white dwarfs

Course on Compact Objects: Assignment 4

Instructor: Prof. P. Ajith Author: Md Arif Shaikh, Postdoc, ICTS-TIFR Date: Feb 5, 2021

Problem 1

Show that the pressure exerted by a gas of particle with isotropic momentum distribution $n(p)$ is given by

$$P = \frac{1}{3}\int_0^\infty p v_p n(p) dp$$

where v_p is the velocity associated with momentum p.

Let us consider the pressure in the z direction. The component of velocity of along the z direction would be given by $v\cos\theta$ where θ is the angle of the velocity vector with the z axis.

$$v_z = v\cos\theta$$

and therefore,

$$p_z = p\cos\theta$$

Now the number of particles in a volume element d^3p in momentum space with momentum between p and $p + dp$ is

$$d^3p = p^2 dp \sin\theta d\theta d\phi$$

Now the rate of change of momentum is given by

$$\frac{p_z}{\Delta z/v_z} = \frac{p_z v_z}{\Delta z} = \frac{pv\cos^2\theta}{\Delta z}$$

So the pressure per particle is

$$\frac{pv\cos^2\theta}{\Delta z\Delta x\Delta y} = \frac{pv\cos^2\theta}{V}$$

So the total pressure would be given by

$$P = \int_0^\infty \frac{pv\cos^2\theta}{V} N(p)dp$$

where $N(p)$ is the total number of particles with momentum between p and $p + dp$ and with angle θ to $\theta + d\theta$ which is $2\pi p^2 dp \sin\theta d\theta$ (which is obtained by integrating d^3p over ϕ)

Thus,

$$P = \int_0^\infty \frac{pv\cos^2\theta}{V} 2\pi \sin\theta d\theta p^2 dp$$

or

$$P = \int_0^\pi \cos^2\theta d\theta \sin\theta \int_0^\infty \frac{pv}{V} 2\pi p^2 dp = \frac{1}{3}\int_0^\infty \frac{pv}{V} 4\pi p^2 dp = \frac{1}{3}\int_0^\infty pvn(p)dp$$

where $n(p) = 4\pi p^2/V$

Problem 2

Argue why we are justified in using a 'cold' degenerate equation of state to describe a white dwarf with a temperature $T \sim 10^4$K (Hint: Show that the degeneracy parameter $\mu/kT \gg 0$, where μ is the chemical potential and k the Boltzmann constant. The density of the white dwarf is $\sim 10^6 g/cm^3$ and the chemical potential $\sim$ the Fermi energy. Assume that $\mu_e = 2$).

Problem 3

Derive Lane-Emden equation

Lane-Emden equation

$$\frac{dm(r)}{dr} = 4\pi r^2 \rho(r), \frac{dp(r)}{dr} = -\frac{Gm(r)\rho(r)}{r^2}$$

. These two equations can be combined to give

$$\frac{1}{r^2}\frac{d}{dr}\left(\frac{r^2}{\rho(r)}\frac{dp(r)}{dr}\right) = -4\pi G\rho(r)$$

This can be further written in terms of a dimensionless form using

$$\rho = \rho_c\theta^n, r = a\xi, \Gamma = 1 + \frac{1}{n}$$

Then we get the following equation

$$\frac{1}{\xi^2}\frac{d}{d\xi}\left(\xi^2\frac{d\theta}{d\xi}\right) = -\theta^n$$

This the Lane-Emden equation.

Problem 4

Solve Lane-Emden Equation

To solve the lane-emden equation we write it as two first order odes by defining $\psi = \frac{d\theta}{d\xi}$

$$\frac{d\theta}{d\xi} = \psi, \frac{d\psi}{d\xi} = -\frac{2\psi}{\xi} - \theta^n$$

With the following initial values

$$\theta(0) = 1, \psi(0) = 0$$

```
using DifferentialEquations
```

```
using StaticArrays
```

```
laneEmden (generic function with 1 method)
```

```
function laneEmden(u, n, ξ)
    θ, ψ = u
    dθ = ψ
    dψ = - (2 * ψ / ξ) - θ^n
    @SVector [dθ, dψ]
end
```

```
u0 = ▸StaticArrays.SArray{Tuple{2},Float64,1,2}: [1.0,  0.0]
```

```
u0 = @SVector [1., 0.]
```

```
ξspan1 = ▸(1.0e-10,  10)
```

```
ξspan1 = (1e-10, 10)
```

```
problem1 =
[36mODEProblem[0m with uType [36mStaticArrays.SArray{Tuple{2},Float64,1,2}[0m and tTyp
timespan: (1.0e-10, 10.0)
u0: [1.0, 0.0]
```

```
problem1 = ODEProblem(laneEmden, u0, ξspan1, 1.)
```

