Effects of Time-Dependent Heat Sources On Neutron Star Crust Cooling

A. Smith* and H. M. Doss
Physics and Engineering Department, Point Loma Nazarene University
(2019-2020 PLNU Honors Program)
(Dated: April 30, 2020)

Neutron star crust cooling can give insights to the interior composition of the stars. Using dStar and the neutron star cooling simulation code NSCool, a variable mass is accreted onto different neutron star models. Previous accretion simulations have commonly assumed a constant mass accretion over a long epoch; this research investigates the effects of different accreting mass accretion rate distributions and focuses on a periodic Gaussian distribution, with a finer time epoch. The simulations produce plots of mass accretion rate distributions as well as effective temperature and luminosity as viewed by a distant observer over time. The effects on the quiescent cooling curves due to the time-dependent distribution shapes of accreted mass are found to have only short time-scale effects, with differences only noticeable on timescales as on the order of the accretion events.

I. INTRODUCTION

Since the theorization of neutron stars (NSs) in 1933, and their discovery in 1967 [1], NSs have played a key part in understanding how the universe works, as they are natural laboratories to study matter at extreme densities. Specifically, NSs are theorized to be responsible for some of the origins of heavy nuclei, and in recent years the LIGO and VIRGO spectrometers have seen NS mergers, providing new opportunities to understand the interior of NSs [2]. However, binary NSs only represent a fraction of all NSs. NSs with a companion star, that is not a NS, provide their own opportunities for new knowledge. Of specific interest in this research is the accretion of matter from the companion star onto the surface of the NS and how the star cools after the accretion period ends. Computer programs, such as MESA (Modules for Experiments in Stellar Astrophysics), dStar, and NSCool, combined with observational data have provided some understanding of the processes occurring in the crust of a NS.

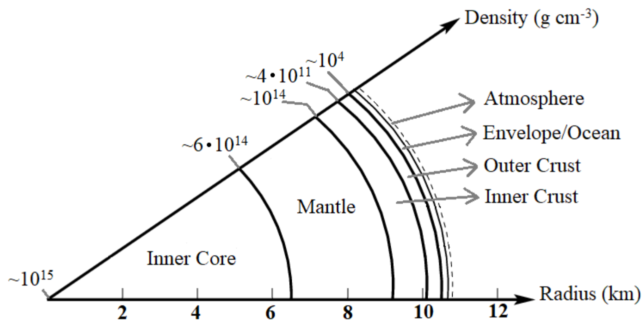

FIG. 1. A cross section of a NS

A NS is the corpse of a massive main sequence star that has gone supernova. A NS supports its mass by neutron degeneracy pressure and nucleon-nucleon interactions, as opposed to thermal pressure in a standard star. Degeneracy pressure arises from the Pauli exclusion principle where two identical half-integer spin particles, known as fermions, are prohibited from occupying the same quantum state. The degeneracy is due to the compactness of the object, and the degeneracy pressure is high enough to support the star against gravitational collapse. NSs are extremely compact with, on average, 1.4 $M_\odot$ in a sphere of radius 11 km. The star is not uniformly made of pure neutrons however, it has several layers as shown in Fig. 1. The star has a very thin atmosphere with lighter atoms extending not more than a meter above its surface. Below the surface is a very thin envelope, sometimes called an ocean composed of a plasma of electrons and nuclei. The outer crust is composed of a dense crystalline lattice embedded in a gas of degenerate electrons. The nuclei in the lattice may also have more neutrons present in the nucleus, creating heavier neutron-rich atomic nuclei which are stabilized from decaying by the high density. Meanwhile, in the inner crust heavier and heavier neutron-rich atomic nuclei are formed, with some free neutrons in the mix. The transition from the inner crust to the mantel is characterized by what is known as nuclear pasta, where nuclei are compressed so much that they form "stringy," pasta-like structures.

Many different theories try to explain the core of a NS. Quark stars or hybrid neutron-quark stars are theorized to occur when the dense environment of a neutron star allows for quarks to deconfine themselves from neutrons and form quark matter. In hybrid stars, the core of the star is theorized to be made of this quark matter while the mantel is made of nuclear matter. Furthermore, it has been theorized that strange quark matter could be present inside NSs. These stars, called strange stars, occur when quark matter transforms into strange matter, where the quark turns into equal parts up, down, and strange quarks. Other baryonic matter may be present inside NSs, such as hyperons, pions, or kaons [1].

Two ways to investigate the makeup of the cores of NSs are either through observations of collisions or

* asmith1493@pointloma.edu

through the investigation of the heating and cooling of NSs after outbursts. Heat can be transferred to the interior of a NS so that the NS remains incandescent for a while after accretion ends and quiescence begins; this serves as a thermometer for the inner layers of a NS [3]. It also provides a window to processes in the core of the star and may provide clues about its makeup. One group has used a program to model multiple outbursts of Aql X-1 [4]. They found that Aql X-1 does not reach crust-core thermal equilibrium and does not reach the base level temperature between outbursts. They also found that they were able to closely reproduce data if they fit the data with a model that varied envelope composition and heating parameters.

Accretion heats up a NS. Matter is pulled off a companion star by the strong gravitational attraction to the NS and eventually lands on the surface of the NS. The fresh matter is compressed by the strong gravitational interaction to the point of pyconuclear fusion in the inner crust, which releases about 2 MeV/u. Two-step electron captures will create local heat sources as well; both these sources of heating occur only during active accretion [5].

The NS then cools by means of electromagnetic radiation, primarily in X-rays; heat is conducted to the surface predominately by degenerate electrons and is set by electron-ion scattering [5]. This allows the heat from the inner crust to escape via photons at the surface of the NS. A NS can also cool by means of Urca cooling. This is a cyclic electron-capture and β^--decay, which produces neutrinos that escape the NS and thus radiate heat away. This cycle is active during and after accretion and is not one-way like the electron-capture heating [5].

