Week 3 Report

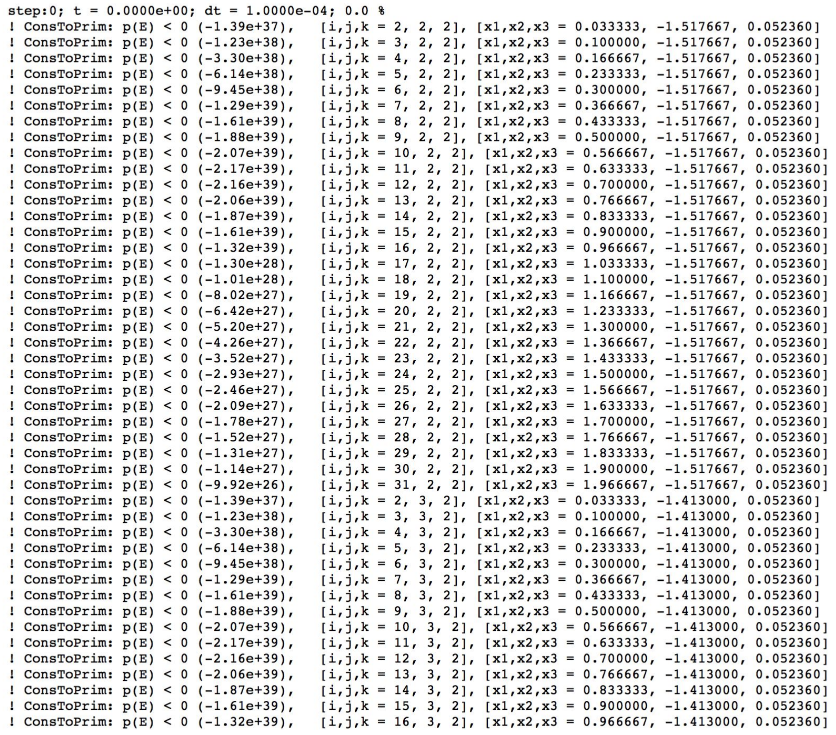
Sam Frederick

Monday / Tuesday

- Reworking code in spherical coordinates
  - Modifications to potential
    - Computational radius ‘r’ defined over the computed domain $\{0 < r <= 2\}$ where R stellar radius is 1.0. In physical terms, unitless computational radius is equivalent to ‘r_{cgs} / R_{cgs}’.
      - By substituting $r_{computational} = r_{cgs} / R_{cgs}$, this allows for proper calculation of potential using a dimensionless computational radius.
  - Hard coding of large-magnitude quantities to support computational efficiency and accuracy.
- Approved for downloading FLASH Software
  - Some difficulties with FLASH not accessing the proper `make` file.
  - https://flash.uchicago.edu/site/index.shtml

Wednesday: Making sense of 'dimensionless units'

