1D heat transfer code

I. INTRODUCTION

This document is written with the purpose of giving a reader information of basic structure of 1D heat transfer code as well as instructions of how to use this code. 1D heat transfer code simulates thermal evolution of isolated static neutron stars. All the Physics which is required to build heat capacity, neutrino emissivity, thermal conductivity functions and TiTe relation was copied from O. Gnedin's Neutron Star Evolution code. The code has been written in python programming language.

II. HEAT EQUATION

If we have a static spherically symmetric neutron star (NS) with an isotropic magnetic field, all we need to solve in our numerical computation is the heat equation. Taking into account rotational symmetry, we make temperature T be a function only of two variables: r - the distance from the center of NS and t - time, that is counted form the birth of NS. Therefore, we come up with a following equation:

$$C(T,\rho)\frac{\partial T}{\partial t} = \frac{1}{4\pi r^2} div(4\pi r^2 \kappa(T,\rho)\nabla(T)) - Q(T,\rho). \tag{1}$$

Here C is the heat capacity per unit volume, Q is the neutrino emissivity and κ is the thermal conductity. We make these three functions depend not on r but on ρ instead for computational reasons. It is worth to say, that we can make ρ be a function of r and vice versa.

It is known, that neutron stars are extremely dense objects in the Universe, so one cannot neglet General relativity effects. In this case we introduce metric for a spherically symmetric static star:

$$ds^2 = \left(1 - \frac{2GM(r)}{rc^2}\right) c^2 dt^2 - \left(1 - \frac{2GM(r)}{rc^2}\right)^{-1} dr^2 - r^2\left(d\theta^2 + sin^2\theta d\varphi^2\right).$$

By M(r) function we mean the mass, that is enclosed within a sphere with a radius r.

In such metric equation 1 takes more complicated form:

$$C\frac{\partial\tilde{T}}{\sqrt{1-\frac{2GM}{rc^2}}\partial t} = \frac{1}{4\pi r^2}div\left(4\pi r^2\sqrt{1-\frac{2GM}{rc^2}}\kappa e^{\Phi}\nabla(\tilde{T})\right) - \frac{\tilde{Q}}{\sqrt{1-\frac{2GM}{rc^2}}}, \tag{2}$$

where $\tilde{T} \equiv Te^{\Phi}$ is the redshifted temperature, $\tilde{Q} \equiv Qe^{2\Phi}$ is the redshifted neutrino emissivity, $\Phi \equiv \frac{\phi(r)}{c^2}$ is the dimensionless gravitational potential.

This equation is a nonhomogeneous parabolic equation, so we have to introduce two boundary conditions and one initial condition. At $r = 0$ for radial component of redshifted luminosity vector $\tilde{L}$ we have

$$\tilde{L}_r = -4\pi r^2\sqrt{1-\frac{2GM}{rc^2}}\kappa e^{\Phi}\nabla(\tilde{T}) = 0.$$

At $r = R$, where R is the radius of NS, we obtain

$$\tilde{L}_r = -4\pi r^2\sqrt{1-\frac{2GM}{rc^2}}\kappa e^{\Phi}\nabla(\tilde{T}) = 4\pi R^2\sigma T_e^4(T)e^{2\Phi}, \tag{3}$$

where second term represents photon radiation from the surface of NS and $T_e(T)$ - is the effective surface temperature. Finally, the time condition: at t = 0

$$\tilde{T} = T_0,$$

where T_0 is the initial redshifted temperature of the NS.

There is one more thing left to cover in equation 2 - we need somehow to compute C, $\tilde{Q}$ and κ functions. In this document we are not going to go into details and talk about physics, that is behind those function. If you are a meticulous reader, we encourage you to look for such information in Gnedin's Neutron Star Evolution code.

III. NUMERICAL SCHEME

We use finite difference **backward Euler method** (BEM) to solve equation 2. It is implicit and unconditionally stable, so we are free to choose any time step dt. BEM solution converges as $O(dt)$ in time and $O(dr^2)$) in spatial dimension.

Let's look closer at how BEM solves equation 2.