Observations of NSs provide simulations with some realistic parameters to base simulations on. The NS Aql X-1 has garnered particular interest due to its frequent outbursts followed by period of quiescence and cooling. Data has been collected by the *Rossi X-ray Timing Explorer (RXTE)*, *Neil Gehrels Swift Observatory (Swift)*, and *Monitor of All-Sky X-ray Image (MAXI)*.

Sending a probe to these stars is not an option due to the incredibly far distances to NSs, hence computer simulations are the best way to test and predict properties of NSs. The basis for several stellar simulation programs is MESA, which has been developed by a large international collaboration [6–10]. The MESA EOS is a blend of the OPAL [11], SCVH [12], PTEH [13], HELM [14], and PC [15] EOSes. Radiative opacities are primarily from OPAL [16, 17], with low-temperature data from [18] and the high-temperature, Compton-scattering dominated regime by [19]. Electron conduction opacities are from [20]. Nuclear reaction rates are a combination of rates from NACRE [21], JINA REACLIB [22], plus additional tabulated weak reaction rates [23–25]. Screening is included via the prescription of [26]. Thermal neutrino loss rates are from [27]. MESA can be used to evolve a selection of standard stars, as well as simulate accretion events onto a NS and simulate explosive nucleosynthesis from supernova events. MESA serves as the foundation for dStar [28], a separate and more specific simulation program simulating properties of NSs. The program dStar has been developed primarily by Dr. Edward Brown at Michigan State University. Inside dStar is NSCool, which models the cooling of the crust of the NS, as well as many sample routines to understand how to input simulations to run.

This work explores simulated luminosities of cooling NSs when differently shaped mass distribution are accreted onto the NS; the distinguishability of the luminosities are examined considering the thermal diffusion timescale.

II. SIMULATION METHODS

Using MESA version 12115 and its SDK [29], dStar, and python 3.7.3, we developed a python wrapper program (Appendix A) that generates Gaussian shaped mass accretion rate distributions. These distributions accrete onto a 1.6 $M_\odot$ NS with a core radius of 10.42 km. NSCool outputs “observed” temperatures from infinity. In order to compare these theoretical values with real observed luminosity, the Stefan-Boltzmann law is used

$$L_{\rm ph}^{\infty} = 4\pi\sigma_{\rm SB}R_{\infty}^{2}(T_{\rm eff}^{\infty})^{4}, \quad (1)$$

where $L_{\rm ph}^{\infty}$ is the luminosity observed by a distant observer, $\sigma_{\rm SB}$ is the Stephan-Boltzmann constant, $T_{\rm eff}^{\infty}$ is the effective temperature observed by a distant observer. R_∞ is the radius of the NS according to a distant observer

$$R_\infty = R(1 + z_{\rm g}), \quad (2)$$

$$(1 + z_{\rm g}) = \frac{1}{\sqrt{1 - \frac{2GM}{Rc^2}}}, \quad (3)$$

where M is the mass of the NS, G is the universal gravitational constant, R is the radius of the NS, and c is the speed of light.

The inlist file controls many of the parameters that affect the heating and cooling of a NS. Since the primary focus of this research is how differences in time-dependent mass accretion affects the cooling curve, these parameters were not changed. However, these likely also affect the cooling curve. In this research, the NS core temperature is fixed, at 3.25×10^7 K, while the atmosphere and crust temperatures are not fixed. Sixty-four epochs were used in total, with 45 epochs occurring in the time frame −90 to 0 days where mass accretion would be inputted. Other routines were set to the values seen in Fig. 2. These include a factor for heating caused by neutrinos that originate from the decay of pions [30], which were created during the impact of material, thermal conductivity factors in the nuclear pasta layer of a NS set (set by the impurity parameter $Q_{\rm imp}$ in the pasta layer), pressure boundary conditions, and an atmospheric light element composition factor [30]. $Q_{\rm imp}$ is set to 1 and

```
! other routines
use_other_set_Qimp = .TRUE.
use_other_set_heating = .TRUE.
! extra controls for hook routines
! defined here
! 1. extra heating from pion -> neutrino
! 2. Q in the pasta
! 3.-4. density limits for extra heating
extra_real_controls = 4.0,80.0,1.0e12,1.0e13

! core properties
core_mass = 1.6     ! Msun
core_radius = 10.42    ! km

! crust boundaries (pressure)
eos_pasta_transition_in_fm3 = 0.05
lgPcrust_bot = 33.0 ! cgs
lgPcrust_top = 26.0 ! cgs

! heating
turn_on_extra_heating = .TRUE.
Q_heating_shallow = 1.0
lgP_min_heating_shallow = 27.0
lgP_max_heating_shallow = 28.0

! shell Urca cooling
turn_on_shell_Urca = .FALSE.

which_neutron_1S0_gap = 'gipsf08'

! atmosphere composition
lg_atm_light_element_column = 9.0

! impurities
fix_Qimp = .TRUE.
Qimp = 1.0

turn_on_shell_Urca = .FALSE.
```

FIG. 2. The other inputs and controls in the inlist file, all of which affect the heating and cooling of NSs. These values were held constant throughout the simulations.

arises from impurities causing additional electron scattering which inhibits thermal diffusion and therefore sets the thermal gradient in the crust at low temperatures [5]. Urca cooling factors are also included here, though they are turned off.

The python program generates multiple randomized distributions and passes these values into the inlist file. The Gaussian distributions are formed with the skewnorm.rvs() python function, with a sample size of fifty. One hundred of these randomly generated Gaussians are accreted to obtain uncertainties. It then retrieves the temperature output from dStar, calculates the luminosity, and averages these values. A plot is produced displaying the mass accretion rate distribution, temperature, and luminosity with their corresponding errors. The luminosity values are also saved so that the effects of the different distributions can be statistically compared.

