Investigando en Física Efectos de Supernovas en Sistemas Planetarios

September 30, 2022

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy import optimize
import astropy.units as u
import astropy.constants as cons
```

Hola buenas, si estás leyendo esto, probablemente vienes porque estabas viendo el póster de Investigando en Física y escaneaste el QR. En este documento se muestran definidas en python todas las funciones paramétricas usadas en dicha investigación. Cualquier problema y/o duda con las funciones no duden en contactarme. Ni idea de si se puede contactar a través del github, así que les dejo mi correo: alejandro.guzman.a@gmail.com. Este documento esta disponible en formato pdf para lectura, como archivo .ipynb por si quieren ver el código en jupyter para probar las funciones y también pasaré las funciones a un archivo .py para la gente que no usa jupyter (úsenlo, es super bacán :)).

Ojito: Muchas de estas funciones contienen 'if statements' que operan con variables booleanas, esto hace que las funciones no puedan operar con arrays en sus parámetros. Digamos que deseas graficar alguna de estas funciones, entonces alguno de los parámetros tendría que ser un array de datos, ahí estas funciones no te van a entregar un array nuevo asociado a cada elemento, de hecho te va a tirar un error por lo que acabo de decir. Una forma de solucionar esto es evaluar para cada elemento individualmente usando ciclos for, pero sería muy demandante y feo. Personalmente utilizaba la función de numpy llamada vectorize para solucionar esto y los invito a hacerlo también. ;)

1 Radio estelar

Estas son las funciones utilizadas para calcular el radio estelar durante la secuencia principal en función de la masa, metalicidad y tiempo en la secuencia principal. Todas las funciones fueron obtenidas de Hurley et al.(2000), a excepción de la función R_{ZAMS} (R_ZAMS), la cual se obtiene de Tout et al.(1996) (El paper de Hurley mismo nos dice que saquemos esta función de aquí).

[2]:
```
#Variable dseta que es de utilidad en las siguientes funciones
def dseta(Z):
    '''
    Función que entrega una variable asociada a la metalicidad de la estrella.

    Es una cantidad adimensional.
```

```
    Depende de:

    Z: Metalicidad de la estrella
    '''

    return np.log10(Z/0.02)

#Radio estelar a edad cero de la secuancia principal
def R_ZAMS(M, Z):
    '''
    Función que entrega el radio de una estrella cuando entra a la secuencia
    principal (Zero Age Main Sequence).

    El radio entregado está en radios solares.
    Depende de:

    Z: Metalicidad de la estrella
    M: Masa de la estrella (Masas solares)
    '''

    x = dseta(Z)     #Obtenemos dzeta a partir de la metalicidad

    # Calculamos los coeficientes en función de dseta
    theta   =  1.71535900 + 0.62246212*x -  0.92557761*x**2 -  1.16996966*x**3
 ↪- 0.30631491*x**4
    iota    =  6.59778800 - 0.42450044*x - 12.13339427*x**2 - 10.73509484*x**3
 ↪- 2.51487077*x**4
    kappa   = 10.08855000 - 7.11727086*x - 31.67119479*x**2 - 24.24848322*x**3
 ↪- 5.33608972*x**4
    lamda   =  1.01249500 + 0.32699690*x -  0.00923418*x**2 -  0.03876858*x**3
 ↪- 0.00412750*x**4
    mu      =  0.07490166 + 0.02410413*x +  0.07233664*x**2 +  0.03040467*x**3
 ↪+ 0.00197741*x**4
    nu      =  0.01077422
    xi      =  3.08223400 + 0.94472050*x -  2.15200882*x**2 -  2.49219496*x**3
 ↪- 0.63848738*x**4
    omicron = 17.84778000 - 7.45345690*x - 48.96066856*x**2 - 40.05386135*x**3
 ↪- 9.09331816*x**4
    pi      =  0.00022582 - 0.00186899*x +  0.00388783*x**2 +  0.00142402*x**3
 ↪- 0.00007671*x**4

    #Calculamos el radio en función de los coeficientes y la masa estelar
    R_ZAMS = (theta*(M**2.5) + iota*(M**6.5) + kappa*(M**11) + lamda*(M**19) +
 ↪mu*(M**19.5))/(nu + xi*(M**2) + omicron*(M**8.5) + M**18.5 + pi*(M**19.5))

    return R_ZAMS
```

```
#Masa de "enganche"
def M_hook(Z):
    '''
    Función que calcula la masa que debe tener una estrella de metalicidad Z
    para que haya un gancho en la secuencia principal.

    La masa entregada está en masas solares
    Depende de:

    Z: Metalicidad de la estrella
    '''

    x = dseta(Z)     #Obtenemos dzeta a partir de la metalicidad

    #Calculamos M_hook
    M_hook = 1.0185 + 0.16015*x + 0.0892*x**2

    return M_hook

#Tiempo que se tarda en llegar una estrella a la base de la rama de las gigantes
def t_BGB(M, Z):
    '''
    Función que entrega el tiempo que tarda una estrella en llegar a la base de
    la rama de las gigantes.

    El tiempo entregado está en Myr.
    Depende de:

    Z: Metalicidad de la estrella
    M: Masa de la estrella (Masas solares)
    '''

    x = dseta(Z)     #Obtenemos dzeta a partir de la metalicidad

    #Obtenemos los coeficientes en función de dseta
    a1 = 1.593890*(10**3)  + 2.053038*(10**3)*x  + 1.231226*(10**3)*x**2  + 2.
↪327785*(10**2)*x**3
    a2 = 2.706708*(10**3)  + 1.483131*(10**3)*x  + 5.772723*(10**2)*x**2  + 7.
↪411230*(10**1)*x**3
    a3 = 1.466143*(10**2)  - 1.048442*(10**2)*x  - 6.795374*(10**1)*x**2  - 1.
↪391127*(10**1)*x**3
    a4 = 4.141960*(10**-2) + 4.564888*(10**-2)*x + 2.958542*(10**-2)*x**2 + 5.
↪571483*(10**-3)*x**3
    a5 = 3.426349*(10**-1)

    #Calculamos t_BGB en función de la masa y la metalicidad
```

```
    t_BGB = (a1 + a2*M**4 + a3*M**5.5 + M**7)/(a4*M**2 + a5*M**7)

    return t_BGB

#Tiempo en el que aparece el "enganche" en el camino de la secuencia principal
def t_hook(M, Z):
    '''
    Función que entrega el tiempo que tarda la estrella en llegar a la parte de
    su camino por la secuencia principal donde aparece un gancho.

    El tiempo entregado está en Myr.
    Depende de:

    Z: Metalicidad de la estrella
    M: Masa de la estrella (Masas solares)
    '''

    x = dseta(Z)     #Obtenemos dzeta a partir de la metalicidad

    #Obtenemos los coeficientes en función de dseta
    a6  = 1.949814*(10**1)  + 1.758178*(10**0)*x  - 6.008212*(10**0)*x**2  - 4.
 ↪470533*(10**0)*x**3
    a7  = 4.903830*(10**0)
    a8  = 5.212154*(10**-2) + 3.166411*(10**-2)*x - 2.750074*(10**-3)*x**2 - 2.
 ↪271549*(10**-3)*x**3
    a9  = 1.312179*(10**0)  - 3.294936*(10**-1)*x + 9.231860*(10**-2)*x**2 + 2.
 ↪610989*(10**-2)*x**3
    a10 = 8.073972*(10**-1)

    #Calculamos el coeficiente mu
    mu = max(0.5, 1.0 - 0.01*max((a6/(M**a7)), a8 + (a9/(M**a10))))

    #Obtenemos t_BGB en función de M y Z
    t_hook = mu*t_BGB(M,Z)

    return t_hook

#Tiempo que pasará la estrella en la secuencia principal
def t_MS(M, Z):
    '''
    Función que entrega el tiempo de vida de una estrella en la secuencia
    principal.

    El tiempo entregado está en Myr.
    Depende de:

    Z: Metalicidad de la estrella
```

```
    M: Masa de la estrella (Masas solares)
    '''

    x = dseta(Z)     #Obtenemos dzeta a partir de la metalicidad

    #Calculamos el coeficiente x
    X = max(0.95, min(0.95 - 0.03*(x + 0.30103), 0.99))

    #Obtenemos t_MS en función de M y Z
    t_MS = max(t_hook(M, Z), X*t_BGB(M, Z))

    return t_MS

#Tau (t/t_MS)
def tau(t, M, Z):
    '''
    Función que entrega la fracción del tiempo de vida de una estrella en la
    secuencia principal.

    Es una cantidad adimensional.
    Depende de:

    t: Tiempo de vida de la estrella (En Myr)
    Z: Metalicidad de la estrella
    M: Masa de la estrella (Masas solares)
    '''
    #Calcula t_MS y luego divide el tiempo t sobre este valor
    tau = t/t_MS(M, Z)

    return tau

#Radio de la estrella al término de la secuencia principal
def R_TMS(M, Z):
    '''
    Función que entrega el radio de una estrella cuando sale de la secuencia
    principal.