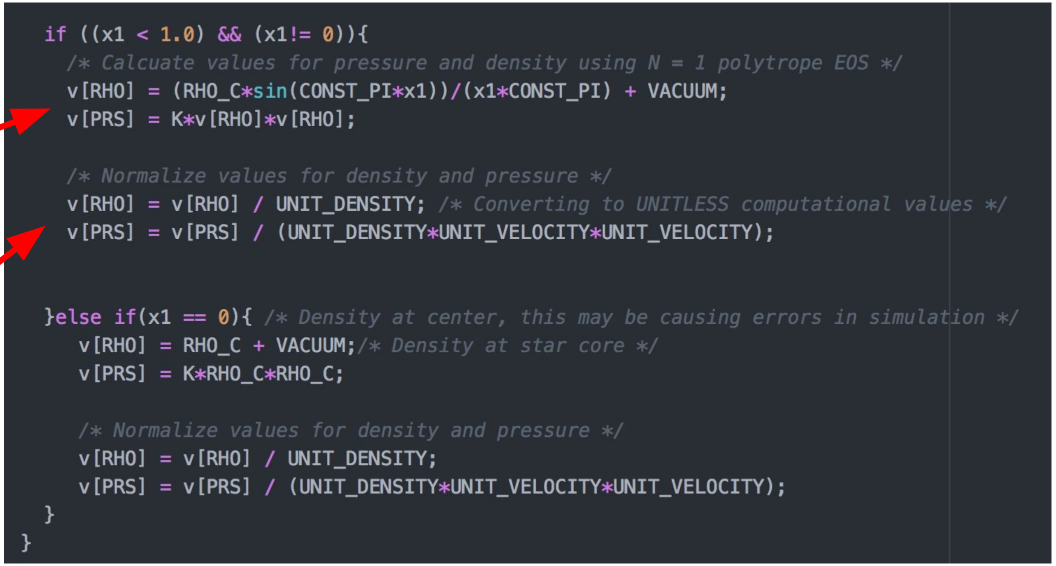
- Tests of computing directly with either CGS or MKS units results in large overflow errors.
  - Displayed in console warning error messages regarding negative densities.
- Unit bases (defined in '`definitions.h`') can solve this issue.



```
step:0; t = 0.0000e+00; dt = 1.0000e-04; 0.0 %
! ConsToPrim: p(E) < 0 (-1.39e+37),   [i,j,k = 2, 2, 2], [x1,x2,x3 = 0.033333, -1.517667, 0.052360]
! ConsToPrim: p(E) < 0 (-1.23e+38),   [i,j,k = 3, 2, 2], [x1,x2,x3 = 0.100000, -1.517667, 0.052360]
! ConsToPrim: p(E) < 0 (-3.30e+38),   [i,j,k = 4, 2, 2], [x1,x2,x3 = 0.166667, -1.517667, 0.052360]
! ConsToPrim: p(E) < 0 (-6.14e+38),   [i,j,k = 5, 2, 2], [x1,x2,x3 = 0.233333, -1.517667, 0.052360]
! ConsToPrim: p(E) < 0 (-9.45e+38),   [i,j,k = 6, 2, 2], [x1,x2,x3 = 0.300000, -1.517667, 0.052360]
! ConsToPrim: p(E) < 0 (-1.29e+39),   [i,j,k = 7, 2, 2], [x1,x2,x3 = 0.366667, -1.517667, 0.052360]
! ConsToPrim: p(E) < 0 (-1.61e+39),   [i,j,k = 8, 2, 2], [x1,x2,x3 = 0.433333, -1.517667, 0.052360]
! ConsToPrim: p(E) < 0 (-1.88e+39),   [i,j,k = 9, 2, 2], [x1,x2,x3 = 0.500000, -1.517667, 0.052360]
! ConsToPrim: p(E) < 0 (-2.07e+39),   [i,j,k = 10, 2, 2], [x1,x2,x3 = 0.566667, -1.517667, 0.052360]
! ConsToPrim: p(E) < 0 (-2.17e+39),   [i,j,k = 11, 2, 2], [x1,x2,x3 = 0.633333, -1.517667, 0.052360]
! ConsToPrim: p(E) < 0 (-2.16e+39),   [i,j,k = 12, 2, 2], [x1,x2,x3 = 0.700000, -1.517667, 0.052360]
! ConsToPrim: p(E) < 0 (-2.06e+39),   [i,j,k = 13, 2, 2], [x1,x2,x3 = 0.766667, -1.517667, 0.052360]
! ConsToPrim: p(E) < 0 (-1.87e+39),   [i,j,k = 14, 2, 2], [x1,x2,x3 = 0.833333, -1.517667, 0.052360]
! ConsToPrim: p(E) < 0 (-1.61e+39),   [i,j,k = 15, 2, 2], [x1,x2,x3 = 0.900000, -1.517667, 0.052360]
! ConsToPrim: p(E) < 0 (-1.32e+39),   [i,j,k = 16, 2, 2], [x1,x2,x3 = 0.966667, -1.517667, 0.052360]
! ConsToPrim: p(E) < 0 (-1.30e+28),   [i,j,k = 17, 2, 2], [x1,x2,x3 = 1.033333, -1.517667, 0.052360]
! ConsToPrim: p(E) < 0 (-1.01e+28),   [i,j,k = 18, 2, 2], [x1,x2,x3 = 1.100000, -1.517667, 0.052360]
! ConsToPrim: p(E) < 0 (-8.02e+27),   [i,j,k = 19, 2, 2], [x1,x2,x3 = 1.166667, -1.517667, 0.052360]
! ConsToPrim: p(E) < 0 (-6.42e+27),   [i,j,k = 20, 2, 2], [x1,x2,x3 = 1.233333, -1.517667, 0.052360]
! ConsToPrim: p(E) < 0 (-5.20e+27),   [i,j,k = 21, 2, 2], [x1,x2,x3 = 1.300000, -1.517667, 0.052360]
! ConsToPrim: p(E) < 0 (-4.26e+27),   [i,j,k = 22, 2, 2], [x1,x2,x3 = 1.366667, -1.517667, 0.052360]
