Numerical Preheating

Thoughts and Exercises: Afternoon

Tom Giblin
RIKEN Interdisciplinary Theoretical and Mathematical Sciences Program
Saitama, Japan
March 5, 2025

There are many (many) codes

…many of which are open-source or available

- LatticeEasy, Gary Elder 2000 (https://www.felderbooks.com/latticeeasy/index.html)
    - CLUSTEREasy, arXiv:0712.0813 (https://www.felderbooks.com/latticeeasy/index.html)
- DEFrost, Andrei Frolov, 2008, arXiv:0809.4904, (https://www.sfu.ca/physics/cosmology/defrost/)
- CUDAEasy, Jani Sanio, 2009, arXiv:0911.5692
- PSpectRe, Richard Easther, Hal Finkle, Nathaniel Roth, arXiv:1005.1921
- HLATTICE, Zhiqi Huang, arXiv: 1102.0227 (https://www.cita.utoronto.ca/~zqhuang/hlat/)
- GABE, JTG, Hillary Child, J. Tate Deskins, arXiv:1305.0561, (https://cosmo.kenyon.edu/gabe.html)
- PyCOOL, Jani Sainio, arXiv:1201.5029
- CosmoLattice, 2020, Daniel G. Figueroa, Adrien Florio, Francisco Torrenti, Wessel Valkenburg, arXiv:2006.15122, (https://cosmolattice.net/)

Why GABE?

…why now?

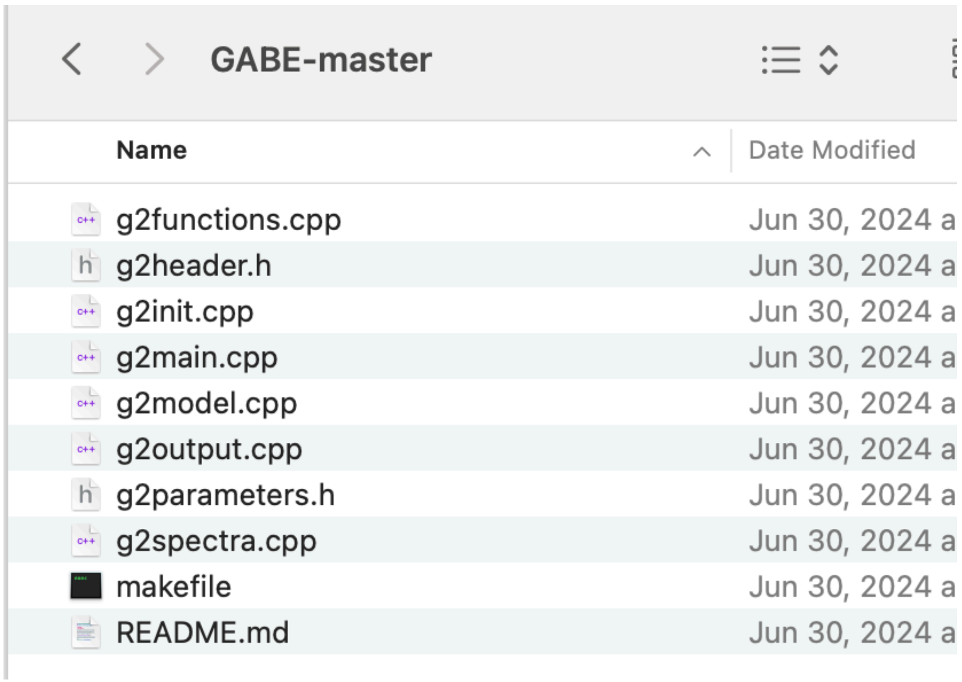
- GABE does much of what (most of) these codes can do.
  - The user has control over the model (potential) and the ability to change most of the numerical parameters
  - The code has been shown to be applicable to many scenarios
- GABE uses an RK-2 (or for BSSN RK-4) integration scheme
  - Possibly, uniquely, GABE was written to be flexible with non-canonical fields and kinetic structures
- GABE has extensions for Gravitational Waves, linearized gravity and Numerical Relativity (not widely distributed)