    El radio entregado está en radios solares.
    Depende de:

    Z: Metalicidad de la estrella
    M: Masa de la estrella (Masas solares)
    '''

    x = dseta(Z)     #Obtenemos dzeta a partir de la metalicidad

    #Obtenemos sigma, que es el logaritmo de la metalicidad
```

```
    sigma = np.log10(Z)

    #Obtenemos los coeficientes a partir de dseta y sigma
    a17   = 10**(max(0.097 - 0.1072*(sigma + 3), max(0.097, min(0.1461, 0.1461
↪+ 0.1237*(sigma + 2)))))
    a18_p = 2.187715*(10**-1) - 2.154437*(10**0)*x  - 3.768678*(10**0)*x**2  -
↪1.975518*(10**0)*x**3 - 3.021475*(10**-1)*x**4
    a19_p = 1.466440*(10**0)  + 1.839725*(10**0)*x  + 6.442199*(10**0)*x**2  +
↪4.023635*(10**0)*x**3 + 6.957529*(10**-1)*x**4
    a20   = 2.652091*(10**1)  + 8.178458*(10**1)*x  + 1.156058*(10**2)*x**2  +
↪7.633811*(10**1)*x**3 + 1.950698*(10**1)*x**4
    a21   = 1.472103*(10**0)  - 2.947609*(10**0)*x  - 3.312828*(10**0)*x**2  -
↪9.945065*(10**-1)*x**3
    a22   = 3.071048*(10**0)  - 5.679941*(10**0)*x  - 9.745523*(10**0)*x**2  -
↪3.594543*(10**0)*x**3
    a23   = 2.617890*(10**0)  + 1.019135*(10**0)*x  - 3.292551*(10**-2)*x**2 -
↪7.445123*(10**-2)*x**3
    a24   = 1.075567*(10**-2) + 1.773287*(10**-2)*x + 9.610479*(10**-3)*x**2 +
↪1.732469*(10**-3)*x**3
    a25   = 1.476246*(10**0)  + 1.899331*(10**0)*x  + 1.195010*(10**0)*x**2  +
↪3.035051*(10**-1)*x**3
    a26   = 5.502535*(10**0)  - 6.601663*(10**-2)*x + 9.968707*(10**-2)*x**2 +
↪3.599801*(10**-2)*x**3
    a18   = a18_p*a20
    a19   = a19_p*a20
    c1    = -8.672073*10**-2 #Este es un coeficiente cuyo valor es entregado

    #Obtenemos M_x, que va a ser un límite para nuestra función seccionada
    M_x = a17 + 0.1

    #Calculamos unos coeficientes que nos serviran para hacer una interpolación
    #lineal entre los límites de ambas secciones de la función. Son simplemente
    #la función evaluada en los límites de cada sección.
    y1 = (a18 + a19*a17**a21)/(a20 + a17**a22)
    y2 = (c1*M_x**3 + a23*M_x**a26 + a24*M_x**(a26 + 1.5))/(a25 + M_x**5)
    ##########################################################################
    #Ahora calculamos R_TMS, pero cómo se calcula depende del valor de M:

    if M <= a17: #Forma de calcularlo si M <= a17
        R_TMS = (a18 + a19*M**a21)/(a20 + M**a22)

        if M < 0.5: #Esta es una condición extra que ocurre cuando M < 0.5
                R_TMS = max(R_TMS, 1.5*R_ZAMS(M, Z))
        else:
            pass
```

```
    elif a17 < M < M_x: #Esta es la interpolación lineal entre a17 y M_x
        R_TMS = ((y2 - y1)/(M_x - a17))*(M - a17) + y1

    elif M >= M_x: #Forma de calcularlo si M >= M_x
        R_TMS = (c1*M**3 + a23*M**a26 + a24*M**(a26 + 1.5))/(a25 + M**5)


    return R_TMS

#Tau_1
def tau_1(t, M, Z):
    '''
    Función que entrega tau_1.

    Es una cantidad adimensional.
    Depende de:

    t: Tiempo de vida de la estrella (En Myr)
    Z: Metalicidad de la estrella
    M: Masa de la estrella (Masas solares)
    '''

    tau_1 = min(1.0, t/t_hook(M, Z))

    return tau_1

#Tau_2
def tau_2(t, M, Z):
    '''
    Función que entrega tau_2.

    Es una cantidad adimensional.
    Depende de:

    t: Tiempo de vida de la estrella (En Myr)
    Z: Metalicidad de la estrella
    M: Masa de la estrella (Masas solares)
    '''

    tau_2 = max(0.0, min(1.0, (t - (1.0 - 0.01)*t_hook(M, Z))/(0.01*t_hook(M,␣
 ↪Z))))

    return tau_2

#Delta R
def deltaR(M, Z):
    '''
```

```
    Función que entrega deltaR.

    Es una cantidad adimensional.
    Depende de:

    Z: Metalicidad de la estrella
    M: Masa de la estrella (Masas solares)
    '''

    x = dseta(Z)      #Obtenemos dseta a partir de la metalicidad

    #Calculamos los coeficientes que dependen de dseta
    a38 = 7.330122*(10**-1) + 5.192827*(10**-1)*x + 2.316416*(10**-1)*x**2 + 8.
↪346941*(10**-3)*x**3
    a39 = 1.172768*(10**0)  - 1.209262*(10**-1)*x - 1.193023*(10**-1)*x**2 - 2.
↪859837*(10**-2)*x**3
    a40 = 3.982622*(10**-1) - 2.296279*(10**-1)*x - 2.262539*(10**-1)*x**2 - 5.
↪219837*(10**-2)*x**3
    a41 = 3.571038*(10**0)  - 2.223625*(10**-2)*x - 2.611794*(10**-2)*x**2 - 6.
↪359648*(10**-3)*x**3
    a42 = 1.9848*(10**0)    + 1.1386*(10**0)*x    + 3.5640*(10**-1)*x**2
    a43 = 6.300*(10**-2)    + 4.810*(10**-2)*x    + 9.840*(10**-3)*x**2
    a44 = 1.200*(10**0)     + 2.450*(10**0)*x

    #Aplicamos condiciones sobre los valores de algunos coeficientes
    a42 = min(1.25, max(1.10, a42))
    a44 = min(1.30, max(0.45, a44))

    #Calculamos el valor B, que viene a ser deltaR cuando M = 2
    B   = ((a38 + a39*2.0**3.5)/(a40*2.0**3 + 2.0**a41)) - 1.0

    #Obtenemos M_hook
    m   = M_hook(Z)

    ################################################################################
    #Ahora calculamos deltaR, pero la forma de calcularla depende de M

    if M <= m: #Forma de calcular deltaR cuando M <= M_{hook}
        deltaR = 0.0

    elif m < M <= a42: #Forma de calcular deltaR cuando M_{hook} < M <= a42
        deltaR = a43*((M - m)/(a42 - m))**0.5

    elif a42 < M < 2.0: #Forma de calcular deltaR cuando a42 < M < 2
        deltaR = a43 + (B - a43)*((M - a42)/(2.0 - a42))**a44

    elif 2.0 <= M: #Forma de calcular deltaR cuando 2 <= M
```

```
        deltaR = ((a38 + a39*M**3.5)/(a40*M**3 + M**a41)) - 1.0


    return deltaR

#Alpha R
def alpha_R(M, Z):
    '''
    Función que entrega alpha_R.