! ConsToPrim: p(E) < 0 (-3.52e+27),   [i,j,k = 23, 2, 2], [x1,x2,x3 = 1.433333, -1.517667, 0.052360]
! ConsToPrim: p(E) < 0 (-2.93e+27),   [i,j,k = 24, 2, 2], [x1,x2,x3 = 1.500000, -1.517667, 0.052360]
! ConsToPrim: p(E) < 0 (-2.46e+27),   [i,j,k = 25, 2, 2], [x1,x2,x3 = 1.566667, -1.517667, 0.052360]
! ConsToPrim: p(E) < 0 (-2.09e+27),   [i,j,k = 26, 2, 2], [x1,x2,x3 = 1.633333, -1.517667, 0.052360]
! ConsToPrim: p(E) < 0 (-1.78e+27),   [i,j,k = 27, 2, 2], [x1,x2,x3 = 1.700000, -1.517667, 0.052360]
! ConsToPrim: p(E) < 0 (-1.52e+27),   [i,j,k = 28, 2, 2], [x1,x2,x3 = 1.766667, -1.517667, 0.052360]
! ConsToPrim: p(E) < 0 (-1.31e+27),   [i,j,k = 29, 2, 2], [x1,x2,x3 = 1.833333, -1.517667, 0.052360]
! ConsToPrim: p(E) < 0 (-1.14e+27),   [i,j,k = 30, 2, 2], [x1,x2,x3 = 1.900000, -1.517667, 0.052360]
! ConsToPrim: p(E) < 0 (-9.92e+26),   [i,j,k = 31, 2, 2], [x1,x2,x3 = 1.966667, -1.517667, 0.052360]
! ConsToPrim: p(E) < 0 (-1.39e+37),   [i,j,k = 2, 3, 2], [x1,x2,x3 = 0.033333, -1.413000, 0.052360]
! ConsToPrim: p(E) < 0 (-1.23e+38),   [i,j,k = 3, 3, 2], [x1,x2,x3 = 0.100000, -1.413000, 0.052360]
! ConsToPrim: p(E) < 0 (-3.30e+38),   [i,j,k = 4, 3, 2], [x1,x2,x3 = 0.166667, -1.413000, 0.052360]
! ConsToPrim: p(E) < 0 (-6.14e+38),   [i,j,k = 5, 3, 2], [x1,x2,x3 = 0.233333, -1.413000, 0.052360]
! ConsToPrim: p(E) < 0 (-9.45e+38),   [i,j,k = 6, 3, 2], [x1,x2,x3 = 0.300000, -1.413000, 0.052360]
! ConsToPrim: p(E) < 0 (-1.29e+39),   [i,j,k = 7, 3, 2], [x1,x2,x3 = 0.366667, -1.413000, 0.052360]
! ConsToPrim: p(E) < 0 (-1.61e+39),   [i,j,k = 8, 3, 2], [x1,x2,x3 = 0.433333, -1.413000, 0.052360]
! ConsToPrim: p(E) < 0 (-1.88e+39),   [i,j,k = 9, 3, 2], [x1,x2,x3 = 0.500000, -1.413000, 0.052360]
! ConsToPrim: p(E) < 0 (-2.07e+39),   [i,j,k = 10, 3, 2], [x1,x2,x3 = 0.566667, -1.413000, 0.052360]
! ConsToPrim: p(E) < 0 (-2.17e+39),   [i,j,k = 11, 3, 2], [x1,x2,x3 = 0.633333, -1.413000, 0.052360]
! ConsToPrim: p(E) < 0 (-2.16e+39),   [i,j,k = 12, 3, 2], [x1,x2,x3 = 0.700000, -1.413000, 0.052360]
! ConsToPrim: p(E) < 0 (-2.06e+39),   [i,j,k = 13, 3, 2], [x1,x2,x3 = 0.766667, -1.413000, 0.052360]
! ConsToPrim: p(E) < 0 (-1.87e+39),   [i,j,k = 14, 3, 2], [x1,x2,x3 = 0.833333, -1.413000, 0.052360]
! ConsToPrim: p(E) < 0 (-1.61e+39),   [i,j,k = 15, 3, 2], [x1,x2,x3 = 0.900000, -1.413000, 0.052360]
! ConsToPrim: p(E) < 0 (-1.32e+39),   [i,j,k = 16, 3, 2], [x1,x2,x3 = 0.966667, -1.413000, 0.052360]
```