Suppose, we have a stiff inhomogeneous parabolic equation in the general form

$$a(r,T)T'_t = b(r,T)\left(c(r,T)T'_r\right)'_r - d(r,T), \quad T = T(r,t).$$

We have an 1-dimensional code, so we need to create a mesh by dividing our neutron star into spherical layers with width dr_i. Therefore, we can write

$$\sum_{i=1}^{N} dr_i = R.$$

We discretize t in the same way:

$$\sum_{n=1}^{N'} dt_n = t.$$

We consider the situation, where $dr_i \neq dr_{i+1}$ and $dt_n \neq dt_{n+1}$ can take place. After straightforward steps we get

$$a_i^n \frac{T_i^{n+1} - T_i^n}{dt_{n+\frac{1}{2}}} = b_i^n \left(c_i^n (T_i^{n+1})'_r\right)'_r - d_i^n$$

and

$$a_i^n \frac{T_i^{n+1} - T_i^n}{dt_{n+\frac{1}{2}}} = \frac{b_i^n}{dr_i}\left(c^n_{i+\frac{1}{2}} \frac{T_{i+1}^{n+1} - T_i^{n+1}}{dr_{i+\frac{1}{2}}} - c^n_{i-\frac{1}{2}} \frac{T_i^{n+1} - T_{i-1}^{n+1}}{dr_{i-\frac{1}{2}}}\right) - d_i^n. \tag{4}$$

If one find $i + \frac{1}{2}$ notation bizzare, all we did was create a 1D mesh with n cells ($i = 1, 2, ..., n+1$) and divided each cell into two new cells, so in this case ($i = 1, \frac{3}{2}, 2, \frac{5}{2}, ..., n+1$). Taking such partition, we get more accurate solution. Notice, that $dr_{i+\frac{1}{2}}$ is the interval between dr_{i+1} and dr_i, but $c^n_{i+\frac{1}{2}}$ is function c, computed in the node r_{i+1} at time t^n.

Introducing coefficients

$$A_i = \frac{b_i^n c^n_{i+\frac{1}{2}} dt_{n+\frac{1}{2}}}{dr_{i+\frac{1}{2}} dr_i a_i^n}, \quad B_i = \frac{b_i^n c^n_{i-\frac{1}{2}} dt_{n+\frac{1}{2}}}{dr_{i-\frac{1}{2}} dr_i a_i^n}, \quad D_i = \frac{dt_{n+\frac{1}{2}}}{a_i^n},$$

from equation 4 we get

$$(1 + A_i + B_i)T_i^{n+1} - A_i T_{i+1}^{n+1} - B_i T_{i-1}^{n+1} = T_i^n - D_i d_i^n.$$

The last equation can be represented in matrix-vector form:

$$\begin{bmatrix} (1+A_1+B_1) & -A_1 & 0 & \cdots & 0 \\ -B_2 & (1+A_2+B_2) & -A_2 & \cdots & 0 \\ 0 & -B_3 & (1+A_3+B_3) & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & (1+A_N+B_N) \end{bmatrix} \times \begin{bmatrix} T_1^{n+1} \\ T_2^{n+1} \\ T_3^{n+1} \\ \vdots \\ T_N^{n+1} \end{bmatrix} = \begin{bmatrix} T_1^n - D_1 + \Psi \\ T_2^n - D_2 \\ T_3^n - D_3 \\ \vdots \\ T_N^n - D_N + \Phi \end{bmatrix},$$

where functions Φ and Ψ have appeared due to boudary conditions at the first and the last node in the mesh.

So, knowing temperature T at time t^n, we can find T at time t^{n+1} by solving matrix equation above. Our code uses Thomas algorithm to solve that system with a tridiagonal matrix.

IV. ISOTHERMAL PROFILE ALGORITHM

When the temperature profile of NS does not change shape anymore, it is reasonable to create something faster and rather simpler, than the above mentioned numerical schemes. Therefore, in this code we also place the third numerical scheme - **isothermal profile algorithm** (IPA). If temperature $\tilde{T}$ is constant(r) in equation 2, we can use such scheme. It works faster, than the other two algorythms, because IPA solves not the linear system, but the linear equation, using an explicit Euler scheme. To get formulas for IPA, we need to integrate first and third term in equation 3 over the the total volume of NS. So, considering the above said, we come out with formula

$$C_{total}(T_i)\frac{\partial \tilde{T}_i}{\partial t} = -\tilde{L}(T_e) - \tilde{Q}_{total}(T_i),$$

where T_i is the temperature in the cell, that is closest to the NS surface. The total heat capacity C_{total} and the total neutrino luminosity Q_{total} are defined as follows:

$$C_{total} = \int_0^R C \frac{1}{\sqrt{1-\frac{2GM}{rc^2}}} 4\pi r^2 dr$$

$$\tilde{Q}_{total} = \int_0^R \tilde{Q} \frac{1}{\sqrt{1-\frac{2GM}{rc^2}}} 4\pi r^2 dr$$

The photon luminosity $\tilde{L}$ is just the luminosity at the stellar boundary, is defined by the formula

$$\tilde{L} = \sigma T_e^4(T_i) e^{2\Phi_R},$$

where $\Phi_R = \Phi(R)$.