In order to determine if the mass accretion rate distributions cause a difference, the thermal diffusion timescale also needs to be calculated. dStar periodically saves outer crust data which can be used to calculate the diffusion time using the following equation

$$\tau = \frac{1}{4}\left[\int_0^P \left(\frac{\rho C_p}{K}\right)^{1/2} \frac{\mathrm{d}P}{\rho g}\right]^2, \tag{4}$$

where ρ is the mass density, C_p is the specific heat per unit mass, K is the thermal conductivity, P is the pressure, and g is the gravity [31].

III. SIMULATION RESULTS

The Gaussian distributions are centered at -45 days, and have the same total amount of mass accreted. The luminosity values with their standard deviations are compared with a two sample t-test, with the null hypothesis that the luminosity values between two different mass accretion rate distributions are different and the alternative hypothesis that they are the same. If the p-value, or probability that alternative hypothesis is true, is above 0.01, then the null hypothesis is rejected in favor of the alternative hypothesis. This statistical test provides insight to the day that the difference between "observed" luminosities are indistinguishable when compared between the different initial mass accretion rate distributions.

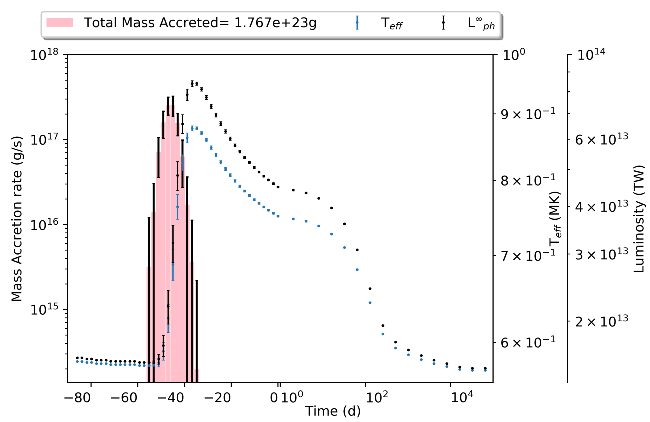

FIG. 3. A single Gaussian mass accretion rate distribution accreted with mean at $\mu = -45$ days and a standard deviation of $\sigma = 3$ days

Fig. 3 shows a single accretion event centered at -45 days, the total mass being accreted is 1.767×10^{23} g, where as Fig. 4 shows this mass equally split between two accretion events still being centered at -45 days, with the too modes having a separation of 45 days. All of the distributions have a spread of 3. As seen by the p-values in Table I, between 64 and 128 days, the observed luminosities between the accretion distributions shown in Fig. 3 and Fig. 4 become indistinguishable from one another.

Fig. 5 shows a single accretion event centered at -45 days, the total mass being accreted is 1.767×10^{23} g,

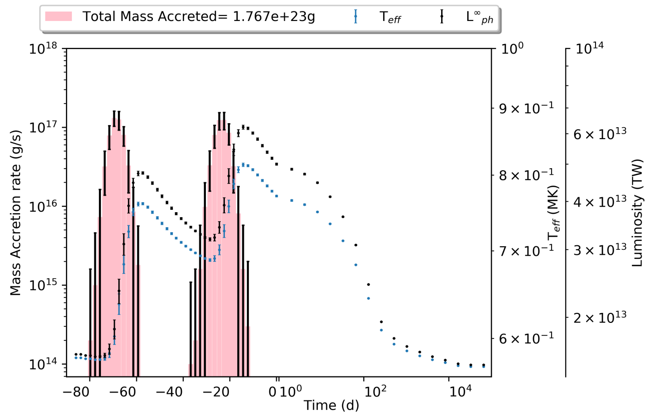

FIG. 4. Two Gaussian mass accretion rate distributions accreted($\mu = -67.5, 22.5, \sigma = 3$)

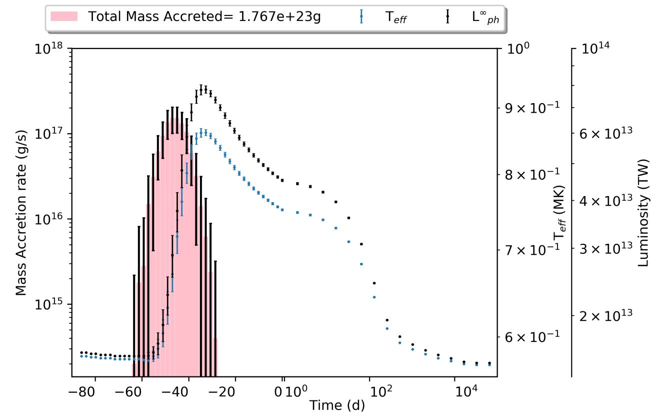

FIG. 5. Single Gaussian mass accretion rate distribution accreted ($\mu = -45, \sigma = 5$)

Days	Fig. 3 - Fig. 4 p-values	Fig. 5 - Fig. 6 p-values	Fig. 7 - Fig. 8 p-values
0	0	0	0
1	0	0	0
2	0	0	0
4	0	0	0
8	0	0	0
16	0	0	0
32	0	0.005	0
64	0	0.003	0
128	0.012	0.146	1
256	1	0.155	1
512	1	1	1
1024	1	1	1
2048	1	1	1
4096	1	1	0.312
8192	1	1	1
16384	1	1	1
32768	1	1	1
65536	1	1	1

TABLE I. Days with p-values for the differences between luminosities from Figs. 3-8, comparing the results out to temperatures and luminosities observed before the accretion outburst.