    Es una cantidad adimensional.
    Depende de:

    Z: Metalicidad de la estrella
    M: Masa de la estrella (Masas solares)
    '''

    x = dseta(Z)     #Obtenemos dseta a partir de la metalicidad

    #Calculamos los coeficientes que dependen de dseta
    a58 = 4.907546*(10**-1) - 1.683928*(10**-1)*x - 3.108742*(10**-1)*x**2 - 7.
↪202918*(10**-2)*x**3
    a59 = 4.537070*(10**0)  - 4.465455*(10**0)*x  - 1.612690*(10**0)*x**2  - 1.
↪623246*(10**0)*x**3
    a60 = 1.796220*(10**0)  + 2.814020*(10**-1)*x + 1.423325*(10**0)*x**2  + 3.
↪421036*(10**-1)*x**3
    a61 = 2.256216*(10**0)  + 3.773400*(10**-1)*x + 1.537867*(10**0)*x**2  + 4.
↪396373*(10**-1)*x**3
    a62 = 8.4300*(10**-2)   - 4.7500*(10**-2)*x   - 3.5200*(10**-2)*x**2
    a63 = 7.3600*(10**-2)   + 7.4900*(10**-2)*x   + 4.4260*(10**-2)*x**2
    a64 = 1.3600*(10**-1)   + 3.5200*(10**-2)*x
    a65 = 1.564231*(10**-3) + 1.653042*(10**-3)*x - 4.439786*(10**-3)*x**2 - 4.
↪951011*(10**-3)*x**3 -1.216530*(10**-3)*x**4
    a66 = 1.4770*(10**0)    + 2.9600*(10**-1)*x
    a67 = 5.210157*(10**0)  - 4.143695*(10**0)*x  - 2.120870*(10**0)*x**2
    a68 = 1.1160*(10**0)    + 1.6600*(10**-1)*x

    #Aplicamos condiciones sobre los valores de algunos coeficientes
    a62 = max(0.065, a62)

    if Z < 0.004:
        a63 = min(0.055, a63)
    else:
        pass

    a64 = max(0.091, min(0.121, a64))
    a66 = max(a66, min(1.6, -0.308 - 1.046*x))
```

```
    a66 = max(0.8, min(0.8-2*x, a66))
    a68 = max(0.9, min(a68, 1.0))

    B = (a58*a66**a60)/(a59 + a66**a61) #El valor de alpha_R cuando M = a66
    C = (a58*a67**a60)/(a59 + a67**a61) #El valor de alpha_R cuando M = a67

    if a68 > a66:
         a64 = B
    else:
        pass

    a68 = min(a68, a66)
    ###############################################################################
    #Ahora calculamos alpha_R, pero la forma de calcularla depende de M

    if M < 0.50: #Forma de calcular alpha_R cuando M < 0.5
        alpha_R = a62

    elif 0.50 <= M < 0.65: #Forma de calcular alpha_R cuando 0.5 <= M <0.65
        alpha_R = a62 + (((a63 - a62)*(M - 0.5))/0.15)

    elif 0.65 <= M < a68: #Forma de calcular alpha_R cuando 0.65 <= M < a68
        alpha_R = a63 + ((a64 - a63)*(M - 0.65))/(a68 - 0.65)

    elif a68 <= M < a66: #Forma de calcular alpha_R cuando a68 <= M < a66
        alpha_R = a64 + ((B - a64)*(M - a68))/(a66 - a68)

    elif a66 <= M <= a67: #Forma de calcular alpha_R cuando a66 <= M <= a67
        alpha_R = (a58*M**a60)/(a59 + M**a61)

    elif a67 < M: #Forma de calcular alpha_R cuando a67 < M
        alpha_R = C + a65*(M - a67)


    return alpha_R

#Beta R
def beta_R(M, Z):
    '''
    Función que entrega beta_R.

    Es una cantidad adimensional.
    Depende de:

    Z: Metalicidad de la estrella
    M: Masa de la estrella (Masas solares)
    '''
```

```
    x = dseta(Z)      #Obtenemos dseta a partir de la metalicidad

    #Calculamos los coeficientes que dependen de dseta
    a69 = 1.071489*(10**0)  - 1.164852*(10**-1)*x - 8.623831*(10**-2)*x**2 - 1.
↪582349*(10**-2)*x**3
    a70 = 7.108492*(10**-1) + 7.935927*(10**-1)*x + 3.926983*(10**-1)*x**2 + 3.
↪622146*(10**-2)*x**3
    a71 = 3.478514*(10**0)  - 2.585474*(10**-2)*x - 1.512955*(10**-2)*x**2 - 2.
↪833691*(10**-3)*x**3
    a72 = 9.132108*(10**-1) - 1.653695*(10**-1)*x + 3.636784*(10**-2)*x**3
    a73 = 3.969331*(10**-3) + 4.539076*(10**-3)*x + 1.720906*(10**-3)*x**2 + 1.
↪897857*(10**-4)*x**3
    a74 = 1.600*(10**0)     + 7.640*(10**-1)*x    + 3.322*(10**-1)*x**2

    #Aplicamos condiciones sobre los valores de algunos coeficientes
    if Z > 0.01:
        a72 = max(a72, 0.95)
    else:
        pass

    a74 = max(1.4, min(a74, 1.6))

    B   = (a69*2**3.5)/(a70 + 2**a71)   #El valor de beta_R cuando M = 2
    C   = (a69*16**3.5)/(a70 + 16**a71) #El valor de beta_R cuando M = 16

    ###############################################################################
    #Ahora calculamos beta_R, pero la forma de calcularla depende de M

    if M <= 1.0: #Forma de calcular beta_R cuando M <=1
        beta_R = 1.06

    elif 1.0 < M < a74: #Forma de calcular beta_R cuando 1 < M <a74
        beta_R = 1.06 + ((a72 - 1.06)*(M - 1.0))/(a74 - 1.06)

    elif a74 <= M < 2.0: #Forma de calcular beta_R cuando a74 <= M <2
        beta_R = a72 + ((B - a72)*(M - a74))/(2.0 - a74)

    elif 2.0 <= M <= 16.0: #Forma de calcular beta_R cuando 2 <= M <= 16
        beta_R = (a69*M**3.5)/(a70 + M**a71)

    elif 16.0 < M: #Forma de calcular beta_R cuando 16 < M
        beta_R = C + a73*(M - 16.0)


    return beta_R - 1
```

```
#Gamma
def gamma(M, Z):
    '''
    Función que entrega gamma.

    Es una cantidad adimensional.
    Depende de:

    Z: Metalicidad de la estrella
    M: Masa de la estrella (Masas solares)
    '''

    x = dseta(Z)     #Obtenemos dseta a partir de la metalicidad

    #Calculamos los coeficientes que dependen de dseta
    a75 =  8.109*(10**-1)    - 6.282*(10**-1)*x
    a76 =  1.192334*(10**-2) + 1.083057*(10**-2)*x + 1.230969*(10**0)*x**2  + 1.
↪551656*(10**0)*x**3
    a77 = -1.668868*(10**-1) + 5.818123*(10**-1)*x - 1.105027*(10**1)*x**2  - 1.
↪668070*(10**1)*x**3
    a78 =  7.615495*(10**-1) + 1.068243*(10**-1)*x - 2.011333*(10**-1)*x**2 - 9.
↪371415*(10**-2)*x**3
    a79 =  9.409838*(10**0)  + 1.522928*(10**0)*x
    a80 = -2.7110*(10**-1)   - 5.7560*(10**-1)*x   - 8.3800*(10**-2)*x**2
    a81 =  2.4930 *(10**0)   + 1.1475*(10**0)*x

    #Aplicamos condiciones sobre los valores de algunos coeficientes
    a75 = max(1.0, min(a75, 1.27))
    a75 = max(a75, 0.6355 - 0.4192*x)
    a76 = max(a76, -0.1015564 - 0.2161264*x - 0.05182516*x**2)
    a77 = max(-0.3868776 - 0.5457078*x - 0.1463472*x**2, min(0.0, a77))
    a78 = max(0.0, min(a78, 7.454 + 9.046*x))
    a79 = min(a79, max(2.0, -13.3 - 18.6*x))
    a80 = max(0.0585542, a80)
    a81 = min(1.5, max(0.4, a81))

    B   = a76 + a77*(1.0 - a78)**a79 #El valor de gamma cuando M = 1

    if a75 == 1.0:
        C = B
    else:
        C = a80

    ###########################################################################
    #Ahora calculamos gamma, pero la forma de calcularla depende de M

    if M <= 1.0: #Forma de calcular gamma cuando M <= 1
```

```
        gamma = a76 + a77*(M - a78)**a79

    elif 1.0 < M <= a75: #Forma de calcular gamma cuando 1 < M <= a75
        gamma = B + (a80 - B)*((M - 1.0)/(a75 - 1.0))**a81

    elif a75 < M < (a75 + 0.1): #Forma de calcular gamma cuando a75<M<(a75+0.1)
        gamma = C - 10.0*(M - a75)*C

    elif (a75 + 0.1) <= M: #Forma de calcular gamma cuando (a75 + 0.1) <= M
        gamma = 0


    return gamma

#Radio de la estrella en la secuencia principal
def R_MS(t, M, Z):
    '''
    Función que entrega el radio de una estrella en la secuencia principal.