sol1 =

timestamp	value1	value2

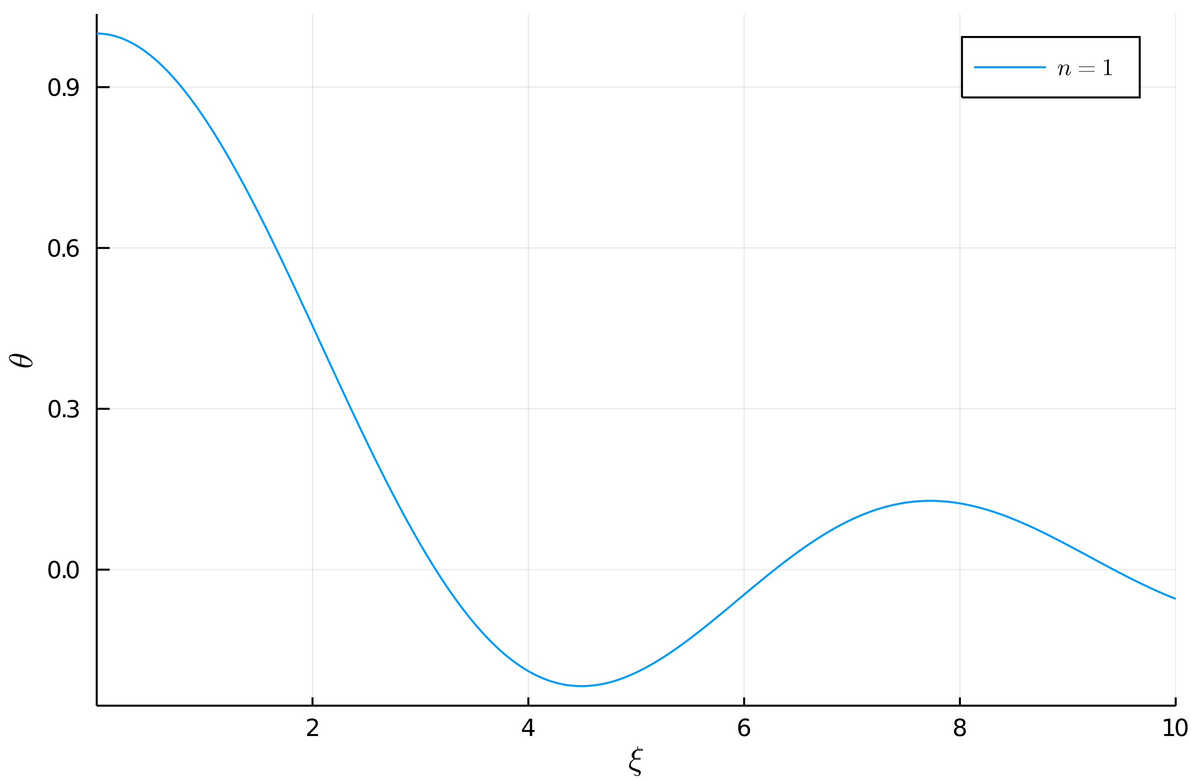
	timestamp	value1	value2
1	1.0e-10	1.0	0.0
2	3.05653e-5	1.0	-9.0812e-6
3	4.38094e-5	1.0	-1.40642e-5
4	0.000107578	1.0	-3.57691e-5
5	0.000187092	1.0	-6.2334e-5
6	0.000467431	1.0	-0.000155805
7	0.000989918	1.0	-0.000329971
8	0.00256049	0.999999	-0.000853496
9	0.00736913	0.999991	-0.00245636
10	0.0331064	0.999817	-0.0110343

```
sol1 = solve(problem1)
```

```
▸Plots.GRBackend()
```

```
using Plots; gr()
```

```
using LaTeXStrings
```




```
plot(sol1, vars = (0, 1), xlabel = L"\xi", ylabel = L"\theta", label = L"n = 1")
```