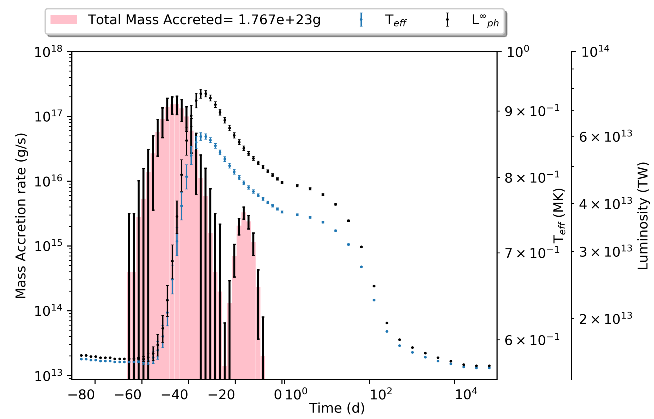

FIG. 6. Gaussian mass accretion rate distribution accreted ($\mu = -45, \sigma = 5$) with a small trailing accretion event ($\mu = -15, \sigma = 2.5$)

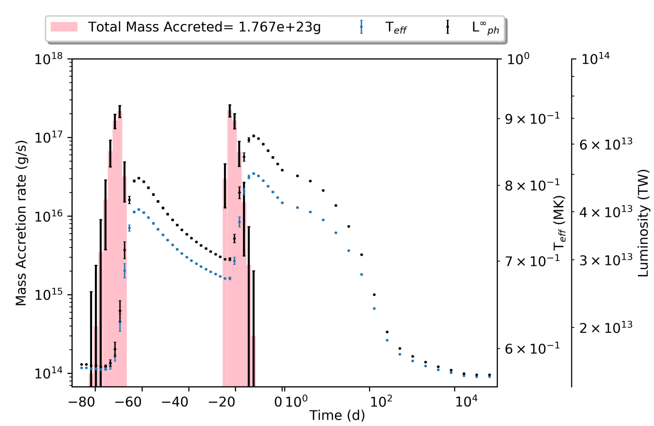

FIG. 7. Two skewed Gaussian mass accretion rate distributions accreted, with the skews facing outward

whereas Fig. 6 shows this mass split between two accretion events. One is the main event, with a mass of 1.749×10^{23} g, and a trailing, smaller event with a mass of 1.767×10^{21} g. The center of the main event is at -45 days, with a spread of 5, and the trailing event has a center of -15 days, and a spread of 2.5. As seen by the p-values in Table I, between 64 and 128 days, the observed luminosities between the accretion distributions shown in Fig. 5 and Fig. 6 become indistinguishable from one another.

Fig. 7 shows two skewed Gaussian accretion events at centered at -45 days. The distributions have a spread of 5 and a skew factor of 5, with the tails facing towards the beginning and end of the accretion event. Fig. 8 also shows two skewed Gaussian accretion events at centered at -45 days. The distributions also have a spread of 5

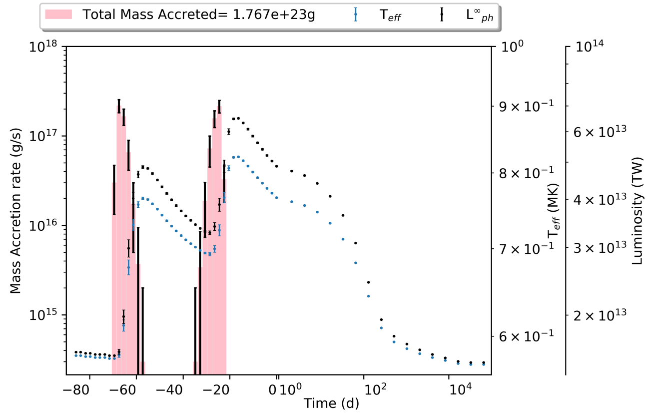

FIG. 8. Two skewed Gaussian mass accretion rate distributions accreted, with the skews facing inward

and a skew factor of 5, although the tails facing towards the center of the accretion event. As seen by the p-values in Table I, between day 64 and day 128, the observed luminosities between the accretion distributions shown in Fig. 7 and Fig. 8 become indistinguishable from one another.

The thermal diffusion timescale, Eq. (4), is around 114 days for the simulated star. This corresponds with the order of magnitude that the luminosity curves become indistinguishable.

IV. SUMMARY

Throughout these simulations the only variable that is changed is the shape of the mass accretion rate distribution within a period of 90 days. Overall, these results suggest that time-dependent mass accretion has only brief effects on cooling curves and that these effects become indistinguishable observationally after the thermal diffusion timescale. Other factors could affect the responsivity of quiescent cooling based on time-dependent mass accretions, but for these moderate constraints somewhere between 109 and 173 days after the center of accretion, all the distributions investigated are statistically identical. When investigating long term cooling of NSs, these results suggest that only the total amount of mass accreted has an effect on observed luminosities long after the accretion event, not the shape or distribution of the mass within a given duration. Consistent measurements of luminosity closer to and during the accretion could provide enough information to reconstruct the accretion distributions. These findings suggest that for NSs with periodic accretions, such as Aql X-1, the mass accretion distribution shape does not play a significant role in the luminosity after a short time into the quiescence period. Prior to quiescence, between close outbursts, and for a short time into the quiescence period, these shapes play a role in simulated luminosities.

There is also the question about how precise the cooling curve simulations are. Changing the number of simulation runs affects the point at which values become significant. Small step sizes in the positive time section of the cooling period also have an affect on the p-values, creating fluctuating significance. With larger step sizes though, this uncertainty, possibly due to boundary conditions within the NSCool simulation, have little effect.

These conclusions are only based on a small selection of accretion shapes. The next step will be to vary the size and separation between the main event and the trailing event to get a clear picture if small, trailing accretion outbursts could cause observable effects on the initial conditions of the next larger outburst.

[1] N. K. Glendenning, *Compact Stars*, 2nd ed., Astronomy and Astrophysics Library (Springer, 2000).

[2] B. P. Abbott and et al. (LIGO Scientific Collaboration and Virgo Collaboration), Gw170817: Observation of gravitational waves from a binary neutron star inspiral, Phys. Rev. Lett. **119**, 161101 (2017).

[3] E. Brown, L. Bildsten, and R. Rutledge, Crustal heating and quiescent emission from transiently accreting neutron stars, The Astrophysical Journal **505**, L95 (1998).

[4] L. S. Ootes, R. Wijnands, D. Page, and N. Degenaar, A cooling neutron star crust after recurrent outbursts: modelling the accretion outburst history of aql x-1, Monthly Notices of the Royal Astronomical Society **477**, 2900–2916 (2018).