    El radio entregado está en radios solares.
    Depende de:

    t: Tiempo de vida de la estrella en la secuencia principal (En Myr)
    Z: Metalicidad de la estrella
    M: Masa de la estrella (Masas solares)
    '''

    #Evaluamos todos los coeficientes y variables definidos con anterioridad
    r_ZAMS = R_ZAMS(M, Z)
    a_R = alpha_R(M, Z)
    b_R = beta_R(M, Z)
    g_R = gamma(M, Z)
    r_TMS = R_TMS(M, Z)
    dR = deltaR(M, Z)
    ta = tau(t, M, Z)
    ta1 = tau_1(t, M, Z)
    ta2 = tau_2(t, M, Z)

    #Evaluamos todo para obtener el valor del lado derecho de la ecuación
    exp = a_R*ta + b_R*ta**10 + g_R*ta**40 + (np.log10(r_TMS/r_ZAMS) - a_R -␣
↪b_R - g_R)*ta**3 - dR*(ta1**3 - ta2**3)

    #Hacemos que este valor eleve a un 10 y lo multiplicamos por R_ZAMS
    #(Esto sería despejar R_MS de la ecuación)
    R_MS = r_ZAMS*10**exp

    return R_MS
```

A modo de prueba, calculemos el radio del Sol con nuestra función:

[3]:
```
R_MS(4500, 1, 0.02)
```

[3]:
```
0.9826121223481187
```

0.98 radios solares: Un valor bastante razonable.

2 Luminosidad estelar

Ahora siguen las funciones que definen la luminosidad de una estrella en función de la masa, metalicidad y tiempo en la secuencia principal. Las funciones fueron sacadas de Hurley et al.(2000) también, con la excepción de L_{ZAMS}(L_ZAMS) que viene de Tout et al.(1996). Muchas funciones que se utilizan para calcular la luminosidad son las mismas que se utilizan para obtener el radio, así que no se volverán a definir.

[4]:
```
#Luminosidad de la estrella cuando entra en la secuencia principal
def L_ZAMS(M, Z):
    '''
    Función que entrega la luminosidad de una estrella cuando entra a la
    secuencia principal (Zero Age Main Sequence).

    La luminosidad entregada está en luminosidades solares.
    Depende de:

    Z: Metalicidad de la estrella
    M: Masa de la estrella (Masas solares)
    '''

    x = dseta(Z)     #Obtenemos dzeta a partir de la metalicidad

    # Calculamos los coeficientes en función de dseta
    alpha   = 0.39704170 -  0.32913574*x +  0.34776688*x**2 +  0.37470851*x**3␣
↪+ 0.09011915*x**4
    beta    = 8.52762600 - 24.41225973*x + 56.43597107*x**2 + 37.06152575*x**3␣
↪+ 5.45624060*x**4
    gamma   = 0.00025546 -  0.00123461*x -  0.00023246*x**2 +  0.00045519*x**3␣
↪+ 0.00016176*x**4
    delta   = 5.43288900 -  8.62157806*x + 13.44202049*x**2 + 14.51584135*x**3␣
↪+ 3.39793084*x**4
    epsilon = 5.56357900 - 10.32345224*x + 19.44322980*x**2 + 18.97361347*x**3␣
↪+ 4.16903097*x**4
    zeta    = 0.78866060 -  2.90870942*x +  6.54713531*x**2 +  4.05606657*x**3␣
↪+ 0.53287322*x**4
    eta     = 0.00586685 -  0.01704237*x +  0.03872348*x**2 +  0.02570041*x**3␣
↪+ 0.00383376*x**4
```

```
    #Calculamos la luminosidad en función de los coeficientes y la masa estelar
    L_ZAMS = (alpha*(M**5.5) + beta*(M**11))/(gamma + M**3 + delta*(M**5) +␣
↪epsilon*(M**7) + zeta*(M**8) + eta*(M**9.5))

    return L_ZAMS

#Luminosidad de la estrella al salir de la secuencia principal
def L_TMS(M, Z):
    '''
    Función que entrega la luminosidad de una estrella cuando sale de la
    secuencia principal.

    La luminosidad entregada está en luminosidades solares.
    Depende de:

    Z: Metalicidad de la estrella
    M: Masa de la estrella (Masas solares)
    '''

    x = dseta(Z)     #Obtenemos dzeta a partir de la metalicidad

    #Obtenemos los coeficientes a partir de dseta y sigma
    a11_p = 1.031538*(10**0)  - 2.434480*(10**-1)*x  + 7.732821*(10**0)*x**2  +␣
↪6.460705*(10**0)*x**3 + 1.374484*(10**0)*x**4
    a12_p = 1.043715*(10**0)  - 1.577474*(10**0)*x   - 5.168234*(10**0)*x**2  -␣
↪5.596506*(10**0)*x**3 - 1.299394*(10**0)*x**4
    a13   = 7.859573*(10**2)  - 8.542048*(10**0)*x   - 2.642511*(10**1)*x**2  -␣
↪9.585707*(10**0)*x**3
    a14   = 3.858911*(10**3)  + 2.459681*(10**3)*x   - 7.630093*(10**1)*x**2  -␣
↪3.486057*(10**2)*x**3 - 4.861703*(10**1)*x**4
    a15   = 2.888720*(10**2)  + 2.952979*(10**2)*x   + 1.850341*(10**2)*x**2  +␣
↪3.797254*(10**1)*x**3
    a16   = 7.196580*(10**0)  + 5.613746*(10**-1)*x  + 3.805871*(10**-1)*x**2 +␣
↪8.398728*(10**-2)*x**3
    a11   = a11_p*a14
    a12   = a12_p*a14

    ###############################################################################
    #Ahora calculamos L_TMS:

    L_TMS = (a11*(M**3) + a12*(M**4) + a13*(M**(a16 + 1.8)))/(a14 + a15*(M**5)␣
↪+ M**a16)


    return L_TMS
```

```
#Delta L
def deltaL(M, Z):
    '''
    Función que entrega deltaL.

    Es una cantidad adimensional.
    Depende de:

    Z: Metalicidad de la estrella
    M: Masa de la estrella (Masas solares)
    '''

    x = dseta(Z)     #Obtenemos dseta a partir de la metalicidad

    #Calculamos los coeficientes que dependen de dseta
    a34 = 1.910302*(10**-1) + 1.158624*(10**-1)*x + 3.348990*(10**-2)*x**2 + 2.
 ↪599706*(10**-3)*x**3
    a35 = 3.931056*(10**-1) + 7.277637*(10**-2)*x - 1.366593*(10**-1)*x**2 - 4.
 ↪508946*(10**-2)*x**3
    a36 = 3.267776*(10**-1) + 1.204424*(10**-1)*x + 9.988332*(10**-2)*x**2 + 2.
 ↪455361*(10**-2)*x**3
    a37 = 5.990212*(10**-1) + 5.570264*(10**-2)*x + 6.207626*(10**-2)*x**2 + 1.
 ↪777283*(10**-2)*x**3
    a33 = min(1.4, 1.5135 + 0.3769*x)
    a33 = max(0.6355 - 0.4192*x, max(1.25, a33))

    #Calculamos el valor B, que viene a ser deltaL cuando M = a33
    B   = min(a34/(a33**a35), a36/(a33**a37))

    #Obtenemos M_hook
    m   = M_hook(Z)

    ###########################################################################
    #Ahora calculamos deltaL, pero la forma de calcularla depende de M

    if M <= m: #Forma de calcular deltaR cuando M <= M_{hook}
        deltaL = 0.0

    elif m < M < a33: #Forma de calcular deltaR cuando M_{hook} < M < a33
        deltaL = B*((M - m)/(a33 - m))**0.4

    elif a33 <= M: #Forma de calcular deltaR cuando a33 <= M
        deltaL = min(a34/(M**a35), a36/(M**a37))

    return deltaL

#Alpha L
```

```
def alpha_L(M, Z):
    '''
    Función que entrega alpha_L.

    Es una cantidad adimensional.
    Depende de:

    Z: Metalicidad de la estrella
    M: Masa de la estrella (Masas solares)
    '''

    x = dseta(Z)      #Obtenemos dseta a partir de la metalicidad

    #Calculamos los coeficientes que dependen de dseta
    a45 = 2.321400*(10**-1) + 1.828075*(10**-3)*x - 2.232007*(10**-2)*x**2 - 3.
↪378734*(10**-3)*x**3
    a46 = 1.163659*(10**-2) + 3.427682*(10**-3)*x + 1.421393*(10**-3)*x**2 - 3.
↪710666*(10**-3)*x**3
    a47 = 1.048020*(10**-2) - 1.231921*(10**-2)*x - 1.686860*(10**-2)*x**2 - 4.
↪234354*(10**-3)*x**3
    a48 = 1.555590*(10**0)  - 3.223927*(10**-1)*x - 5.197429*(10**-1)*x**2 - 1.
↪066441*(10**-1)*x**3
    a49 = 9.7700*(10**-2)   - 2.3100*(10**-1)*x   - 7.5300*(10**-2)*x**2
    a50 = 2.4000*(10**-1)   + 1.8000*(10**-1)*x   + 5.9500*(10**-1)*x**2
    a51 = 3.3000*(10**-1)   + 1.3200*(10**-1)*x   + 2.1800*(10**-1)*x**2
    a52 = 1.1064*(10**0)    + 4.1500*(10**-1)*x   + 1.8000*(10**-1)*x**2
    a53 = 1.1900*(10**0)    + 3.7700*(10**-1)*x   + 1.7600*(10**-1)*x**2