Solutions for different integer n

```
Enter cell code...
```

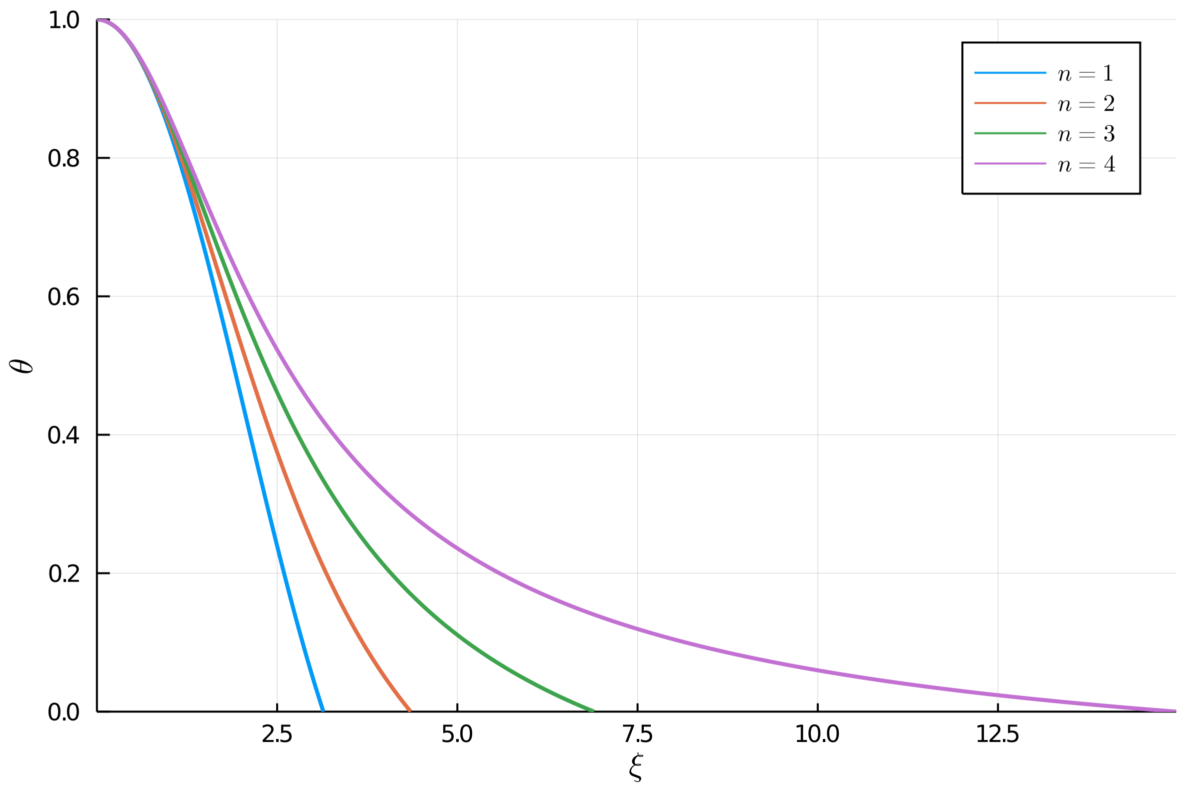
```
condition(u, ξ, integrator) =  u[1];
```

```
affect!(integrator) = terminate!(integrator);
```

```
cb = ContinuousCallback(condition, affect!);
```

```
ξspan = ▸(1.0e-10, 20)
```

```
ξspan = (1e-10, 20)
```




```
begin
    p = plot();
    for n in 1:4
        problem = ODEProblem(laneEmden, u0, ξspan, n)
        sol = solve(problem, callback = cb)
        plot!(p, sol, vars = (0, 1), lw = 2, label = L"n = %$n")
    end
    plot!(xlabel = L"\xi", ylabel = L"\theta", ylim = (0, 1))
    p
end
```