Using GABE

…for the practicals

- If you’re running a MAC, hopefully you’ve had a chance to follow the “upgrade instructions” to get an open-mp compatible compiler and FFTW.
- If you’re running LINUX, and you have FFTW-3 installed, you should be able to run GABE by modifying the makefile
- Otherwise, you can use a remote version
    - $ssh ithems@ann.kenyon.edu
    - Password: Il0veGABE! (Upper-case I then lower-case l then number 0)

The Software Files



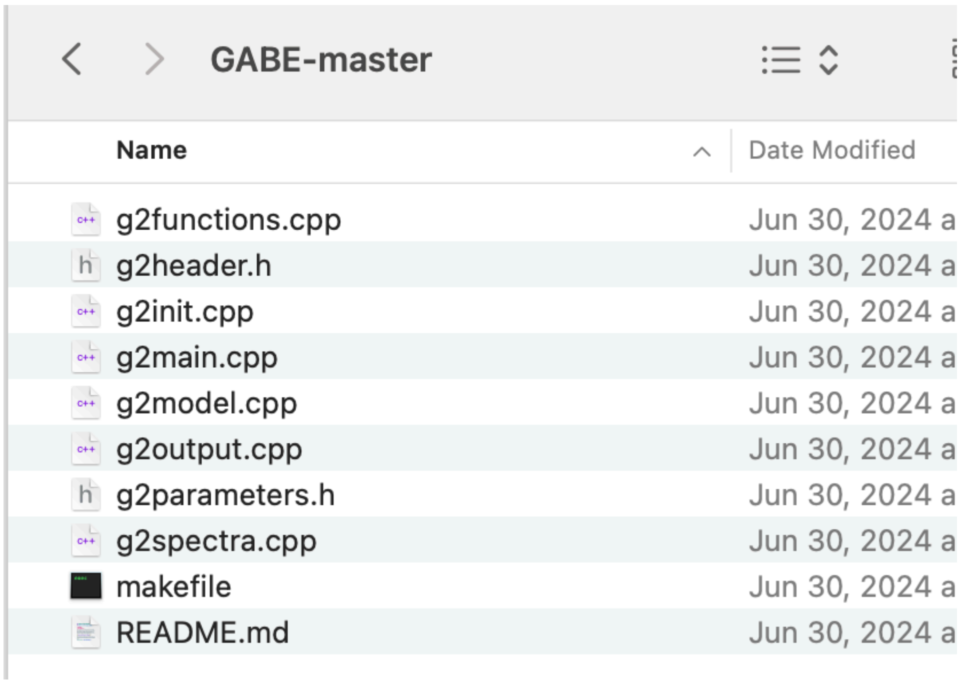
GABE-master

Name	Date Modified
g2functions.cpp	Jun 30, 2024 a
g2header.h	Jun 30, 2024 a
g2init.cpp	Jun 30, 2024 a
g2main.cpp	Jun 30, 2024 a
g2model.cpp	Jun 30, 2024 a
g2output.cpp	Jun 30, 2024 a
g2parameters.h	Jun 30, 2024 a
g2spectra.cpp	Jun 30, 2024 a
makefile	Jun 30, 2024 a
README.md	Jun 30, 2024 a

Files we're not going to look at today

- g2functions.cpp: functions required for field and spacetime evolution
- g2header.h: common header file
- g2init.cpp: initialization routines
- g2main.cpp: the main function
- g2outoput.cpp and g2spectra.cpp: output routines

The Software Files


GABE-master

Name	Date Modified
g2functions.cpp	Jun 30, 2024 a
g2header.h	Jun 30, 2024 a
g2init.cpp	Jun 30, 2024 a
g2main.cpp	Jun 30, 2024 a
g2model.cpp	Jun 30, 2024 a
g2output.cpp	Jun 30, 2024 a
g2parameters.h	Jun 30, 2024 a
g2spectra.cpp	Jun 30, 2024 a
makefile	Jun 30, 2024 a
README.md	Jun 30, 2024 a