[5] Z. Meisel, A. Deibel, L. Keek, P. Shternin, and J. Elfritz, Nuclear physics of the outer layers of accreting neutron stars, Journal of Physics G: Nuclear and Particle Physics **45**, 093001 (2018).

[6] B. Paxton, L. Bildsten, A. Dotter, F. Herwig, P. Lesaffre, and F. Timmes, Modules for Experiments in Stellar Astrophysics (MESA), **192**, 3 (2011), arXiv:1009.1622 [astro-ph.SR].

[7] B. Paxton, M. Cantiello, P. Arras, L. Bildsten, E. F. Brown, A. Dotter, C. Mankovich, M. H. Montgomery, D. Stello, F. X. Timmes, and R. Townsend, Modules for Experiments in Stellar Astrophysics (MESA): Planets, Oscillations, Rotation, and Massive Stars, **208**, 4 (2013), arXiv:1301.0319 [astro-ph.SR].

[8] B. Paxton, P. Marchant, J. Schwab, E. B. Bauer, L. Bildsten, M. Cantiello, L. Dessart, R. Farmer, H. Hu, N. Langer, R. H. D. Townsend, D. M. Townsley, and F. X. Timmes, Modules for Experiments in Stellar Astrophysics (MESA): Binaries, Pulsations, and Explosions, **220**, 15 (2015), arXiv:1506.03146 [astro-ph.SR].

[9] B. Paxton, J. Schwab, E. B. Bauer, L. Bildsten, S. Blinnikov, P. Duffell, R. Farmer, J. A. Goldberg, P. Marchant, E. Sorokina, A. Thoul, R. H. D. Townsend, and F. X. Timmes, Modules for Experiments in Stellar Astrophysics (MESA): Convective Boundaries, Element Diffusion, and Massive Star Explosions, **234**, 34 (2018),

arXiv:1710.08424 [astro-ph.SR].

[10] B. Paxton, R. Smolec, J. Schwab, A. Gautschy, L. Bildsten, M. Cantiello, A. Dotter, R. Farmer, J. A. Goldberg, A. S. Jermyn, S. M. Kanbur, P. Marchant, A. Thoul, R. H. D. Townsend, W. M. Wolf, M. Zhang, and F. X. Timmes, Modules for Experiments in Stellar Astrophysics (MESA): Pulsating Variable Stars, Rotation, Convective Boundaries, and Energy Conservation, **243**, 10 (2019), arXiv:1903.01426 [astro-ph.SR].

[11] F. J. Rogers and A. Nayfonov, Updated and Expanded OPAL Equation-of-State Tables: Implications for Helioseismology, Astrophys. J. **576**, 1064 (2002).

[12] D. Saumon, G. Chabrier, and H. M. van Horn, An Equation of State for Low-Mass Stars and Giant Planets, **99**, 713 (1995).

[13] O. R. Pols, C. A. Tout, P. P. Eggleton, and Z. Han, Approximate input physics for stellar modelling, **274**, 964 (1995), astro-ph/9504025.

[14] F. X. Timmes and F. D. Swesty, The Accuracy, Consistency, and Speed of an Electron-Positron Equation of State Based on Table Interpolation of the Helmholtz Free Energy, **126**, 501 (2000).

[15] A. Y. Potekhin and G. Chabrier, Thermodynamic Functions of Dense Plasmas: Analytic Approximations for Astrophysical Applications, Contributions to Plasma Physics **50**, 82 (2010), arXiv:1001.0690 [physics.plasm-ph].

[16] C. A. Iglesias and F. J. Rogers, Radiative opacities for carbon- and oxygen-rich mixtures, Astrophys. J. **412**, 752 (1993).

[17] C. A. Iglesias and F. J. Rogers, Updated Opal Opacities, Astrophys. J. **464**, 943 (1996).

[18] J. W. Ferguson, D. R. Alexander, F. Allard, T. Barman, J. G. Bodnarik, P. H. Hauschildt, A. Heffner-Wong, and A. Tamanai, Low-Temperature Opacities, Astrophys. J. **623**, 585 (2005), astro-ph/0502045.

[19] J. R. Buchler and W. R. Yueh, Compton scattering opacities in a partially degenerate electron plasma at high temperatures, Astrophys. J. **210**, 440 (1976).

[20] S. Cassisi, A. Y. Potekhin, A. Pietrinferni, M. Catelan, and M. Salaris, Updated Electron-Conduction Opacities: The Impact on Low-Mass Stellar Models, Astrophys. J. **661**, 1094 (2007), astro-ph/0703011.

[21] C. Angulo, M. Arnould, M. Rayet, P. Descouvemont, D. Baye, C. Leclercq-Willain, A. Coc, S. Barhoumi, P. Aguer, C. Rolfs, R. Kunz, J. W. Hammer, A. Mayer, T. Paradellis, S. Kossionides, C. Chronidou, K. Spyrou, S. degl'Innocenti, G. Fiorentini, B. Ricci, S. Zavatarelli, C. Providencia, H. Wolters, J. Soares, C. Grama, J. Rahighi, A. Shotter, and M. Lamehi Rachti, A compilation of charged-particle induced thermonuclear reaction rates, Nuclear Physics A **656**, 3 (1999).

[22] R. H. Cyburt, A. M. Amthor, R. Ferguson, Z. Meisel, K. Smith, S. Warren, A. Heger, R. D. Hoffman, T. Rauscher, A. Sakharuk, H. Schatz, F. K. Thielemann, and M. Wiescher, The JINA REACLIB Database: Its Recent Updates and Impact on Type-I X-ray Bursts, **189**, 240 (2010).

[23] G. M. Fuller, W. A. Fowler, and M. J. Newman, Stellar weak interaction rates for intermediate-mass nuclei. IV - Interpolation procedures for rapidly varying lepton capture rates using effective log (ft)-values, Astrophys. J. **293**, 1 (1985).

