\text{old}}/2$, where ρ_{old} is the value from the previous iteration.

The final question is how many iterations to run for the Newton-Raphson method. The more iterations, the more accurate an answer, but the slower. There are two things. First, there should always be a maximum number of iterations, so that the loop cannot run forever. But you really want to terminate the loop based on the accuracy of the answer. Let's place the maximum number of iterations at 50, and the *tolerance* at 10^{-7}. The tolerance is defined by

$$\text{tol} = \frac{|\rho_{\text{new}} - \rho_{\text{old}}|}{\rho_{\text{new}} + \rho_{\text{old}}}. \tag{84}$$

Break the Newton-Raphson iteration when $\text{tol} < 10^{-7}$.

You can use the code based on the analytical equation and the new code based on this numerical root finding method for checking that your foot finding method is working. Once you have it working, we can move on to some different equations of state.

5.1 Polytropic equation of state

Once you have the Newton-Raphson method working, there are two new equations of state I would like to you implement. The first is called a *polytropic equation of state*. It is roughly as simple as the ultrarelativistic equation of state you are using, but does require root finding for determining the primitive variables from the conservative variables. It is given by

$$P = K\rho^{\Gamma}, \tag{85}$$

where K and γ are constants. It is easy to take the derivative:

$$\frac{dP}{d\rho} = K\Gamma\rho^{\Gamma-1}. \tag{86}$$

For now, use the values

$$K = 100, \qquad \Gamma = 2. \tag{87}$$

5.2 SLy equation of state

The other equation of state is called SLy. This one is referred to as *realistic*, because it includes the physics of neutrons, protons, electrons, and muons, among other things, and could realistically describe the matter inside a neutron star. The way that we are going to use it, however, is in terms of a *fit*. We are going to use a continuous analytical equation that has been fit to the results of a detailed and complicated computation. For the particular code that you are writing, we need a very smooth and continuous equation of state to work with.

Take a look at the article by Haensel and Potekhin. The equation of state is given in their equation (14), using the parameters in Table 1 (make sure to use the SLy parameters). They use the notation

$$\xi = \log\left(\frac{\rho}{\text{g cm}^{-3}}\right), \qquad \zeta = \log\left(\frac{P}{\text{dyn cm}^{-2}}\right). \tag{88}$$

The denominators are indicating the units that ρ and P must have. But we do not use these units, so we need to do a unit conversion. For pressure, we use units called GeV^4. And for energy density the same units of GeV^4. The conversions are

$$\frac{\text{g}}{\text{cm}^3} = 4.30955 \times 10^{-18}\ \text{GeV}^4. \tag{89}$$

and

$$\frac{\text{dyne}}{\text{cm}^2} = 4.7953 \times 10^{-39}\ \text{GeV}^4. \tag{90}$$

You'll need to write code that includes all the numbers in Table 1 and properly computes the pressure for a given energy density. The units conversions are very small numbers, but they will be inside the logs. You should evaluate the numbers in the logs, turning them into more reasonable size numbers. Finally, you will need to take the ρ derivative to form $dP/d\rho$. This is a little tedious, but is necessary for using the Newton-Raphson method.

Once this is up and running, your code will be able to simulate a very realistic neutron star.

6 6/29/19

A Fourier transform transforms a function from one form to another. Often times it is easier to analyze the transformed function. As an example, consider the function

$$x(t) = A\sin(2\pi f t). \tag{91}$$

This function oscillates *exactly* at a frequency f. The Fourier transform is defined by

$$X(k) = \int_{-\infty}^{\infty} x(t) e^{-2\pi i k t} dt. \tag{92}$$

For $x(t) = A\sin(2\pi f t)$, this can be evaluated analytically, although the answer may not be something you have seen before. The answer is

$$\begin{aligned} X(k) &= \int_{-\infty}^{\infty} dt\, A\sin(2\pi f t) e^{-2\pi i k t} \\ &= \frac{A}{2i}\int dk \left[e^{-2\pi i(k+f)t} - e^{-2\pi i(k-f)t}\right] \\ &= \frac{A}{2i}\left[\delta(k-f) - \delta(k+f)\right], \end{aligned} \tag{93}$$

where $\delta(z)$ is the Dirac delta function. All you need to know about the Dirac delta function is that it is a spike at position $z = 0$. Thus, the Fourier transform has two spikes at $k = f$ and $k = -f$.

This simple example illustrates what Fourier transforms do: They have a spike at the oscillation frequencies. We could have very complicated functions or data or whatever. If we Fourier transform it, we will see spikes at the oscillation frequencies. This is true even if a system is oscillating at multiple frequencies.