    #Aplicamos condiciones sobre los valores de algunos coeficientes
    a49 = max(a49, 0.145)
    a50 = min(a50, 0.306 + 0.053*x)
    a51 = min(a51, 0.3625 + 0.062*x)
    a52 = max(a52, 0.9)

    if Z > 0.01:
        a52 = min(a52, 1.0)
    else:
        pass

    a53 = max(a53, 1.0)

    if Z > 0.01:
        a53 = min(a53, 1.1)
    else:
        pass

    #El valor de alpha_L cuando M = 2.0
```

```
    B = (a45 + a46*2.0**a48)/(2.0**0.4 + a47*2.0**1.9)

    ###########################################################################
    #Ahora calculamos alpha_L, pero la forma de calcularla depende de M

    if M < 0.50: #Forma de calcular alpha_L cuando M < 0.5
        alpha_L = a49

    elif 0.50 <= M < 0.7: #Forma de calcular alpha_L cuando 0.5 <= M <0.7
        alpha_L = a49 + 5.0*(0.3 - a49)*(M - 0.5)

    elif 0.7 <= M < a52: #Forma de calcular alpha_L cuando 0.7 <= M < a52
        alpha_L = 0.3 + ((a50 - 0.3)*(M - 0.7))/(a52 - 0.7)

    elif a52 <= M < a53: #Forma de calcular alpha_L cuando a52 <= M < a53
        alpha_L = a50 + ((a51 - a50)*(M - a52))/(a53 - a52)

    elif a53 <= M < 2.0: #Forma de calcular alpha_L cuando a53 <= M < 2.0
        alpha_L = a51 + ((B - a51)*(M - a53))/(2.0 - a53)

    elif 2.0 <= M: #Forma de calcular alpha_L cuando 2.0 <= M
        alpha_L = (a45 + a46*M**a48)/(M**0.4 + a47*M**1.9)


    return alpha_L

#Beta L
def beta_L(M, Z):
    '''
    Función que entrega beta_L.

    Es una cantidad adimensional.
    Depende de:

    Z: Metalicidad de la estrella
    M: Masa de la estrella (Masas solares)
    '''

    x = dseta(Z)     #Obtenemos dseta a partir de la metalicidad

    #Calculamos los coeficientes que dependen de dseta
    a54 = 3.855707*(10**-1) - 6.104166*(10**-1)*x + 5.676742*(10**0)*x**2 + 1.
 ↪060894*(10**1)*x**3 + 5.284014*(10**0)*x**4
    a55 = 3.579064*(10**-1) - 6.442936*(10**-1)*x + 5.494644*(10**0)*x**2 + 1.
 ↪054952*(10**1)*x**3 + 5.280991*(10**0)*x**4
    a56 = 9.587587*(10**-1) + 8.777464*(10**-1)*x + 2.017321*(10**-1)*x**2
    a57 = min(1.4, 1.5135 + 0.3769*x)
```

```
    a57 = max(0.6355 - 0.4192*x, max(1.25, a57))

    B   = max(0.0, a54 - a55*a57**a56)   #El valor de beta_L cuando M = a57

    ###########################################################################
    #Ahora calculamos beta_L, pero la forma de calcularla depende de M

    beta_L = max(0.0, a54 - a55*M**a56)

    if M > a57 and beta_L > 0:

        beta_L = max(0.0, B - 10.0*(M - a57)*B)

    return beta_L

#eta
def eta(M, Z):
    '''
    Función que calcula el exponente eta.

    Es una cantidad adimensional.
    Depende de:

    Z: Metalicidad de la estrella
    M: Masa de la estrella (Masas solares)
    '''
    if Z <= 0.0009:
        if M <= 1.0:
            eta = 10
        elif 1.0 < M < 1.1:
            eta = 100*M - 90
        elif 1.1 <= M:
            eta = 20
    else:
        eta = 10

    return eta

#Luminosidad de la estrella en la secuencia principal
def L_MS(t, M, Z):
    '''
    Función que entrega la luminosidad de una estrella en la secuencia
    principal.

    La luminosidad entregada está en luminosidades solares.
    Depende de:
```

```
    t: Tiempo de vida de la estrella en la secuencia principal (En Myr)
    Z: Metalicidad de la estrella
    M: Masa de la estrella (Masas solares)
    '''

    #Evaluamos todos los coeficientes y variables definidos con anterioridad
    l_ZAMS = L_ZAMS(M, Z)
    a_L = alpha_L(M, Z)
    b_L = beta_L(M, Z)
    et = eta(M, Z)
    l_TMS = L_TMS(M, Z)
    dL = deltaL(M, Z)
    ta = tau(t, M, Z)
    ta1 = tau_1(t, M, Z)
    ta2 = tau_2(t, M, Z)

    #Evaluamos todo para obtener el valor del lado derecho de la ecuación
    exp = a_L*ta + b_L*ta**et + (np.log10(l_TMS/l_ZAMS) - a_L - b_L)*ta**2 -␣
↪dL*(ta1**2 - ta2**2)

    #Hacemos que este valor eleve a un 10 y lo multiplicamos por L_ZAMS
    #(Esto sería despejar L_MS de la ecuación)
    L_MS = l_ZAMS*10**exp

    return L_MS
```

Nuevamente, por fines de ejemplificar, vamos a ver que valor nos entrega con parámetros solares.

[5]:
```
L_MS(4500, 1, 0.02)
```

[5]:
```
0.9501363604365745
```

0.95 luminosidades estelares también es un resultado razonable.

3 Parámetros vientos estelares

Ahora vienen las funciones que determinan los parámetros de vientos estelares, siendo de especial utilidad para esta investigación su velocidad y densidad, para luego obtener la presión de arrastre. Todas estas funciones vienen de las publicación de dos partes Johnstone et al.(2015a,b).

[6]:
```
#Temperatura base de vientos solares
def T_0(M, R, output='mean'):
    '''
    Función que calcula la temperatura base de los vientos estelares de una
    estrella de masa M y radio R.

    La temperatura entregada está en MK (Mega Kelvin).
    Depende de:
```

```
    M: Masa de la estrella (En Masas solares)
    R: Radio de la estrella (En Radios solares)
    output: String que nos dice si queremos que la temperatura entregada
            sea la de los vientos lentos ('slow'), la de los vientos
            rapidos ('fast') o un promedio de sus temperaturas ('mean').
    '''
    #Primero vamos a definir algunos parámetros y constantes

    #Constante gravitacional (en (N*m**2)/kg**2)
    G = cons.G.value

    #Masa de un proton (en kg)
    m_p = cons.m_p.value

    #Masa promedio por partícula en unidades de m_p (adimensional)
    mu = 0.6

    #Indice adiabático (adimensional)
    gamma = 5/3

    #Constante de Boltzmann (J/K)
    k_B = cons.k_B.value

    #Cociente entre la velocidad base del sonido y la de escapeen la superficie
    #para vientos lentos (adimensional)
    cv_slow = 0.329

    #Cociente entre la velocidad base del sonido y la de escapeen la superficie
    #para vientos rápidos (adimensional)
    cv_fast = 0.478

    #Ahora vamos a convertir las unidades de medida de M y R al SI
    m = M*cons.M_sun.value #De masas solares a kg
    r = R*cons.R_sun.value #De radios solares a m

    #Calculamos la temperatura base para cada tipo de viento, lo multiplicamos
    #por 10^-6 para entregar la temperatura en MK
    T_0_slow = ((2*G*mu*m_p)/(gamma*k_B))*(cv_slow**2)*(m/r)*10**-6
    T_0_fast = ((2*G*mu*m_p)/(gamma*k_B))*(cv_fast**2)*(m/r)*10**-6

    #Dependiendo de lo que se ponga en output, se entrega la temperatura base
    #de los vientos lentos, rápidos o un promedio
    if output == 'mean':
        return ((T_0_slow + T_0_fast)/2)

    elif output == 'slow':
```

```
        return T_0_slow

    elif output == 'fast':
        return T_0_fast

#Rapidez de los vientos a 1 AU
def v_1AU(M, R):
    '''
    Función que calcula la rapidez de los vientos ubicados a 1 AU de una
    estrella de masa M y radio R.

    La rapidez entregada está en km/s.
    Depende de:

    M: Masa de la estrella (En Masas solares)
    R: Radio de la estrella (En Radios solares)
    '''

    #Calculamos la rapidez
    v = 73.39 + 224.14*T_0(M, R) - 11.28*T_0(M, R)**2 + 0.28*T_0(M, R)**3

    return v

#Cambio de rapidez de los vientos
def dvdr(M, R):
    '''
    Función que calcula el cambio de la rapidez de los vientos al moverse
    radialmente respecto a una estrella de masa M y radio R.