[24] T. Oda, M. Hino, K. Muto, M. Takahara, and K. Sato, Rate Tables for the Weak Processes of sd-Shell Nuclei in Stellar Matter, Atomic Data and Nuclear Data Tables **56**, 231 (1994).

[25] K. Langanke and G. Martínez-Pinedo, Shell-model calculations of stellar weak interaction rates: II. Weak rates for nuclei in the mass range /A=45-65 in supernovae environments, Nuclear Physics A **673**, 481 (2000), nucl-th/0001018.

[26] A. I. Chugunov, H. E. Dewitt, and D. G. Yakovlev, Coulomb tunneling for fusion reactions in dense matter: Path integral MonteCarlo versus mean field, Phys. Rev. D **76**, 025028 (2007), arXiv:0707.3500.

[27] N. Itoh, H. Hayashi, A. Nishikawa, and Y. Kohyama, Neutrino Energy Loss in Stellar Interiors. VII. Pair, Photo-, Plasma, Bremsstrahlung, and Recombination Neutrino Processes, **102**, 411 (1996).

[28] E. F. Brown, dStar: Neutron star thermal evolution code (2015), ascl:1505.034.

[29] R. Townsend, Mesa sdk for mac os, Version: 20190503 `http://doi.org/10.5281/zenodo.2669543` (2019).

[30] F. J. Fattoyev, E. F. Brown, A. Cumming, A. Deibel, C. J. Horowitz, B.-A. Li, and Z. Lin, Deep crustal heating by neutrinos from the surface of accreting neutron stars, Phys. Rev. C **98**, 025801 (2018), arXiv:1710.10367 [astro-ph.HE].

[31] L. Henyey and J. L. L'Ecuyer, Studies in stellar evolution. viii. the time scale for the diffusion of energy in the stellar interior, The Astrophysical Journal **156**, 549 (1969).

Appendix A: Python Program

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
@author: austinsmith
DStar/NSCcool Wrapper program that generates and formats mass distributions for the inlist file

"""

import matplotlib.pyplot as plt
import numpy as np
import os
from scipy.stats import skewnorm


```

```
"""
--------------------------------------------------------------------------------------------------------
function that produces mass distributions
--------------------------------------------------------------------------------------------------------
"""
def MandT(mu,sig,skw,multimodal,modes,uniform,trailer,start,end,steps):
    #uniform distibution
    if uniform == True:

        #creating data
        x = np.random.uniform(start,end,50)
        bins=np.linspace(start,end,steps)
        values, bins, _ = plt.hist(x,bins, density=True)
        plt.clf()
        area=np.diff(bins)*values
        area=area*(10**18)

        """
        for i in range(0,len(area)):
            a=area[i]
            if a<(10**10):
                area[i]=0
        """

        #formatting values for entry
        Mdot=format(area[0],".3e")
        for i in range(1,len(area)):
            Mdot=Mdot+","+str(format(area[i],".3e"))
        Mdot=Mdot.replace("+","")
        Mdot=Mdot.replace("0.000e00","0.0")
        Mdot=Mdot+","+str(65-steps)+"*0.0"

        Tbounds=str(bins[0])
        for i in range(1,steps):
            Tbounds=Tbounds+","+str(bins[i])
        for i in range(1,(65-steps)+1):
            Tbounds=Tbounds+","+str(105+i)
    else:
        #standard single skewed
        if multimodal==False:

            #creating data
            x = sig*skewnorm.rvs(skw,size=50)+mu
            bins=np.linspace(start,end,steps)
            values, bins, _ = plt.hist(x,bins, density=True)
            plt.clf()
            area=np.diff(bins)*values


            if trailer==True:
                x = (sig/2)*skewnorm.rvs(skw,size=50)+(mu+6*sig)
                bins=np.linspace(start,end,steps)
                values, bins, _ = plt.hist(x,bins, density=True)
                plt.clf()
                area51=np.diff(bins)*values
                area=area*(10**18-10**16)
                area=area+area51*(10**16)
            else:
                area=area*(10**18)
            """
            for i in range(0,len(area)):
                a=area[i]
                if a<(10**10):
                    area[i]=0
            """

            #formatting
            Mdot=format(area[0],".3e")
```

```
            for i in range(1,len(area)):
                Mdot=Mdot+","+str(format(area[i],".3e"))
            Mdot=Mdot.replace("+","")
            Mdot=Mdot.replace("0.000e00","0.0")
            Mdot=Mdot+","+str(65-steps)+"*0.0"

            Tbounds=str(bins[0])
            for i in range(1,steps):
                Tbounds=Tbounds+","+str(bins[i])
            for i in range(1,(65-steps)+1):
                Tbounds=Tbounds+","+str(105+i)

        #Multimodal set up
        if multimodal==True:
            Tarea=[0]*(steps-1)
            for j in range(modes,0,-2):

                #creating data
                x = 0
                y = 0
                x = ((sig*skewnorm.rvs(skw,size=50))+(mu+((j-1)*mu)/modes))
                y = ((sig*skewnorm.rvs(-skw,size=50))+(mu-((j-1)*mu)/modes))
                bins=np.linspace(start,end,steps)

                xvalues, bins, _ = plt.hist(x,bins, density=True)
                plt.clf()
                yvalues, bins, _ = plt.hist(y,bins, density=True)
                plt.clf()

                xarea=np.diff(bins)*xvalues
                yarea=np.diff(bins)*yvalues



                xarea=(xarea*(10**18))/modes


                yarea=(yarea*(10**18))/modes

                for t in range(0,len(xarea)):
                    xa=xarea[t]
                    """
                    if xa<(10**10):
                       xarea[t]=0
                    """
                    if j !=1:
                        Tarea[t]=Tarea[t]+xarea[t]

                for u in range(0,len(yarea)):
                    ya=yarea[u]
                    """
                    if ya<(10**10):
                       yarea[u]=0
                    """
                    Tarea[u]=Tarea[u]+yarea[u]