On a computer, it is usually not straightforward to evaluate the integral in (92). Instead, we approximate the Fourier transform with something called the discreet Fourier transform (DFT). We then use a very famous algorithm called the fast Fourier transform (FFT) to compute the DFT. The FFT is one of the most important algorithms in all of computing. We will use it to determine the oscillation frequencies of our neutron star simulations.

I'll give a brief review of the DFT and then review using Python to compute the FFT.

6.1 DFT

To understand the DFT, let's begin with our simple (continuous) function

$$x(t) = A\sin(2\pi f t). \tag{94}$$

Let's imagine we sample this function N times from $t = 0$ to $t = T - \Delta t$, where $T = N\Delta t$ and $\Delta t = T/N$. We thus sample at times t_i for $i = 0, 1, \ldots, N-1$ to obtain

$$x_i = A\sin(2\pi f t_i), \qquad i = 0, \ldots, N-1. \tag{95}$$

We have the total number of sampled points N, the distance between sampled t values, Δt, the total time over which samples are taken, $T = N\Delta t$, and the sampling frequency $f_t = 1/\Delta t$. The DFT of the points x_i is defined by

$$X_k = \sum_{j=0}^{N-1} x_j e^{-i2\pi kj/N}, \qquad k = 0, \ldots, N-1. \tag{96}$$

Note that we have the same number of X_k as we do x_j, which is that we have N of them. We can interpret the X_k as N samples of the Fourier transform $X(k)$, a continuous function, at frequencies $f_k = k\Delta f$ for $k = 0, 1, \ldots, N-1$.

Now, the exponent of the DFT can be written

$$-i2\pi\frac{kj}{N} = -i2\pi\frac{kj}{T/\Delta t} = -i2\pi\left(\frac{k}{T}\right)(j\Delta t) = -i2\pi\,(k\Delta f)\,(j\Delta t)\,, \tag{97}$$

which shows us the distance between frequency values is given by $\Delta f = 1/T = 1/(N\Delta t)$ and the maximum frequency is given by when k has its maximum value, $f_{\max} = (N-1)\Delta f = (N-1)/(N\Delta t)$. In other words, the total time, T, over which we have samples tells us the precision, Δf, at which we can determine an oscillation frequency. And the sampling step size, Δt, tells us the maximum frequency which we can determine.

The DFT will determine frequencies that go from $-N\Delta f/2 = -1/2\Delta t$ to $1/2\Delta t$. For every positive frequency determined by the DFT, there will be a symmetric negative frequency which has no meaning for us. We just care about the positive frequencies.

The DFT should approximate the continuous Fourier transform:

$$\int_{-\infty}^{\infty} dt\, x(t) e^{-2\pi ikt} \approx \Delta t \sum_{j=1}^{N} x_j e^{-i2\pi kj/N}. \tag{98}$$

Note the Δt in front on the right hand side. This is the correct normalization for the DFT so that it agrees with the continuous Fourier transform. Even so, normalizing the DFT is not that important for us. We only care where the spikes are and we will not really care how tall the spikes are.

6.2 Python

I will give example Python code for computing the FFT of the function

$$x(t) = A\sin(2\pi f t). \tag{99}$$

First define it:

```
x = lambda t: A*sin(2*pi*f*t)
```

where `A` and `f` can be previously defined. Set up the quantities we need:

```
tmin = 0
tmax = 8
N = 200
dt = (tmax - tmin)/N
t = np.arange(tmin, tmax, dt)
```

which gives `N` equally spaced t-values starting from `tmin` up to but not including `tmax` with equal spacing `dt`. Now make the N samples of the function:

```
sampled_pts = x(t)
```

Now for the FFT code:

```
from scipy.fft import fft, fftfreq
FT = fft(sampled_pts)
freqs = fftfreq(N, dt)
```

`FT` are the Fourier transformed points, X_k. For each point, there is a corresponding frequency, which is held in `freqs`. These can be used as the y and x values for plotting:

```
plt.plot(freqs, np.abs(FT))
```

The Fourier transform can, in general, be complex. That's the reason why take the absolute value. You should see a spike at whatever value you have chosen for `f`.

```
tmin = 0
tmax = 8
N = 200
dt = (tmax - tmin)/N
t = np.arange(tmin, tmax, dt)
```

which gives `N` equally spaced t-values starting from `tmin` up to but not including `tmax` with equal spacing `dt`. Now make the N samples of the function:

```
sampled_pts = x(t)
```

Now for the FFT code:

```
from scipy.fft import fft, fftfreq
FT = fft(sampled_pts)
freqs = fftfreq(N, dt)
```

`FT` are the Fourier transformed points, X_k. For each point, there is a corresponding frequency, which is held in `freqs`. These can be used as the y and x values for plotting:

```
plt.plot(freqs, np.abs(FT))
```

The Fourier transform can, in general, be complex. That's the reason why take the absolute value. You should see a spike at whatever value you have chosen for `f`.