Method of Parameter Calculation

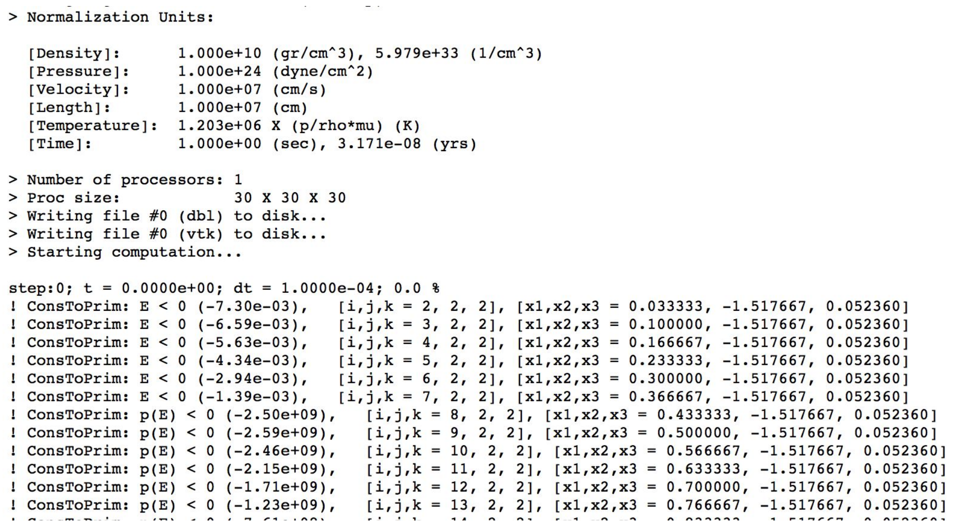
- First, the code computes values for variables in cgs units. Dependent variables such as pressure are also calculated at this stage.
- Then, the code divides these values by unit bases to convert them into unitless values with significantly smaller numerical magnitude.



```
  if ((x1 < 1.0) && (x1!= 0)){
    /* Calcuate values for pressure and density using N = 1 polytrope EOS */
    v[RHO] = (RHO_C*sin(CONST_PI*x1))/(x1*CONST_PI) + VACUUM;
    v[PRS] = K*v[RHO]*v[RHO];

    /* Normalize values for density and pressure */
    v[RHO] = v[RHO] / UNIT_DENSITY; /* Converting to UNITLESS computational values */
    v[PRS] = v[PRS] / (UNIT_DENSITY*UNIT_VELOCITY*UNIT_VELOCITY);


  }else if(x1 == 0){ /* Density at center, this may be causing errors in simulation */
     v[RHO] = RHO_C + VACUUM;/* Density at star core */
     v[PRS] = K*RHO_C*RHO_C;

     /* Normalize values for density and pressure */
     v[RHO] = v[RHO] / UNIT_DENSITY;
     v[PRS] = v[PRS] / (UNIT_DENSITY*UNIT_VELOCITY*UNIT_VELOCITY);
  }
}
```

Wednesday: Making sense of ‘dimensionless units’

- By successively increasing the size of our normalization units, we can effectively minimize underflow/overflow errors.

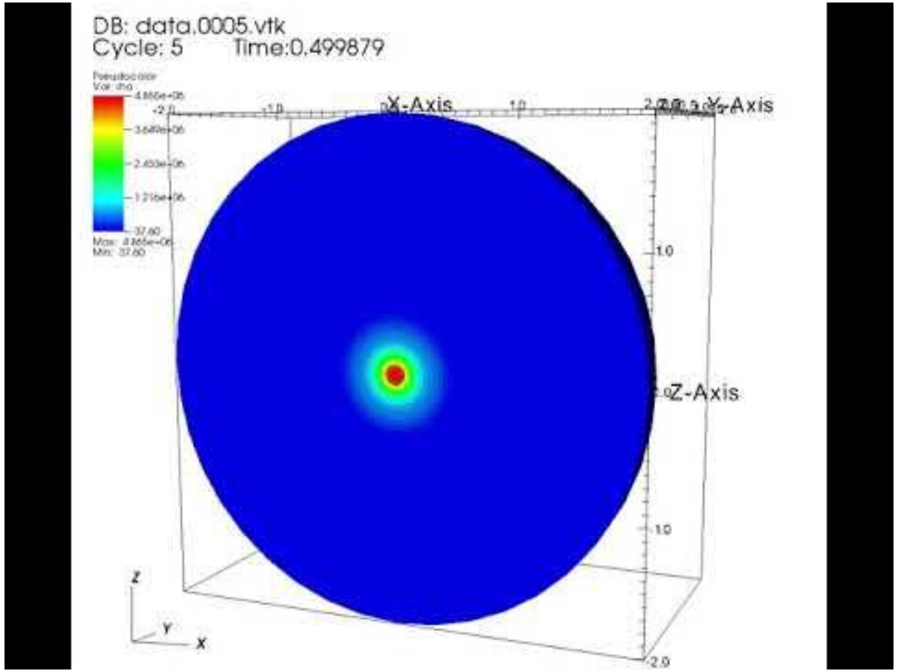


```
> Normalization Units:

  [Density]:      1.000e+10 (gr/cm^3), 5.979e+33 (1/cm^3)
  [Pressure]:     1.000e+24 (dyne/cm^2)
  [Velocity]:     1.000e+07 (cm/s)
  [Length]:       1.000e+07 (cm)
  [Temperature]:  1.203e+06 X (p/rho*mu) (K)
  [Time]:         1.000e+00 (sec), 3.171e-08 (yrs)

> Number of processors: 1
> Proc size:            30 X 30 X 30
> Writing file #0 (dbl) to disk...
> Writing file #0 (vtk) to disk...
> Starting computation...

step:0; t = 0.0000e+00; dt = 1.0000e-04; 0.0 %
! ConsToPrim: E < 0 (-7.30e-03),   [i,j,k = 2, 2, 2], [x1,x2,x3 = 0.033333, -1.517667, 0.052360]
! ConsToPrim: E < 0 (-6.59e-03),   [i,j,k = 3, 2, 2], [x1,x2,x3 = 0.100000, -1.517667, 0.052360]
! ConsToPrim: E < 0 (-5.63e-03),   [i,j,k = 4, 2, 2], [x1,x2,x3 = 0.166667, -1.517667, 0.052360]
! ConsToPrim: E < 0 (-4.34e-03),   [i,j,k = 5, 2, 2], [x1,x2,x3 = 0.233333, -1.517667, 0.052360]
! ConsToPrim: E < 0 (-2.94e-03),   [i,j,k = 6, 2, 2], [x1,x2,x3 = 0.300000, -1.517667, 0.052360]
! ConsToPrim: E < 0 (-1.39e-03),   [i,j,k = 7, 2, 2], [x1,x2,x3 = 0.366667, -1.517667, 0.052360]
! ConsToPrim: p(E) < 0 (-2.50e+09),   [i,j,k = 8, 2, 2], [x1,x2,x3 = 0.433333, -1.517667, 0.052360]
! ConsToPrim: p(E) < 0 (-2.59e+09),   [i,j,k = 9, 2, 2], [x1,x2,x3 = 0.500000, -1.517667, 0.052360]
! ConsToPrim: p(E) < 0 (-2.46e+09),   [i,j,k = 10, 2, 2], [x1,x2,x3 = 0.566667, -1.517667, 0.052360]
! ConsToPrim: p(E) < 0 (-2.15e+09),   [i,j,k = 11, 2, 2], [x1,x2,x3 = 0.633333, -1.517667, 0.052360]
! ConsToPrim: p(E) < 0 (-1.71e+09),   [i,j,k = 12, 2, 2], [x1,x2,x3 = 0.700000, -1.517667, 0.052360]
! ConsToPrim: p(E) < 0 (-1.23e+09),   [i,j,k = 13, 2, 2], [x1,x2,x3 = 0.766667, -1.517667, 0.052360]
```