On top of this there are two files that we will modify:

- g2model.cpp: the file which contains the specific potential of interest as well as other model-specific functions
- g2parameters.h: most frequently changed, contains the physical and numerical parameters of the model

The model file

- GABE comes pre-loaded with the ‘vanilla preheating’ model:

$$\mathcal{L} = -\frac{1}{2}\partial_\mu\phi\partial^\mu\phi - \frac{1}{2}\partial_\mu\chi\partial^\mu\chi - \frac{1}{2}m^2\phi^2 - \frac{g^2}{2}\phi^2\chi^2$$

- Where the only part that we really need is

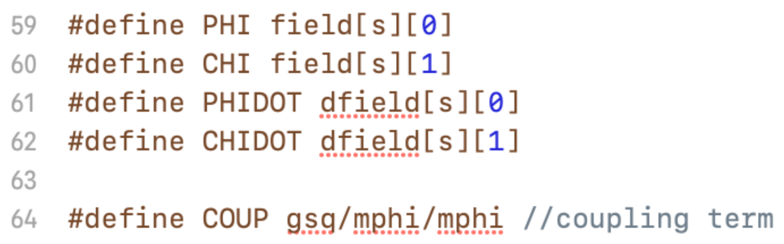
$$V(\phi,\chi) = \frac{1}{2}m^2\phi^2 + \frac{g^2}{2}\phi^2\chi^2$$

Notes on dimensionless variables

...this is often the most difficult part for new

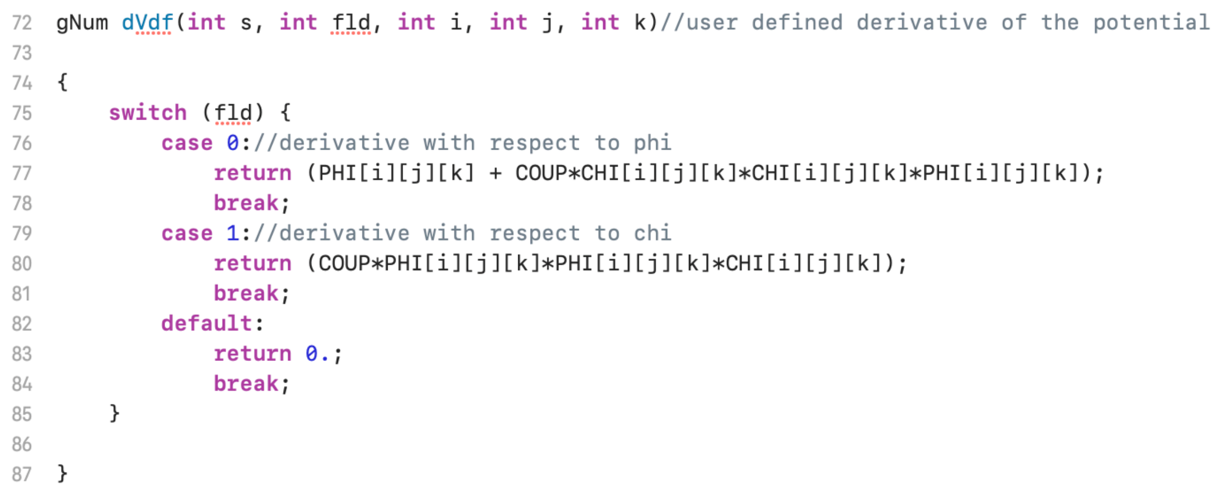
$$V_{\rm pr} = \frac{V(\phi,\chi)}{B^2 m_{\rm pl}^2} = \frac{1}{2}\frac{m^2 m_{\rm pl}^2 \phi_{\rm pr}^2}{B^2 m_{\rm pl}^2} + \frac{g^2}{2}\frac{m_{\rm pl}^4 \phi_{\rm pl}^2 \chi_{\rm pl}^2}{B^2 m_{\rm pl}^2}$$