    El cambio de la rapidez entregada está en km/(s*R_sol).
    Depende de:

    M: Masa de la estrella (En Masas solares)
    R: Radio de la estrella (En Radios solares)
    '''

    #Calculamos el cambio de rapidez
    dvdr = 0.19 + 0.066*T_0(M, R) - 0.0035*T_0(M, R)**2 + (9.97*10**-5)*T_0(M,␣
 ↪R)**3



    return dvdr

#Rapidez de los vientos estelares
def v_sw(r, M, R):
    '''
    Función que entrega la rapidez de los vientos estelares de una estrella de
```

```
    masa M y radio R cuando se está a una distancia r de la estrella.

    La rapidez entregada está en km/s.
    Depende de:

    r: Distancia hacia la estrella (En AU)
    M: Masa de la estrella (En Masas solares)
    R: Radio de la estrella (En Radios solares)
    '''

    #Primero vamos a cambiar las unidades de dv/dr
    dvdr_au = dvdr(M, R)/((cons.R_sun.to(u.AU)).value)  #De km/(s*R_sol) a km/
↪(s*AU)

    #Calculamos la rapidez
    v = v_1AU(M, R) + (r - 1)*dvdr_au

    return v

#Rapidez angular estelar
def omega(t, M):
    '''
    Función que calcula la rapidez angular de una estrella que lleva t tiempo
    en la secuencia principal y una masa M.

    La rapidez angular entregada está en rapideces angulares solares.
    Depende de:

    t: Tiempo en la secuencia principal  (En Gyr)
    M: Masa de la estrella (En Masas solares)
    '''

    #Calculamos la rapidez angular
    omega_s = (1.667*M**0.652)*(t/2)**-0.566

    return omega_s

#Perdida de masa estelar
def M_punto(t, M, R):
    '''
    Función que obtiene la perdida de masa por unidad de tiempo de una estrella
    de masa M, radio R y que lleva t tiempo en la secuencia principal.

    La perdida de masa por unidad de tiempo entregada está en
    "perdidas de masa por unidad de tiempo solares".
    Depende de:
```

```
    t: Tiempo en la secuencia principal  (En Gyr)
    M: Masa de la estrella (En Masas solares)
    R: Radio de la estrella (En Radios solares)
    '''

    #Calculamos la perdida de masa por unidad de tiempo
    Mpunto = (R**2)*(omega(t, M)**1.33)*(M**-3.36)

    return Mpunto

#Densidad de los vientos
def rho_SW(t, r, M, R):
    '''
    Función que nos entrega la densidad del viento estelar ubicado a una
    distancia r de una estrella de masa M, radio R, que lleva un tiempo t en la
    secuencia principal.

    La densidad entregada está en kg/m**3.
    Depende de:

    t: Tiempo en la secuencia principal  (En Gyr)
    r: Distancia hacia la estrella (En AU)
    M: Masa de la estrella (En Masas solares)
    R: Radio de la estrella (En Radios solares)
    '''

    #Primero vamos a calcular M_punto y v_SW, y pasaremos sus unidades junto
    #con la de r a unidades del SI

    #M_punto lo calculamos y lo pasamos a kg/s
    M_punto_si = M_punto(t, M, R)*(((1.4*1e-14)*cons.M_sun)/(1*u.yr.to(u.s))).
 ↪value

    #r lo pasamos a metros
    r_si = r*cons.au.value

    #v_SW lo pasamos a m/s (recordar que v_SW entrega v en km/s)
    v_si = v_sw(r, M, R)*10**3

    #Calculamos la densidad del viento estelar
    rho = M_punto_si/(4*np.pi*v_si*r_si**2)

    return rho

#Presión de los vientos
def P_SW(t, r, M, R):
    '''
```

```
    Función que nos entrega la presión del viento estelar a una distancia
    r de una estrella de masa M, radio R, y que lleva un tiempo t en
    la secuencia principal.

    La presión que nos entrega está en pascales.
    Depende de:

    t: Tiempo en la secuencia principal  (En Gyr)
    r: Distancia hacia la estrella (En AU)
    M: Masa de la estrella (En Masas solares)
    R: Radio de la estrella (En Radios solares)
    '''

    #Primero tomamos la velocidad y pasamos sus unidades a unidades del SI
    v_si = v_sw(r, M, R)*10**3

    #Calculamos la presión
    P = rho_SW(t, r, M, R)*(v_si)**2

    return P
```

4 Parámetros remanente de supernova (Fase Sedov-Taylor)

Las siguientes funciones son parámetros del remanente de supernova, como se está considerando un remanente en fase de Sedov-Taylor, una fase caracterizada por una expansión adiabática, las expresiones para su velocidad y densidad son muy conocidas, las usadas aquí están también presentes en Fields et al.(2008).

[7]:
```
#Grosor del remanente de supernova
def DeltaR_SNR(d):
    '''
    Función que entrega el grosor del remanente de supernova en fase de
    Sedov-Taylor en expansión.

    El grosor entregado está en parsec.
    Depende de:

    d: Distancia del remanente a el punto de origen de la supernova (pc)
    '''
    #Constante adiabática
    gamma = 5/3

    #Calculamos el valor de DeltaR
    DeltaR_SNR = ((gamma - 1)*d)/(3*(gamma + 1))

    return DeltaR_SNR
```

```
#Densidad del remanente
def rho_SNR(d, n=0.1, tipo='Ia'):
    '''
    Función que entrega la densidad del remanente de supernova en fase de
    Sedov-Taylor.

    La densidad entregada está en kg/m^3
    Depende de:

    n: Número de partículas por centímetro cúbico en el medio
       interestelar (cm^-3)
    tipo: String que nos dice si la supernova con la que se está tratando
          es de tipo Ia o II.
    '''
    #Constante adiabática (adimensional)
    gamma = 5/3

    #Calculamos la densidad del medio interestelar en kg/m^3
    rho_ISM = n*(cons.m_p.value)*(10**6)

    #Calculamos la densidad del remanente de supernova en kg/m^3
    rho_SNR = ((gamma + 1)/(gamma - 1))*rho_ISM

    #Aplicamos condiciones sobre rho_SNR

    #Calculamos DeltaR en metros
    dr = DeltaR_SNR(d)*cons.pc.value
    #Calculamos d en metros
    r = d*cons.pc.value
    #Calculamos el volumen del remanente en m^3
    V = (4/3)*np.pi*((r+dr/2)**3 - (r-dr/2)**3)
    #Calculamos la masa del remanente (shell) y la pasamos a M_sol
    M_sh = (rho_SNR*V)/(cons.M_sun.value)

    #Si la supernova es de tipo Ia y el remanente supera una masa de 1.4 Msol,
    #La densidad se redefine rho_SNR de tal manera que la masa sea 1.4 Msol.
    #SI la supernova es de tipo II, el proceso es el mismo, pero para masa de
    #5 Msol.
    if tipo == 'Ia':
        if M_sh < 1.4:
            rho_SNR = (1.4*cons.M_sun.value)/V #Msol a kg y se divide por V
        else:
            pass
    elif tipo == 'II':
        if M_sh < 5.0:
            rho_SNR = (5.0*cons.M_sun.value)/V #Msol a kg y se divide por V
        else:
```

```
            pass

    return rho_SNR

#Rapidez del remanente
def v_SNR(E, d, n=0.1):
    '''
    Función que nos entrega la rapidez del remanente de supernova en fase de
    Sedov-Taylor a una distancia d, considerando que transporta una energía E
    y viaja por el medio interestelar de densidad n.

