**SS.py**

**import** numpy **as** np  
**import** function **as** f  
**from** timeit **import** \*  
  
res = open( **'resultN='** + str( f.N ) + **'.txt'**, **'w'** )  
  
**for** dt **in** f.delta\_t\_mass:  
  
 start = default\_timer()  
  
 f.start\_values()  
  
 *#print( f.u )* Time = f.cycle\_while(f.u, f.r, f.V, f.E, f.P, dt)  
  
 stop = default\_timer()  
 total\_time = stop - start  
  
 *# output running time in a nice format.* mins, secs = divmod(total\_time, 60)  
 hours, mins = divmod(mins, 60)  
  
 **if** dt == f.delta\_t:  
  
 Legend = [str(Time[i]) + **'s' for** i **in** np.arange(0, len( list(f.uy) ), 1)] *# легенда  
  
 # график скорости* f.plot\_graph(f.rx, f.uy, **'R'**, **'U'**, **'U(R)'**, Legend, **False**)  
  
 *# график плотности* f.plot\_graph(f.rx, f.Rhoy, **'R'**, **'rho'**, **'rho(R)'**, Legend, **True**)  
  
 *# график энергии* f.plot\_graph(f.rx, f.Ey, **'R'**, **'E'**, **'E(R)'**, Legend, **True**)  
  
 res.write( **'N ='** + str(f.N) + **' time work ='** + str( hours ) + **'h '** + str( mins ) + **'m '** + str( round(secs, 1) ) + **'sec '** + **'delta\_t = '** + str( dt) + **'\n'** )  
  
 *#print( 'N=',f.N, 'time work='+str( hours ) + 'h ' + str( mins ) + 'm ' + str( round(secs, 1) ) + 'sec','delta\_t=',dt )*res.close()

**Function.py**

**import** numpy **as** np  
**import** matplotlib.pyplot **as** plt  
**from** numba **import** jit  
  
N = 2000  
N\_mass = np.array( [ 4 \* 10 \*\* ( i ) **for** i **in** np.arange(0, 3, 1) ] )  
eps = 6.0 \* 1.0e-3  
delta\_t = 1.0e-4  
delta\_t\_mass = np.array( [ 1.0e-8 \* 10 \*\* ( i ) **for** i **in** np.arange( 0, 5, 1 )] )  
delta\_m = 1.0 / N  
gamma = 5.0 / 3.0  
R\_limit = 120.0  
  
*#характерные параметры*T0 = 3500.0 *# K, Начальная температура*E0 = 3.15 \* 10e9 *#удельная энергия эрг/г*ro0 = 5.45 \* 10e-4 *# начальная плотность, г/см3*P0 = 1.15 \* 10e6 *# Начальное давление эрг/см3*V = ( 3 \* ro0 )  
  
*# Характерные параметры*Rx = 100.0 *# Rx=R0*Ux = 5.6 \* 10e4 *#np.sqrt(E0/M), среднемассовая скорость*tx = 0.847\*10e-3 *#Rx/Ux, sec*rox = ( 1 / 3.0 \* ro0 )  
mx = ( rox \* ( Rx \*\* 3 ) / 3.0 )  
Px = ( ( 1.0 / 3 ) \* ro0 \* E0 )  
Ex = 3.15 \* 10e9  
  
u = np.array([])  
r = np.array([])  
ro = np.array([])  
V = np.array([])  
P = np.array([])  
E = np.array([])  
  
uy = np.array([])  
rx = np.array([])  
Rhoy = np.array([])  
Ey = np.array([])  
k = 0  
  
**def** start\_values():  
  
 **global** u, r, ro, V, P, E  
 **global** uy, rx, Rhoy, Ey, k  
 *# массивы для хранения массивов для графиков* uy = np.zeros( ( 3, N ) )  
 rx = np.zeros( ( 3, N ) )  
 Rhoy = np.zeros( ( 3, N ) )  
 Ey = np.zeros( ( 3, N ) )  
  
 *#начальнеые условия* u = np.zeros( N + 1 )  
 ro = 3.0 \* np.ones( N + 1 )  
 V = 1 / ro  
 P = ro \* ( gamma - 1.0 )  
 E = np.ones( N + 1 )  
 r = np.zeros( N + 1 )  
  
 k = 0  
  
 *#@jit* **def** r\_zeros(r0):  
  