Thursday: Debugging

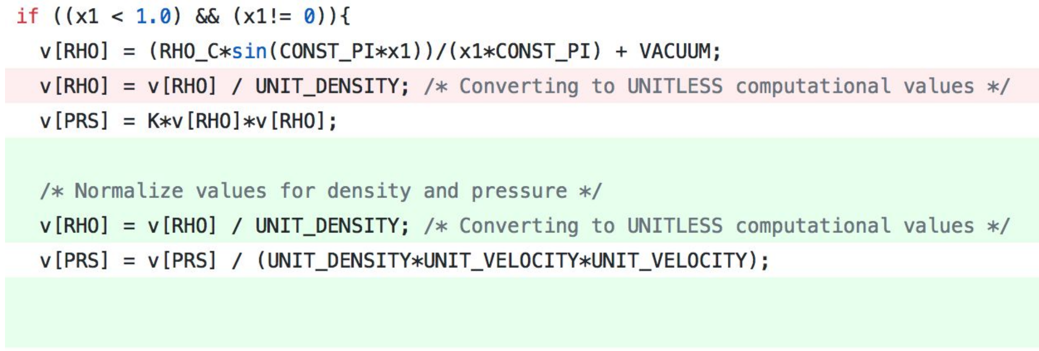
- Pressure was considerably smaller than expected.
  - Resulted in a collapsing polytrope
- In order to achieve stable model, radiation pressure must balance force due to gravitational acceleration
  - Hydrostatic Equilibrium

$$\frac{dP}{dr} = -\frac{GM(r)\rho(r)}{r^2}$$

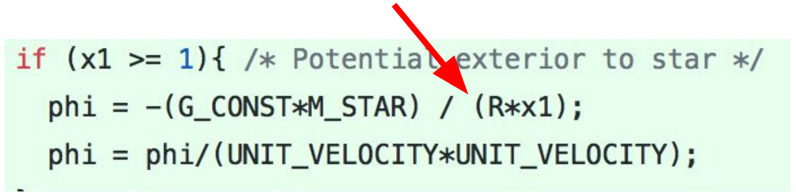


Jumping the Gun: Unit values before Unit Normalization

- Density was being normalized, followed by computing pressure which itself was normalized.
  - This dramatically undervalued pressure magnitude.
- Code was visually organized to clearly differentiate computation in cgs units and variable normalization.
- Minor corrections also made to potential



```
if ((x1 < 1.0) && (x1!= 0)){
  v[RHO] = (RHO_C*sin(CONST_PI*x1))/(x1*CONST_PI) + VACUUM;
  v[RHO] = v[RHO] / UNIT_DENSITY; /* Converting to UNITLESS computational values */
  v[PRS] = K*v[RHO]*v[RHO];

  /* Normalize values for density and pressure */
  v[RHO] = v[RHO] / UNIT_DENSITY; /* Converting to UNITLESS computational values */
  v[PRS] = v[PRS] / (UNIT_DENSITY*UNIT_VELOCITY*UNIT_VELOCITY);
```



```
if (x1 >= 1){ /* Potential exterior to star */
  phi = -(G_CONST*M_STAR) / (R*x1);
  phi = phi/(UNIT_VELOCITY*UNIT_VELOCITY);
```