            #formatting
            Mdot=format(Tarea[0],".3e")
            for i in range(1,len(Tarea)):
                Mdot=Mdot+","+str(format(Tarea[i],".3e"))
            Mdot=Mdot.replace("+","")
            Mdot=Mdot.replace("0.000e00","0.0")
            Mdot=Mdot+","+str(65-steps)+"*0.0"

            Tbounds=str(bins[0])
            for i in range(1,steps):
                Tbounds=Tbounds+","+str(bins[i])
```

```
            for i in range(1,(65-steps)+1):
                Tbounds=Tbounds+","+str(105+i)
    return(Mdot,Tbounds)

"""
----------------------------------------------------------------------------------------------------------------------
constants and some calculations for luminocities
----------------------------------------------------------------------------------------------------------------------
"""
MassSun=1.9891*10**30   #kilograms
MassNS=1.6*MassSun

R=10.42*10**3   #radius in meters

G=6.67408*10**-11   #m3 kg-1 s-2
sigSB=5.670374*10**-8 #Wm^-2K^-4
c=299792458   #m/s

Rinf=R*1/(np.sqrt(1-2*G*MassNS/(R*c**2)))

"""
----------------------------------------------------------------------------------------------------------------------
main routine for calculating and plotting
----------------------------------------------------------------------------------------------------------------------
"""
multi=False #multiple modes or not
unif=False  #uniform or not
cntr=-45    #center
sprd=5      #spread
skw=0      #skew
numModes=2  #number of modes for multimodal
start=-90   #starting epoch for mass accretion
end=0       #ending epoch for mass accretion
steps=45    #number of steps
smpl=100     #number of samples
trailer=False

tauPerRun=[]

for i in range(0,smpl):



    #(center,spread,skew,multimodal?,number of modes,?,start,stop,steps)
    #-----------------------------------------
    Mdot,Tbounds=MandT(cntr,sprd,skw,multi,numModes,unif,trailer,start,end,steps)
    #-----------------------------------------

    #inputting the mass and time bounds to inlist file
    with open('inlist','r') as file:
        input=file.readlines()
        input[30]=str("    epoch_Mdots = "+Mdot+"\n")
        input[31]=str("    epoch_boundaries = "+Tbounds+"\n")
    with open('inlist','w') as file:
        file.writelines(input)

    #running dStar
    os.system("./run_dStar -D/Users/austinsmith/Documents/dStar > Output.txt")

    #extracting Teff from output
    lookup=str("-----------")


    Time=[]
    Teff=[]
    Linf=[]
```

```
    num=0
    line=0
    b=0

    with open('Output.txt','r') as file:
        data=file.readlines()
    with open('Output.txt','r') as file:
        for num,line in enumerate(file,0):
            if lookup in line:
                index=num


    for b in range (index+1,index+60):
        dLine=data[b].split()
        Time.append(float(dLine[0]))
        Teff.append(float(dLine[1]))

    #creating mdot array
    Mdot=Mdot.split(',')
    for y in range(0,len(Mdot)-1):
        Mdot[y]=float(Mdot[y])
    Mdot[len(Mdot)-1]=0.0

    #creating Time arrays
    Tbounds=Tbounds.split(',')
    for y in range(0,len(Tbounds)):
        Tbounds[y]=float(Tbounds[y])

    TimeM=[]
    for yy in range(0,len(Tbounds)):
        if Tbounds[yy] <= 0.0:
            TimeM.append(Tbounds[yy])


    #luminocity array
    for k in range(0,len(Teff)):
        Linf.append(4*np.pi*sigSB*(Rinf**2)*(Teff[k]**4))


    #converting from array to np array
    if i==0:
        TotalM=np.array([Mdot])
        TotalT=np.array([Teff])
        TotalL=np.array([Linf])

    if i>0:
        TotalM=np.append(TotalM,[Mdot],axis=0)
        TotalT=np.append(TotalT,[Teff],axis=0)
        TotalL=np.append(TotalL,[Linf],axis=0)

    """
    ----------------------------------------------------------------------------------------------------------------------
    Finding saved values to calculate diffusion
    ----------------------------------------------------------------------------------------------------------------------
    This is all very very ineffeicient, as far as for loops go,
    but this could be easily cleaned up or expanded to look at
    all zones and profiles for a given run of dStar.
    """


    #reading the output to see how many models/profiles to read in
    modelF=int((data[index-4].split())[0])
    models=[]
    for b in range(1,modelF+1): #profile--------------------

        #opening profiles
```

```python
        profile=str('LOGS/profile'+str(b))
        with open(profile,'r') as file:
            data=file.readlines()
            length = len(data) #how many rows are in a profile

        #finding the time of the model
        modelTime=float((data[4].split())[1])

        #initializing arrays
        P=[]
        rho=[]
        g=[]
        Cp=[]
        K=[]



        #if the model time is around day 0
        if modelTime > -1 and modelTime < 1:
            for d in range(9,length): #ZONE---------------------

                #creating arrays of the data from the zone
                zonedata=data[d].split()

                P.append(float(zonedata[9]))
                rho.append(float(zonedata[10]))
                g.append(float(zonedata[4]))
                Cp.append(float(zonedata[16]))
                K.append(float(zonedata[19]))

            #converting to numpy arrays
            npP=np.array([P])
            npRho=np.array([rho])
            npG=np.array([g])
            npCp=np.array([Cp])
            npK=np.array([K])



            #making dP array
            dP = np.zeros_like(npP)
            dP[0][0] = npP[0][0]

            #finding dP values
            for tt in range(1,dP.shape[1]):
                dP[0][tt] = npP[0][tt]-npP[0][tt-1]


            #calcualting the integrand
            integrand = np.sqrt(npRho*npCp/npK)/(npRho*npG)

            #making tau array
            tau = np.zeros_like(integrand)
            # tau is in units of seconds
            # divide by 86400 to get it in units of days

            lengthT=tau.shape[1]