    La rapidez entregada está en km/s.
    Depende de:

    E: Energía liberada por la supernova (En ergios)
    d: Distancia a la que ocurre la supernova (En parsecs)
    n: Número de partículas por centímetro cúbico en el medio
       interestelar (cm^-3)
    '''
    #Pasamos la energía a joules
    E = E*10**-7
    #Pasamos la distancia a metros
    d = d*cons.pc.value
    #Calculamos la densidad del ISM en kg/m^3
    rho_ISM = n*(cons.m_p.value)*(10**6)
    #Constante adiabática (adimensional)
    gamma = 5/3
    #Factor numperico adimensional que depende de gamma
    beta = 1.1517

    #Calculamos la rapidez en m/s
    v = (4/5)*((beta**2.5)/(gamma + 1))*(E/(rho_ISM*d**3))**0.5

    #Agregamos una condición para que v no sea mayor a 10.000 km/s
    if v > 10000000:
        v = 10000000
    else:
        pass

    return v*10**-3 #Entregamos la rapidez en km/s

#Presión del remanente
def P_SNR(E, d, n = 0.1, tipo='Ia'):
    '''
    Función que nos entrega la presión del remanente de una supernova en fase
    de Sedov-Taylor, la cuál ocurrió a una distancia d liberando una energía E.
```

```
    La presión que entrega está en pascales.
    Depende de:

    E: Energía liberada por la supernova (En ergios)
    d: Distancia a la que ocurre la supernova (En parsecs)
    n: Número de partículas por centímetro cúbico en el medio
       interestelar (cm^-3)
    tipo: String que nos dice si la supernova con la que se está tratando
          es de tipo Ia o II.
    '''
    #Calculamos la densidad del remanente en kg/m^3
    rho = rho_SNR(d, n, tipo)
    #Calculamos la rapidez del remanente en m/s
    v = v_SNR(E, d, n)*10**3
    #Constante adiabática (adimensional)
    gamma = 5/3
    #Calculamos la densidad del medio interestelar en kg/m^3
    rho_ISM = n*(cons.m_p.value)*(10**6)

    #Calculamos la "ram pressure" del remanente en pascales
    P_ram = rho*v**2
    #Calculamos la presión termica del remanente en pascales
    P_term = ((gamma + 1)/2)*rho_ISM*v**2

    return P_ram + P_term #Entregamos la suma de ambas presiones
```

5 Radio de equilibrio

Finalmente llegamos a la parte entretenida del mambo. Ahora vamos a definir dos funciones para calcular el radio de equilibrio, la diferencia entre ambas es que una deja el radio de la estrella como parámetro libre, mientras que la otra utiliza reemplaza el valor de radio obtenido con las funciones definidas al principio, haciendo que el parámetro de radio ya no sea libre, pero introduciendo el parámetro de metalicidad, el cual es más fácil de estimar/obtener.

[8]:
```
#Radio de equilibrio
def R_eq(t, M, R, E, d, n=0.1, tipo='Ia', a=0.0001, b=1000):
    '''
    Función que nos entrega el radio de equilibrio en el cual la presión del
    remanente de una supernova en la fase de Sedov-Taylor y la presión de
    los vientos estelares de una estrella, considerando que la supernova␣
 ↪ocurrió
    a una distancia d liberando una energía E y que la estrella tiene una
    masa M, radio R y lleva un tiempo t en la secuencia principal.

    El radio entregado está en unidades astronómicas.
    Depende de:
```

```
    t: Tiempo en la secuencia principal  (En Gyr)
    M: Masa de la estrella (En Masas solares)
    R: Radio de la estrella (En Radios solares)
    E: Energía liberada por la supernova (En ergios)
    d: Distancia a la que ocurre la supernova (En parsecs)
    n: Número de partículas por centímetro cúbico en el medio
       interestelar (cm^-3)
    tipo: String que nos dice si la supernova con la que se está tratando
          es de tipo Ia o II (Trabajaremos con Ia preferentemente).
    a y b: Números que representan los límites sobre los que la función de
           optimización busca la raiz
    '''

    #Esta función genera otra función (función anidada) que depende de una
    #variable r, que vendría a ser la distancia a la que se encuentra la
    #estrella. Esta función ya tiene reemplazados los parámetros de la primera
    #función, por esto mismo podemos buscar el valor de r que minimiza la
    #función anidada utilizando el método de Brent con la función optimize de
    #scipy. De esta forma se obtiene el valor de r que hace que se igualen las
    #presiones del remanente y de los vientos

    #Generamos la función
    def fun(r):

        f = P_SW(t, r, M, R)-P_SNR(E, d, n, tipo)

        return f

    #Buscamos el valor de r que es raiz y lo entregamos
    sol = optimize.root_scalar(fun, bracket=[a, b], method='brentq')

    return sol.root

#Radio de equilibrio calculando el radio estelar en función de la masa,
#metalicidad y tiempo en la secuencia principal
def R_eqM(t, M, Z, E, d, n=0.1, tipo='Ia', a = 0.0001, b = 1000):
    '''
    Función que calcula el radio de equilibrio entre la presión de los vientos
    estelares y la presión de una supernova en fase de Sedov-Taylor. Se
    calculan las presiones considerando una estrella de masa M, metalicidad Z
    y que lleva un tiempo t en la secuencia principal, además de una supernova
    que ocurre a una distancia d y libera una energía E.

    Esta función calcula el radio directamente de la igualdad de preiones. El
    radio entregado está en AU.
    Depende de:
```

```
    t: Tiempo de vida de la estrella en la secuencia principal (En Gyr)
    M: Masa de la estrella (En Masas solares)
    Z: Metalicidad de la estrella (adimensional)
    E: Energía liberada por la supernova (En ergios)
    d: Distancia a la que ocurre la supernova (En parsecs)
    n: Número de partículas por centímetro cúbico en el medio
       interestelar (cm^-3)
    tipo: String que nos dice si la supernova con la que se está tratando
          es de tipo Ia o II.
    a y b: Números que representan los límites sobre los que la función de
           optimización busca la raiz
    '''

    #Calcula el radio de la estrella, el tiempo se multiplica por mil para
    #pasar el tiempo de Gyr a Myr (R_MS depende de t en Myr)
    R = R_MS(t*1000, M, Z)

    #Calcula el radio de equilibrio utilizando los parámetros entregados
    #y el radio
    Req = R_eq(t, M, R, E, d, n, tipo, a, b)

    return Req
```

A modo de prueba, calculemos el radio de equilibrio para una supernova de tipo Ia que explotó a una distancia de 8 pc, liberando una energía de 10^{51} erg, y considerando que estamos en un sistema estelar con parámetros solares (4.5 Gyr en la secuencia principal, una masa de 1 $M_\odot$ y una metalicidad de 0.02)

[9]:
```
R_eqM(4.5, 1, 0.02, 1e51, 8)
```

[9]:
```
0.5455028671869855
```

O sea, si ocurriese una supernova de tipo Ia a 8 parsecs, la heliopausa retorcedería hasta 0.54 unidades astronómicas. O sea, el remanente llegaría a nuestro planeta :(

6 Zona habitable

Ahora vamos a utilizar las expresiones definidas en Kopparapu et al.(2013) donde se definen radios de la zona habitable, considerando como límite externo la distancia donde la cantidad de agua evaporada en la estratósferaes muy alta, llevando a pérdidas de hidrógeno en la atmósfera y como límite externo la distancia donde la abundancia de CO_2 en la atmósfera genera mucho efecto invernadero, enfriando el planeta. Ambas funciones definidas paramétricamente en función de la luminosidad y el radio de la estrella.

[10]:
```
#Temperatura efectiva de la estrella
def T_eff(L, R):
    '''
    Función que calcula la temperatura efectiva de una estrella de radio R y
```

```
    luminosidad L utilizando relaciones de cuerpo negro.

    La temperatura entregada está en Kelvin.
    Depende de:

    L: Luminosidad de la estrella (Luminosidades solares)
    R: Radio de la estrella (Radios estelares)
    '''
    #Constante de Stefan-Boltzmann (W/m^2*K^4)
    sigma = cons.sigma_sb.value

    #Pasamos R de Rsol a m
    R = R*cons.R_sun.value

    #Pasamos L de Lsol a W
    L = L*cons.L_sun.value

    #Calculamos T_eff en K
    T = (L/(4*np.pi*(R**2)*sigma))**(1/4)

    return T

#Flujo solar efectivo
def S_eff(L, R, tipo):
    '''
    Función que calcula el flujo estelar efectivo en los bordes de la zona
    habitable para un planeta como la Tierra que está orbitando una estrella
    de luminosidad L y radio R.

    El flujo entregado está parametrizado al flujo solar a la Tierra actual
    (1360 W/m^2).
    Depende de:

    L: Luminosidad de la estrella (Luminosidades solares)
    R: Radio de la estrella (Radios estelares)
    tipo: String que determina si el flujo entregado es para el límite interno
          (min) o el exterior (max) de la zona habitable.
    '''
    #Calculamos la variable T_x, que vendría a ser la diferencia entre
    #T_eff y 5780 K
    T_x = T_eff(L, R) - 5780

    #Se definen las constantes S_o, a, b, c y d, las cuales dependen del 'tipo'
    #de flujo que se quiera calcular
    if tipo == 'min':
        S_o, a, b, c, d = 1.014, 8.1774*10**-5, 1.7063*10**-9, -4.3241*10**-12,␣
↪-6.6462*10**-16
```

```
    elif tipo == 'max':
        S_o, a, b, c, d = 0.3438, 5.8942*10**-5, 1.6558*10**-9, -3.
↪0045*10**-12, -5.2983*10**-16

    #Calculamos el flujo efectivo
    S = S_o + a*T_x + b*T_x**2 + c*T_x**3 + d*T_x**4

    return S

#Radios de la zona habitable
def R_HZ(L, R):
    '''
    Función que calcula las distancias desde una estrella hasta los límites de
    la zona habitable. Esto se hace considerando la zona habitable para un
    planeta como la Tierra y una estrella de radio R y luminosidad L.