Specializing the Adiabatic Index (γ)

- Adiabatic index is a measure of the ratio of specific heats at constant pressure, constant volume.
    - For the `IDEAL` thermal EOS, PLUTO sets the default value to be 5/3.
    - Stored as global variable `g_gamma`, can be edited in `init.c`.
- For a polytrope, pressure follows the equation [*]

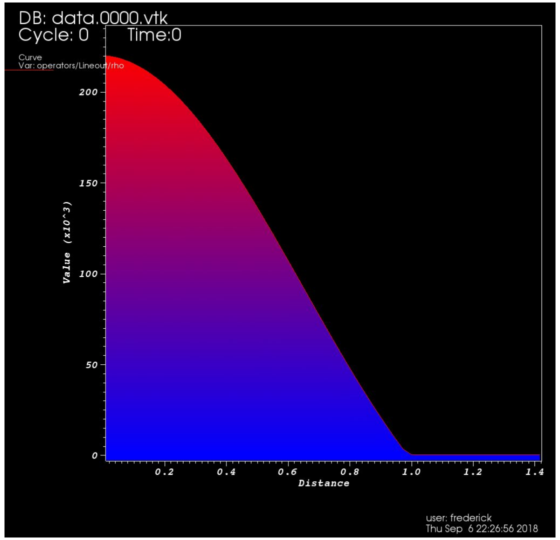
$$P = K\rho^{\gamma} = K\rho^{(n+1)/n}$$

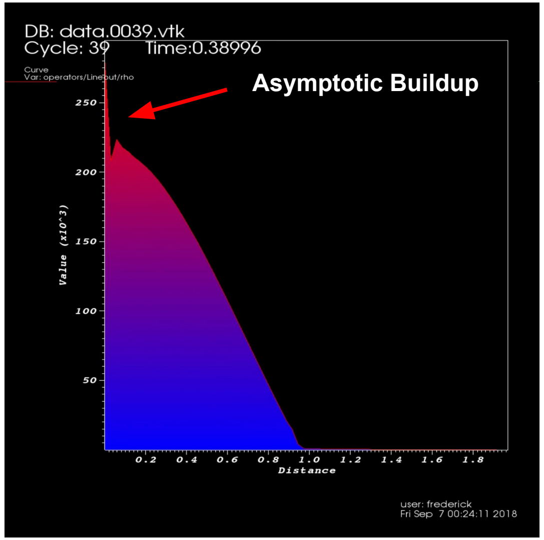
- Using an N = 1 polytrope, this means γ = (1+1) / 1 = **2**
    - Adiabatic index was redefined to be '2' in `init.c`.
- With this addition, we step much closer to hydrostatic equilibrium and model stability.

[*] Cohen, Judith G. "Polytropes – Derivation and Solutions of the Lane-Emden Equation." *Class Notes: Polytropes*, 20 Nov. 2015, www.astro.caltech.edu/~jlc/ay101_fall2015/ay101_polytropes_fall2015.pdf.

Friday: (Nearly) Stable

- Running simulations with the updated adiabatic index and prior code corrections results in an almost uniformly stable model in time.
    - Divergent behavior near r = 0 region, causing asymptotic buildup of density and pressure at origin. Despite this, the vast majority of the model retains state over time.