$$V_{\rm pr} = \frac{1}{2}\phi_{\rm pr}^2 + \frac{1}{2}\frac{g^2 m_{\rm pl}^2}{m^2}\phi_{\rm pr}^2\chi_{\rm pr}^2$$


```
#define PHI field[s][0]
#define CHI field[s][1]
#define PHIDOT dfield[s][0]
#define CHIDOT dfield[s][1]

#define COUP gsq/mphi/mphi //coupling term
```

g2model.cpp

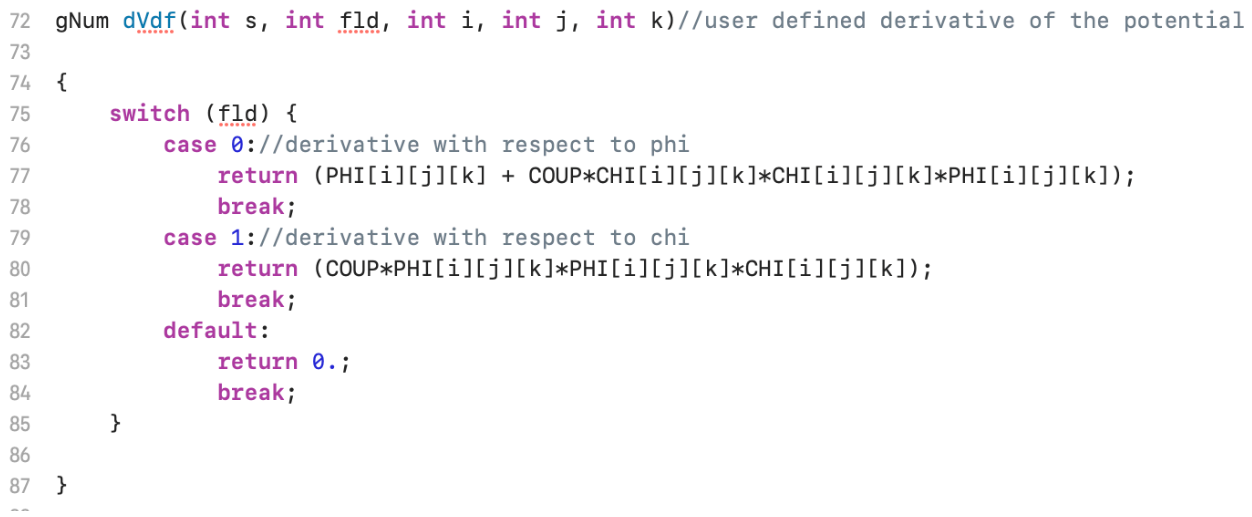
- The model appears in three important functions. These are what need to be changed when the potential is changed.



```
gNum dVdf(int s, int fld, int i, int j, int k)//user defined derivative of the potential

{
    switch (fld) {
        case 0://derivative with respect to phi
            return (PHI[i][j][k] + COUP*CHI[i][j][k]*CHI[i][j][k]*PHI[i][j][k]);
            break;
        case 1://derivative with respect to chi
            return (COUP*PHI[i][j][k]*PHI[i][j][k]*CHI[i][j][k]);
            break;
        default:
            return 0.;
            break;
    }

}
```