 **global** delta\_m, N  
  
 **for** i **in** np.arange( 1, N + 1, 1 ):  
  
 r0[ i ] = ( r0[ i - 1 ] \*\* 3 + delta\_m ) \*\* ( 1 / 3.0 )  
  
 **return** r0  
  
 r = r\_zeros( r )  
  
 *#print( len( list( r ) ) )  
  
#@jit***def** u\_function( P1, P2, r, u ):  
  
 **global** delta\_t, delta\_m  
  
 **return** u - delta\_t / delta\_m \* r \*\* 2 \* ( P1 - P2 )  
  
*#@jit***def** r\_function( r, u ):  
  
 **global** delta\_t  
  
 **return** r + u \* delta\_t  
  
*#@jit***def** P\_function( E, V ):  
  
 **global** gamma  
  
 **return** ( gamma - 1.0 ) \* E / V  
  
*#@jit***def** E\_function( E, V1, V2 ):  
  
 **global** gamma  
  
 **return** E \* V2 / V1 \* ( V1 + ( gamma - 1.0 ) / 2.0 \* ( V1 - V2 ) ) / ( V2 - ( gamma - 1.0 ) / 2.0 \* ( V1 - V2 ) )  
  
@jit  
**def** V\_function( r1, r2 ):  
  
 **global** delta\_m  
  
 **return** ( r1 \*\* 3 - r2 \*\* 3 ) / 3.0 / delta\_m  
  
*#@jit***def** cycle\_for( u, r, V, E, P ):  
  
 **global** u\_function, r\_function, V\_function, P\_function, E\_function  
  
 **global** N  
  
 **for** j **in** np.arange( 0, N - 1, 1 ):  
  
 u[ j + 1 ] = u\_function( P[ j + 1 ], P[ j ], r[ j + 1 ], u[ j + 1 ] )  
  
 r[ j + 1 ] = r\_function( r[ j + 1 ], u[ j + 1 ] )  
  
 V\_old = V[ j ]  
  
 V[ j ] = V\_function( r[ j + 1 ], r[ j ] )  
  
 E[ j ] = E\_function( E[ j ], V\_old, V[ j ] )  
  
 P[ j ] = P\_function( E[ j ], V[ j ] )  
  
 u[N] = u\_function(0, P[N - 1], r[N], u[N])  
  
 r[N] = r\_function(r[N], u[N])  
  
 V\_old = V[N - 1]  
  
 V[N - 1] = V\_function(r[N], r[N - 1])  
  
 E[N - 1] = E\_function(E[N - 1], V\_old, V[N - 1])  
  
 P[N - 1] = P\_function(E[N - 1], V[N - 1])  
  
 *#print( " YES from cycle\_for " )* **return** u, r, V, E, P  
  
*#@jit***def** cycle\_while( u, r, V, E, P, delta\_t ):  
  
 **global** cycle\_for, massive\_for\_print  
 **global** R\_limit, eps, tx  
  
 time = []  
 k = 0  
   
 **while** r[-1] <= R\_limit:  
  
 k = k + 1  
  
 u, r, V, E, P = cycle\_for(u, r, V, E, P)  
  
 **if** np.fabs(R\_limit / 3 \* ( len( time ) + 1 ) - r[-1]) <= eps: *# отбор значений для построение графиков N / step значений по времени* massive\_for\_print(u[:-1], r[:-1], 1.0 / V[:-1], E[:-1])  
  
 time.append( round( k \* delta\_t \* tx, 4 ) )  
  
 *#print( " YSE from cycle\_while " )* **return** time  
  
  
*#@jit***def** massive\_for\_print( U, R, Rho, E ):  
  
 **global** uy, Rhoy, Ey, rx  
 **global** k  
 uy[k] = U  
 Rhoy[k] = Rho  
 Ey[k] = E  
 rx[k] = R  
  
 k = k + 1  
  
*#@jit***def** plot\_graph( x, y, x\_label, y\_label, save, Legend, log ):  
  
 **global** delta\_t, step, MM  
  
 plt.figure( figsize = ( 12, 8 ) )  
  
 plt.xlabel( x\_label )  
 plt.ylabel( y\_label )  
  
 **if** log == **True**:  
  
 plt.yscale( **'log'** )  
 *#plt.xscale( 'log' )* **for** i **in** np.arange( 0, len( x ), 1 ):  
  
 plt.plot( x[i], y[i] )  
  
 plt.legend( Legend )  
 plt.grid()  
 plt.savefig( save+**'.png'**, dpi = 300 )