            #computing tau for every zone
            for ii in range(1,lengthT):
                tau[0][ii] = 0.25*((integrand[0][ii]*dP[0][ii]))**2

            #appending the total thermal diffusion time in days at for the models around day 0
            tauPerRun.append(np.sum(tau)/86400)




npTau=np.array([tauPerRun])
```

```
aveTau=np.mean(npTau)
sdTau=np.std(npTau)



print("Average tau = "+str(aveTau)+" +/- "+str(sdTau))

TotalL=TotalL*(10**12) #converting Luminocites to watts

#average and sd of mass, temp, luminocity
AveM=np.mean(TotalM,axis=0)
AveT=np.mean(TotalT,axis=0)
AveL=np.mean(TotalL,axis=0)

SdM=np.std(TotalM,axis=0)
SdT=np.std(TotalT,axis=0)
SdL=np.std(TotalL,axis=0)



#print(len(TimeM))
#print(len(AveM))
tStep=np.abs(start-end)/(steps-1)

TotM=np.sum([i * tStep for i in AveM])*86400

#print(TotM)
#print(TimeM)
#print(AveM)


"""
-------------------------------------------------------------------------------------------------------------------
plotting
-------------------------------------------------------------------------------------------------------------------
"""
fig, (ax1,ax4) = plt.subplots(1,2,sharey=True,figsize=(7, 4),dpi=500)

ax21 = ax1.twinx()  # instantiate a second axis that shares the same x-axis
ax31 = ax1.twinx()

ax22 = ax4.twinx()  # instantiate a second axis that shares the same x-axis
ax32 = ax4.twinx()  # instantiate a second axis that shares the same x-axis

ax1.set_xlim(start,1)
ax4.set_xlim(1,100000)
ax4.set_xscale('log')

#----------------------------------------
#plotting mass
color = 'tab:blue'
ax1.set_ylabel('Mass Accretion rate (g/s)')  # we already handled the x-label with ax1
ax4.spines['left'].set_visible(False)
ax1.spines['right'].set_visible(False)

ax1.bar(TimeM, AveM,width=(np.abs(start-end)/steps),yerr=SdM,capsize=1,label=('Total Mass Accreted=
     ' +"{:.3e}".format(TotM)+'g' ),color='pink')
ax1.set_yscale('log')
ax4.set_yscale('log')
ax1.set_ylim(0,10**18)
ax4.set_ylim(0,10**18)

ax4.get_yaxis().set_visible(False)


#----------------------------------------

#----------------------------------------
```

```
#plotting Teff
color = 'tab:red'

ax21.errorbar(Time, AveT,yerr=SdT,fmt='.',linewidth=1,ms=3,capsize=1,mew=0.5)

ax21.set_yscale('log')
ax21.get_yaxis().set_visible(False)
ax21.set_ylim(0,1)
ax21.spines['right'].set_visible(False)

#plotting Teff
ax22.set_xlabel('Time (Days)')
ax22.set_ylabel(r'T$_{eff}$ (MK)',labelpad=-5)
ax22.errorbar(Time, AveT,yerr=SdT,fmt='.',linewidth=1,ms=3,capsize=1,mew=0.5,label=(r'T$_{eff}$'))
ax22.set_yscale('log')
ax22.spines['left'].set_visible(False)
ax22.set_ylim(0,1)


#----------------------------------------
#a bit of plotting code i found online
def make_patch_spines_invisible(ax):
    ax.set_frame_on(True)
    ax.patch.set_visible(False)
    for sp in ax.spines.values():
        sp.set_visible(False)
# Offset the right spine of par2.  The ticks and label have already been
# placed on the right by twinx above.
ax32.spines["right"].set_position(("axes", 1.35))
# Having been created by twinx, par2 has its frame off, so the line of its
# detached spine is invisible.  First, activate the frame but make the patch
# and spines invisible.
make_patch_spines_invisible(ax32)
# Second, show the right spine.
ax32.spines["right"].set_visible(True)
#----------------------------------------

#plotting luminocity
color = 'tab:black'
ax31.errorbar(Time, AveL,yerr=SdL,fmt='.',linewidth=1,ms=3,capsize=1,mew=0.5,color='k')
ax31.set_yscale('log')
ax31.set_ylim(0,10**14)
ax31.get_yaxis().set_visible(False)
ax31.spines['right'].set_visible(False)

ax32.set_ylabel('Luminosity (TW)')  # we already handled the x-label with ax1
ax32.errorbar(Time, AveL,yerr=SdL,fmt='.',linewidth=1,ms=3,capsize=1,mew=0.5,color='k',label=('L$^{\infty}$$_{ph}$'))
ax32.set_yscale('log')
ax32.spines['left'].set_visible(False)
ax32.tick_params(axis='y')
ax32.set_ylim(0,10**14)


fig.legend(loc='upper center', bbox_to_anchor=(0.4, 1.1),shadow=True, ncol=3)
fig.tight_layout()  # otherwise the right y-label is slightly clipped

fig.subplots_adjust(wspace=0,hspace=0)

fig.text(0.5,0.02, 'Time (d)', ha='center', va='center')
plt.setp(ax1.xaxis.get_majorticklabels(),ha="right")
plt.setp(ax4.xaxis.get_majorticklabels(),ha="left")


plt.savefig("Plot.png",bbox_inches='tight')

"""
---------------------------------------------------------------------------------------------------------------------
statistics
```

```
-----------------------------------------------------------------------------------------------------------------
"""

if multi==True:
    m="T"
else:
    m="F"

if unif==True:
    u="T"
else:
    u="F"

input= "\n"+str((smpl))+", "+str(cntr)+", "+str(sprd)+", "+str(skw)+", "+str(numModes)+", "+m+", "+u+"\n Averages Lum: ,"+str(list(AveL))[1:-1]+"\n Sd Lum: ,"+ str(list(SdL))[1:-1]
file=open('LumOut.txt','a')
file.write(input) #line not found
```