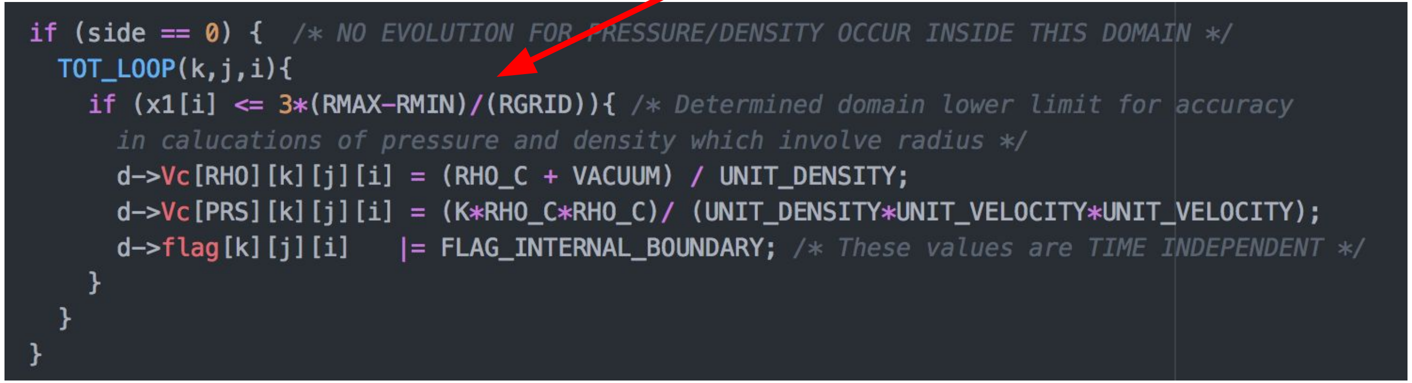




Saturday: Internal Boundary Condition (PLUTO User Manual pg. 54)

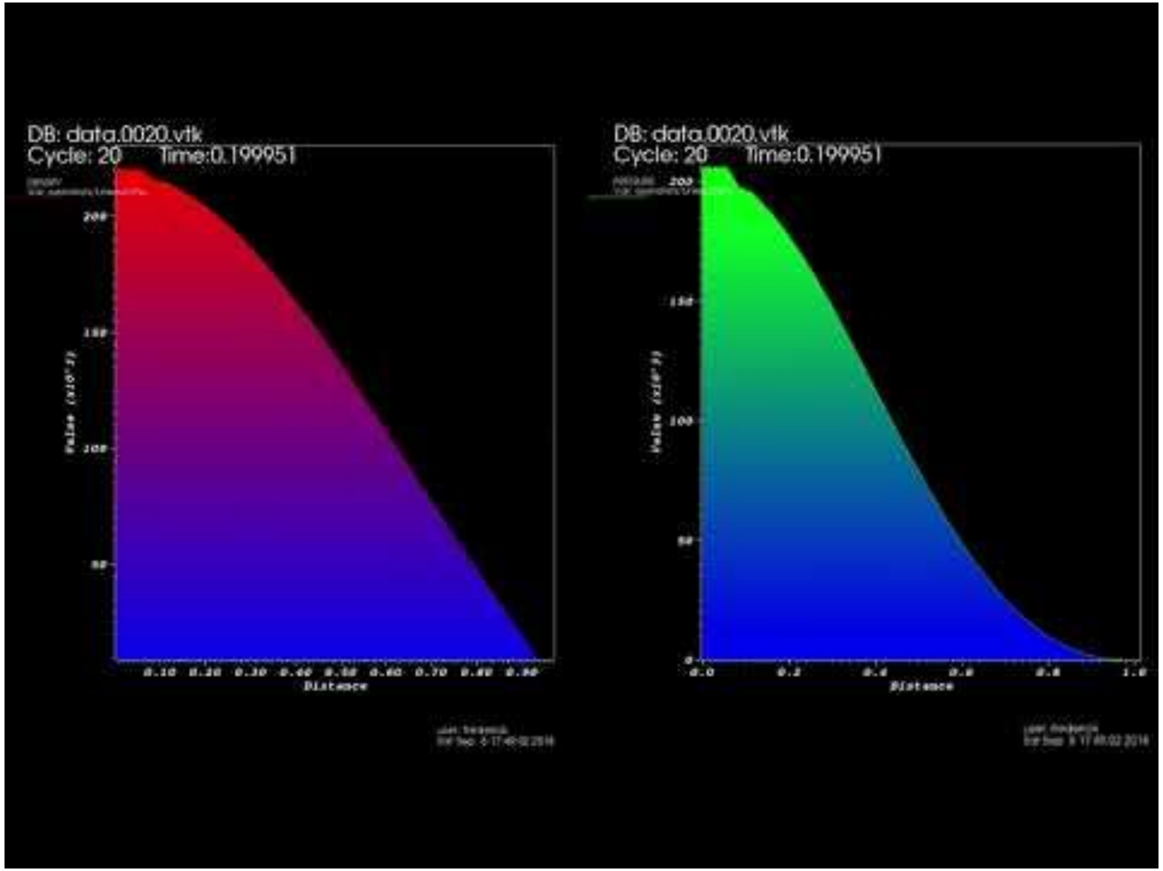
- The `UserDefBoundary()` within the `init.c` file allows us to specify values for computed variables within a chosen region. We may furthermore set the `FLAG_INTERNAL_BOUNDARY` flag to signal to the program that we don't wish to evolve this specified region over the timespan of the simulation. A region about r = 0 was chosen for this condition, setting static values for pressure and density.