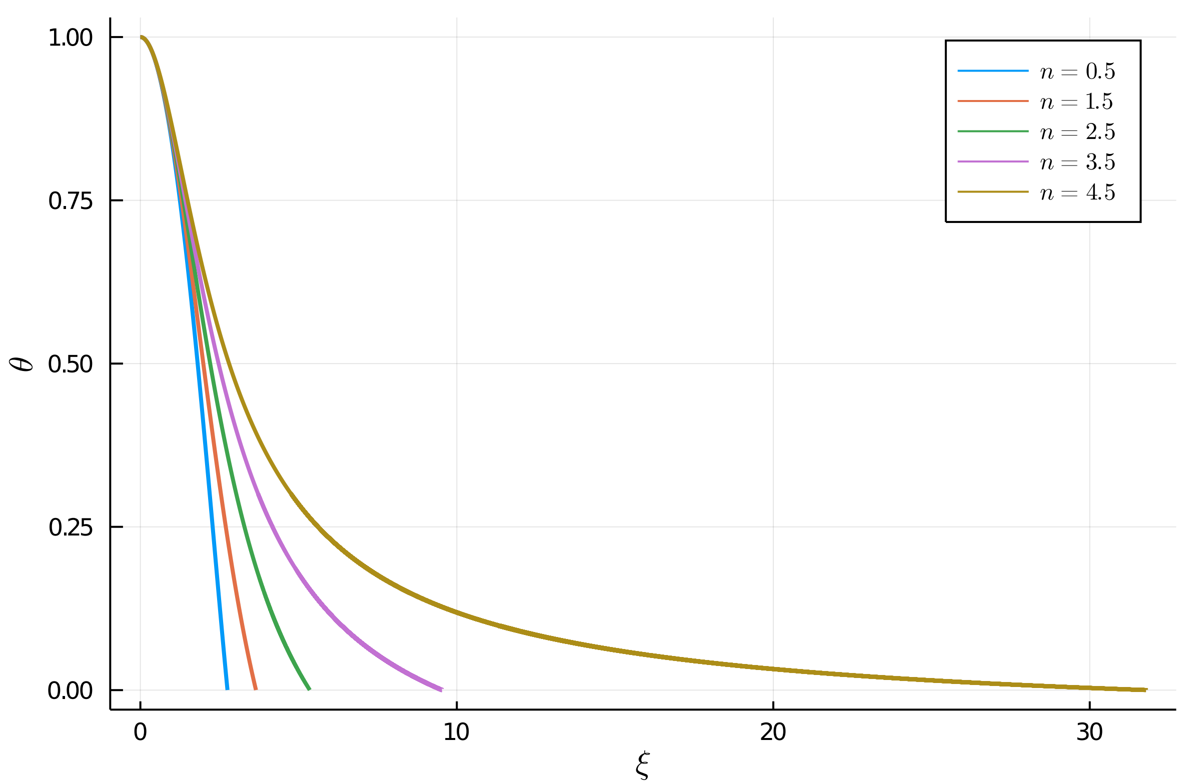
for fractional n the solver would throw error as negative value of θ would give complex value for θ^n. Callback function does not work for some reason. Therefore we manually check at every step whether θ is negative. We stop whenever $\theta < \epsilon$. where ϵ is a very small number

```
begin
    pfrac = plot();
    ns = 0.5:1.0:4.5
    ξ1s = []
    ψs = []
    for n in ns
        θfracsol = []
```

```julia
        ψfracsol = []
        ξs = []
        θfrac = 1.
        ψfrac = 0.
        ξ = 1e-8
        dξ = 1e-4
        while θfrac > dξ
            push!(θfracsol, θfrac)
            push!(ψfracsol, ψfrac)
            push!(ξs, ξ)
            θfracini = θfrac
            ψfracini = ψfrac
            u0frac = @SVector [θfracini, ψfracini]
            ξspanfrac = (ξ, ξ+dξ)
            problem = ODEProblem(laneEmden, u0frac, ξspanfrac, n)
            sol = solve(problem)
            θfrac = sol.u[end][1]
            ψfrac = sol.u[end][2]
            println(sol)
            ξ += dξ
        end
        plot!(pfrac, ξs, θfracsol, lw = 2, xlabel = L"\xi", ylabel = L"\theta", label = L"n = %$n")
        push!(ξ1s, (ξ + dξ/2.))
        push!(ψs, ψfrac)
    end
end
```

```
Any[2.75255, 3.65335, 5.35405, 9.53105, 31.7784]
```

```julia
ξ1s
```




```julia
pfrac
```

```julia
using DataFrames
```

data =

	n	ξ1	ψ1
1	0.5	2.75255	-0.500068
2	1.5	3.65335	-0.203352
3	2.5	5.35405	-0.0763013
4	3.5	9.53105	-0.020812
5	4.5	31.7784	-0.00172083

```
data = DataFrame(n = ns, ξ1 = ξ1s, ψ1 = ψs)
```

Problem 5

Radius and Mass of white dwarf

Mass of the star is given by

$$M = \int_0^\infty 4\pi \rho r^2 dr$$

now,

$$r = \xi \left(\frac{4\pi G {\rho_c}^{1-\frac{1}{n}}}{K(n+1)} \right)^{-\frac{1}{2}}$$

$$\rho = \rho_c \theta^n$$

$$\theta^n = -\frac{1}{\xi^2} \frac{d}{d\xi} \left(\xi^2 \frac{d\theta}{d\xi} \right)$$