    La función entrega una tupla donde el primer valor es el radio del límite
    interno de la zona habitable y el segundo valor el del límite externo,
    ambos valores en AU.
    Depende de:

    L: Luminosidad de la estrella (Luminosidades solares)
    R: Radio de la estrella (Radios estelares)
    '''
    #Calculamos el flujo del límite interno
    S_min = S_eff(L, R, tipo='min')

    #Calculamos el radio del límite interno
    R_min = (L/S_min)**0.5

    #Calculamos el flujo del límite externo
    S_max = S_eff(L, R, tipo='max')

    #Calculamos el flujo del límite externo
    R_max = (L/S_max)**0.5

    return R_min, R_max

#Radios de la zona habitable calculando la luminosidad y radio estelares en
#función de la masa, metalicidad y tiempo en la secuencia principal
def R_HZM(t, M, Z):
    '''
    Función que calcula las distancias desde una estrella hasta los límites de
    la zona habitable. Esto se hace considerando la zona habitable para un
    planeta como la Tierra y una estrella de masa M, metalicidad Z y que lleva
    un tiempo t en la secuencia principal.
```

```
    La función entrega una tupla donde el primer valor es el radio del límite
    interno de la zona habitable y el segundo valor el del límite externo,
    ambos valores en AU.
    Depende de:

    t: Tiempo de vida de la estrella en la secuencia principal (En Gyr)
    Z: Metalicidad de la estrella
    M: Masa de la estrella (Masas solares)
    '''
    #Utiliza t, M y Z para calcular la luminosidad y el radio de la estrella
    L = L_MS(t*10**3, M, Z)
    R = R_MS(t*10**3, M, Z)

    #Calcula los radio límite de la zona habitable en función de L y R
    R_HZm = R_HZ(L, R)

    return R_HZm
```

A modo de prueba, vamos a ver lo límites de la zona habitable de nuestra estrella utilizando los parámetros solares ya usados con anterioridad.

[11]:
```
R_HZM(4.5, 1, 0.02)
```

[11]:
```
(0.969213711237831, 1.6668680258661375)
```

O sea que la zona habitable de nuestra estrella está entre 0.97 y 1.67 unidades astronómicas. Con estos valores se puede concluir que la Tierra en efecto está en la zona habitable del Sol, y que Marte (1.52 AU) también se encuentra, solo que ambos están cercanos a límites opuestos.

7 Erosión atmosférica

Finalmente vienen las funciones de erosión atmosférica, las cuales se obtuvieron de Zendejas et al.(2010). Estas funciones están pensadas para presiones de arrastre de vientos solares, pero nosotros vamos a reemplazarle valores asociados a nuestro remanente de supernova. Primero definimos una función que nos entrega la masa atmosférica del planeta en función del tiempo que lleva pasando el remanente por el planeta, pero como llegamos que el cambio era tan pequeño, decidimos crear una función que en vez de decirnos cuál es la masa atmosférica del planeta, nos diga cuanta masa ha perdido, o sea un $\Delta M_{atm}(t)$ en vez de $M_{atm}(t)$.

[12]:
```
#Tasa de erosión atmosférica
def M_punto_atm(E, d, R_p, alpha=0.03, n=0.1, tipo='Ia'):
    '''
    Función que calcula la tasa de erosión atmosférica ocasionada por el
    remanente de una supernova en fase de Sedov-Taylor que ocurrió a una
    distancia d liberando una energía E sobre un planeta 'earth-like' de
    radio R_p.

    La tasa entregada está en kg/s.
```

```
    Depende de:

    E: Energía liberada por la supernova (En ergios)
    d: Distancia a la que ocurre la supernova (En parsecs)
    R_p: Radio del planeta (Radios terrestres)
    alpha: Coeficiente de arrastre del planeta (adimensional)
    n: Número de partículas por centímetro cúbico en el medio
       interestelar (cm^-3)
    tipo: String que nos dice si la supernova con la que se está tratando
          es de tipo Ia o II.
    '''
    #Obtenemos la densidad del remanente en kg/m^3
    rho = rho_SNR(d, n, tipo)
    #Obtenemos la rapidez del remanente en m/s
    v = v_SNR(E, d, n)*10**3
    #Pasamos el radio del planeta a metros
    R_p = R_p*cons.R_earth.value

    #Calculamos la tasa y la entregamos
    M_punto=2*np.pi*alpha*(R_p**2)*rho*v

    return M_punto

#Tiempo de cruce de supernova
def t_cross(E, d, n=0.1):
    '''
    Función que calcula el tiempo que transcurre mientras que un remanente de
    supernova en fase de Sedov-Taylor con energía E y que ocurrió a una
    distancia d pasa por un planeta.

    El tiempo entregado está en segundos.
    Depende de:

    E: Energía liberada por la supernova (En ergios)
    d: Distancia a la que ocurre la supernova (En parsecs)
    n: Número de partículas por centímetro cúbico en el medio
       interestelar (cm^-3)
    '''
    #Calculamos el grosor del remanente y lo pasamos a metros
    dR = DeltaR_SNR(d)*cons.pc.value
    #Calculamos la rapidez del remanente y lo pasamos a m/s
    v = v_SNR(E, d, n)*10**3

    #Entregamos el cociente entre el grosor y la velocidad
    return dR/v

#Masa atmosférica del planeta earth-like mientras pasa el remanente
```

```
def M_atm(t, E, d, R_p, M_p, P_0, alpha=0.03, n=0.1, tipo='Ia'):
    '''
    Función que entrega la masa de la atmósfera de un planeta a medida que
    un remanente de supernova en fase de Sedov-Taylor (con Energía E y
    distancia d) lleva pasando un tiempo t por un planeta 'earth-like'
    con radio R_p y masa M_p.

    La masa entregada está en kg.
    Depende de:

    t: Tiempo que lleva pasando el remanente (segundos)
    E: Energía liberada por la supernova (En ergios)
    d: Distancia a la que ocurre la supernova (En parsecs)
    R_p: Radio del planeta (Radios terrestres)
    M_p: Masa del planeta (Masas terrestres)
    P_0: Presión atmosférica del planeta antes de que pasase el remanente (atm)
    alpha: Coeficiente de arrastre del planeta (adimensional)
    n: Número de partículas por centímetro cúbico en el medio
       interestelar (cm^-3)
    tipo: String que nos dice si la supernova con la que se está tratando
          es de tipo Ia o II.
    '''
    #Calculamos la tasa de erosión atmosférica en kg/m
    Mpunto = M_punto_atm(E, d, R_p, alpha, n, tipo)
    #Pasamos el radio del planeta a metros
    R_p = R_p*cons.R_earth.value
    #Pasamos la masa del planeta a kg
    M_p = M_p*cons.M_earth.value
    #Pasamos la presión atmosférica del planeta a pascales
    P_0 = P_0*cons.atm.value #atm a Pa
    #Constante gravitacional en (N*m**2)/kg**2
    G = cons.G.value

    #Calculamos la masa atmosférica inicial en kg
    Matm0 = (4*np.pi*(R_p**4)*P_0)/(G*M_p)

    #Hacemos que para tiempos mayores a t_cross, la masa sea igual a la
    #masa cuando t=t_cross
    if t > t_cross(E, d, n):
        t = t_cross(E, d, n)
    else:
        pass

    #Obtenemos la masa atmosférica y la entregamos
    Matm = -Mpunto*t + Matm0

    return Matm
```

```
#Pérdida de masa atmosférica mientras pasa el remanente
def dM_atm(t, E, d, R_p, alpha=0.03, n=0.1, tipo='Ia'):
    '''
    Función que calcula el cambio de masa atmosférica de un planeta
    'earth-like' de radio R_p cuando el remanente de supernova en fase de
    Sedov-Taylor (con energía E y distancia d) lleva un tiempo t pasando por
    el planeta.

    El cambio de masa entregado está en kg.
    Depende de:

    t: Tiempo que lleva pasando el remanente (segundos)
    E: Energía liberada por la supernova (En ergios)
    d: Distancia a la que ocurre la supernova (En parsecs)
    R_p: Radio del planeta (Radios terrestres)
    alpha: Coeficiente de arrastre del planeta (adimensional)
    n: Número de partículas por centímetro cúbico en el medio
       interestelar (cm^-3)
    tipo: String que nos dice si la supernova con la que se está tratando
          es de tipo Ia o II.
    '''
    #Calculamos la tasa de erosión en kg/s
    Mpunto=M_punto_atm(E, d, R_p, alpha, n, tipo)

    #Hacemos que para tiempos mayores a t_cross, la masa perdida sea igual a la
    #perdida cuando t=t_cross
    if t > t_cross(E, d, n):
        t = t_cross(E, d, n)
    else:
        pass

    #Multiplicamos la tasa por el tiempo t (en s) y lo entregamos
    return Mpunto*t
```