First three radial grid points remain static



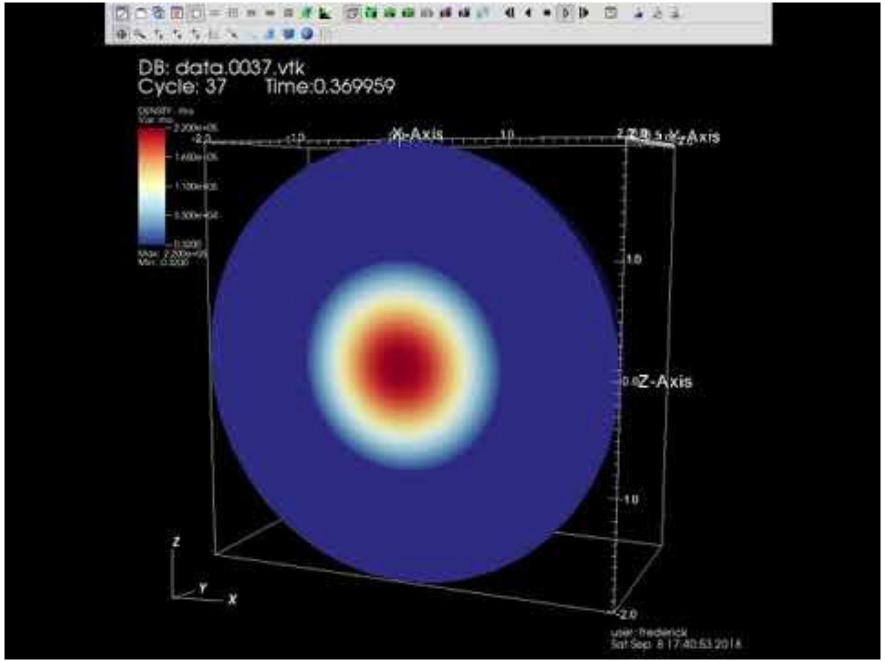
```
if (side == 0) {  /* NO EVOLUTION FOR PRESSURE/DENSITY OCCUR INSIDE THIS DOMAIN */
  TOT_LOOP(k,j,i){
    if (x1[i] <= 3*(RMAX-RMIN)/(RGRID)){ /* Determined domain lower limit for accuracy
      in calucations of pressure and density which involve radius */
      d->Vc[RHO][k][j][i] = (RHO_C + VACUUM) / UNIT_DENSITY;
      d->Vc[PRS][k][j][i] = (K*RHO_C*RHO_C)/ (UNIT_DENSITY*UNIT_VELOCITY*UNIT_VELOCITY);
      d->flag[k][j][i]    |= FLAG_INTERNAL_BOUNDARY; /* These values are TIME INDEPENDENT */
    }
  }
}
```

Animation of Density, pressure over time

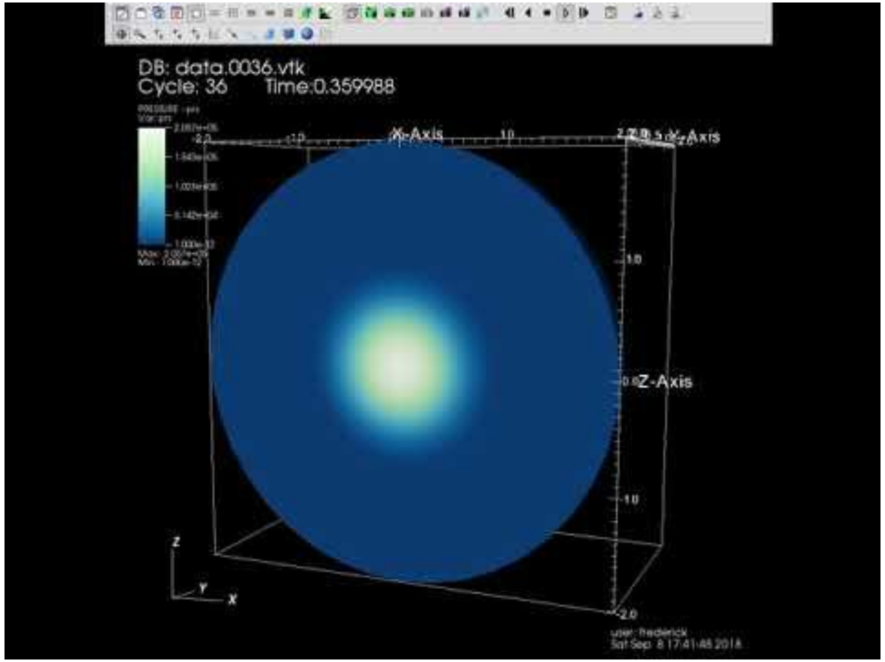


Animations of Model Stability

Density



Pressure



Goals for the Week

- Refine internal boundary condition, determine whether simulation develops asymptotic behavior with only r = 0 gridpoint specified in condition. If so, specify grid points within condition to be various weighted averages of core density/pressure and density/pressure immediately outside IBC region, possibly allow for time evolution.
- Begin work on adding poloidal and toroidal magnetic field components
  - Given Kuhn’s work, can foresee issues with balancing boundary conditions