Thus,

$$M = 4\pi\rho_c \left(\frac{4\pi G {\rho_c}^{1-\frac{1}{n}}}{K(n+1)} \right)^{-3/2} \int_0^{\xi_1} d\left(\xi^2 \frac{d\theta}{d\xi} \right) = 4\pi\rho_c \left(\frac{4\pi G {\rho_c}^{1-\frac{1}{n}}}{K(n+1)} \right)^{-3/2} \left(\xi^2 \frac{d\theta}{d\xi} \right)_{\xi_1}$$

or

$$M = 4\pi\rho_c^{(3-n)/2n} \left(\frac{4\pi G}{K(n+1)} \right)^{-3/2} \left(\xi^2 \frac{d\theta}{d\xi} \right)_{\xi_1}$$

Now, the radius would be given by value of r at $\xi = \xi_1$ which is

$$R_\star = \xi_1 \left(\frac{4\pi G {\rho_c}^{1-\frac{1}{n}}}{K(n+1)} \right)^{-\frac{1}{2}}$$

From this we can express ρ_c in terms of $R_\star$ and ξ_1 as

$$\rho_c = \left(\frac{K(n+1)\xi_1^2}{4\pi G R_\star^2} \right)^{\frac{n}{n-1}}$$

So,

$$\boxed{M = 4\pi R_\star^{(3-n)/(1-n)} \left(\frac{K(n+1)}{4\pi G} \right)^{n/(n-1)} \xi_1^{-(3-n)/(1-n)} \xi_1^2 \frac{d\theta}{d\xi}\bigg|_{\xi_1}}$$

Problem 6

Compute mass and radius of white dwarfs

From problem 4, we get that for $n = 3/2$, $\xi_1 \approx 3.655$. Now given that

$$\rho_c = 10^6 - 10^9 g/cm^3, K \approx 10^{13} \mu_e^{-5/3}, \mu_e = 2.$$

```
MassWhiteDwarf (generic function with 1 method)
```

```
function MassWhiteDwarf(R, K, G, ξ1, ψ1, n)
    4 * pi * R^((3. -n)/(1. - n))*((K*(n+1))/(4*pi*G))^(n/(n-1))*ξ1^(-(3. - n)/(1. -
n))*ξ1^2 * abs(ψ1)
end
```

```
Rstar (generic function with 1 method)
```

```
function Rstar(G, ξ1, ρc, K, n)
    return ξ1 * (4 * pi * G * ρc^(1.0 - (1.0 / n))/(K * (n + 1)))^(-1.0 / 2.0)
end
```

```
ξ1 = 3.653350010003285
```

```
ξ1 = data.ξ1[2]
```

```
ψ1 = -0.2033517841335034
```

```
ψ1 = data.ψ1[2]
```

```
using Unitful, UnitfulAstro
```

```
G = GM⊙ M⊙^-1
```

```
G = u"GMsun"/u"Msun"
```

```
μ = 2.0
```

```
μ = 2.0
```

```
n = 1.5
n = 3/2
```

```
K = 3.149802624737183e12 dyn cm^3 g^-3752999689475413/2251799813685248
K = 1e13 * μ^(-5.0/3.0) * u"dyn/cm^2"/(u"g/cm^3")^(1+(1/n))
```

```
ρcs =
▸Unitful.Quantity{Float64,M L^-3,Unitful.FreeUnits{(g, cm^-3),M L^-3,nothing}}[1.0e6 g cm^
ρcs = [1e6, 1e7, 1e8, 1e9] * u"g/cm^3"
```

```
1.0e7 g cm^-3
ρcs[2]
```

```
r1 = 11194.25414803915 km
r1 = uconvert(u"km", Rstar(G, ξ1, ρcs[1], K, n))
```

```
mass1 = 2.9172203078632432e26 dyn^3 cm^9 M⊙^3 GM⊙^-3 g^-5 km^-3
mass1 = MassWhiteDwarf(r1, K, G, ξ1, ψ1, n)
```

```
0.4934534013092127 M⊙
uconvert(u"Msun", mass1)
```

```
begin
    masses = []
    radii = []

    for ρc in ρcs
        radius = uconvert(u"km", Rstar(G, ξ1, ρc, K, n))
        mass = uconvert(u"Msun", MassWhiteDwarf(radius, K, G, ξ1, ψ1, n))
        push!(masses, mass)
        push!(radii, radius)
    end
end
```

massRadius =

	central_density	radius	mass
1	1.0e6 g cm^-3	11194.3 km	0.493453 M⊙
2	1.0e7 g cm^-3	7626.56 km	1.56044 M⊙
3	1.0e8 g cm^-3	5195.91 km	4.93453 M⊙
4	1.0e9 g cm^-3	3539.93 km	15.6044 M⊙

```
massRadius = DataFrame(central_density = ρcs, radius = radii, mass = masses)
```