V. ANALYTICAL TEST

It is important before running the simulation to make sure, that the numerical scheme, which will be used in the main cycle, works fine. In order to do this, we can create an equation which has the same form as the equation 1, but it also has an analytical solution (it is obvious, that we can find such C, Q and κ). Comparing the numerical solution with the analytical one, we can estimate, how accuate our numerical scheme is.

In the code we use following formulas for C, Q and κ:

$$C(T) = C_0 T, \quad \kappa(T) = \kappa_0 T, \quad Q(T) = Q_0 T^2,$$

where C_0, κ_0 and Q_0 are constants. They equal 10^{12}, 10^{12} and 10 respectively, but can be changed, if needed.

The analytical solution has the following form:

$$T(r,t) = T_0 e^{-\gamma t} \left(\frac{sinkr}{kr} \right)^{\frac{1}{2}},$$

where T_0 - the initial temperature of NS, $k = \frac{\pi}{R}$ and

$$\gamma = \frac{\frac{1}{2}\kappa_0 k^2 + Q_0}{C_0}.$$

VI. THE CODE STRUCTURE

The code contains **main.py** file and **13 modules** divided into 5 groups. Proper description for each module can be found on the last two pages of the document.

VII. TIME STEP

We set time step dt by hand in manager.py file. It varies throughout the simulation. We have one restriction on dt: in paragraph III the values on the right hand side of the matrix-vector equation have to be all positive. It is easy to see, that if we increase dt, they will become negative eventually. The code will be terminated, if such thing happens.

VIII. HOW TO LAUNCH THE CODE

First of all, you need to set the initial parameters you want in manager.py. There you can choose NS model, set initial temperature, partition parameters, time, when simulation ends, or temperature, below which the simulation ends, and so on. In that file everything is commented, so it is easy enough to find, what you are looking for.

After you have made the first step, you need to open main.py file. There you firstly use **data_init()** function to initialize all the parameters needed in the simulation and load necessary input data. Then **main()** function is required to start off the simulation. If one would like to visualize the output data, **show_cooling_curves()** function should be used at the end. So, you write

data_init()
main()
show_cooling_curves()

and run main.py.

In order to launch analytical test procedure, one should use **test_starter()** in the same-named file.

If something goes wrong, the code will let you know via terminal.

Have a nice day!

TABLE I: Code files.

computation	**control**	**other**	**data**	**physics**
PDFsolver.py	constants.py	routines.py	loaddata.py	heatcapacity.py
analytical test.py	manager.py	plot.py		neutrino.py
				thermalconductivity.py
				sf_gap.py
				tite.py
				physics.py

TABLE II: Computation

PDFsolver.py	Contains three numerical algorithm, which were mentioned above (KNA, BEM, IPA). Here the mesh is created. Equation 2 is solved here.
analytical test.py	Contains all we have mentioned in paragraph V.

TABLE III: Control

constants.py	Contains physical constants, which are needed in numerical computation.
manager.py	**The code control panel**.

TABLE IV: Other

routines.py	Contains set of routines to make the code be more understandable. Contains Thomas algorithm to solve a system with a tridiagonal matrix.
plot.py	Contains functions, which plot the input data.

TABLE V: Data

loaddata.py	Loads all the input data. Interpolates loaded data to create functions (except for C, $\tilde{Q}$, κ) needed to solve equation 2. loaddata.py requires the following input data files:
effmass.dat	Data file with the effective masses of baryons.
npsf.dat	Data file with the reduction factors for double superfluidity.
model.dat	Data file with the NS model data. We have three different models of NS.
tite.dat	Data file with Ti-Te relation. Firstly it is created in tite.py and is read in loaddata.py afterward.

TABLE VI: Physics

heatcapacity.py	Computes heat capacity.
neutrino.py	Computes neutrino luminosity.
thermalconductivity.py	Computes thermal conductivity.
sf_gap.py	Computes energy gaps for singlet and triplet superfluid states.
tite.py	Creates Ti-Te relation for a spesific NS model, magnetic field and mass of accreted envelope.
physics.py	Set of initialization routines. Creates C, $\tilde{Q}$, κ to be functions only of density ρ and temperature T for a specific NS model. Can write out tables with such relations for C, $\tilde{Q}$, κ.