g2model.cpp

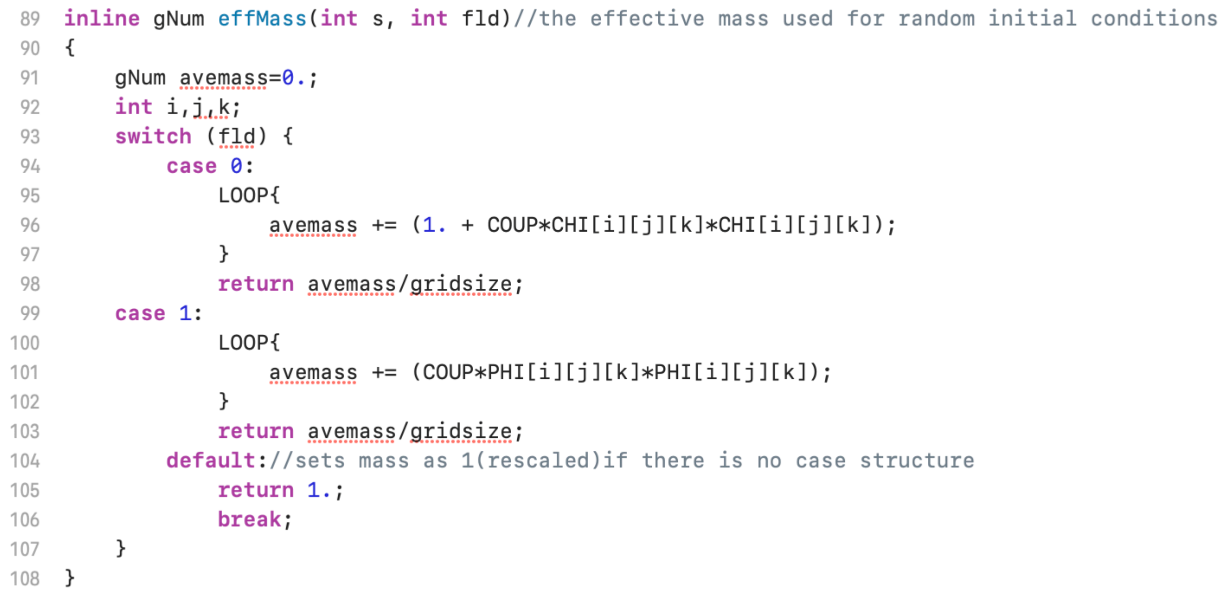
- The model appears in three important functions. These are what need to be changed when the potential is changed.



```
gNum dVdf(int s, int fld, int i, int j, int k)//user defined derivative of the potential

{
    switch (fld) {
        case 0://derivative with respect to phi
            return (PHI[i][j][k] + COUP*CHI[i][j][k]*CHI[i][j][k]*PHI[i][j][k]);
            break;
        case 1://derivative with respect to chi
            return (COUP*PHI[i][j][k]*PHI[i][j][k]*CHI[i][j][k]);
            break;
        default:
            return 0.;
            break;
    }

}
```

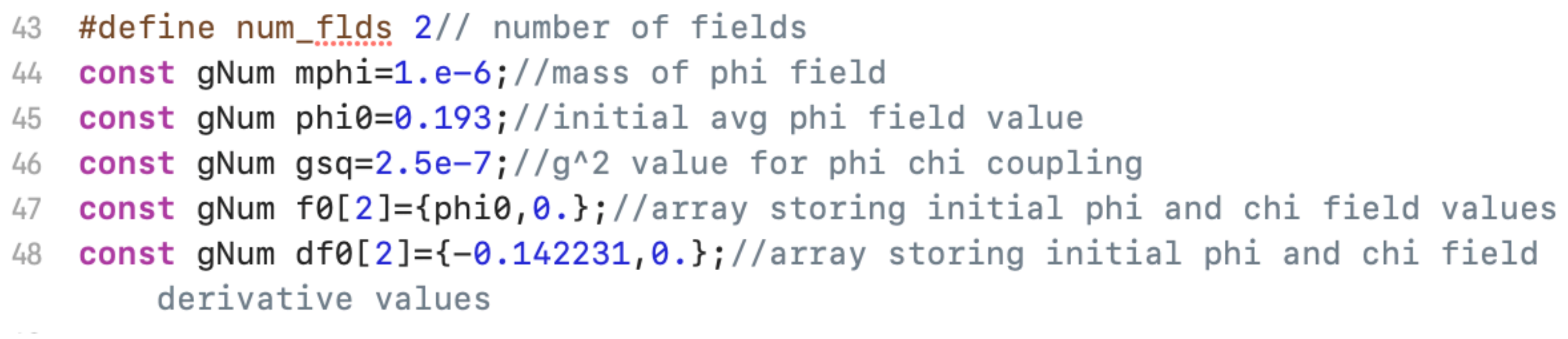
g2model.cpp

- The model appears in three important functions. These are what need to be changed when the potential is changed.



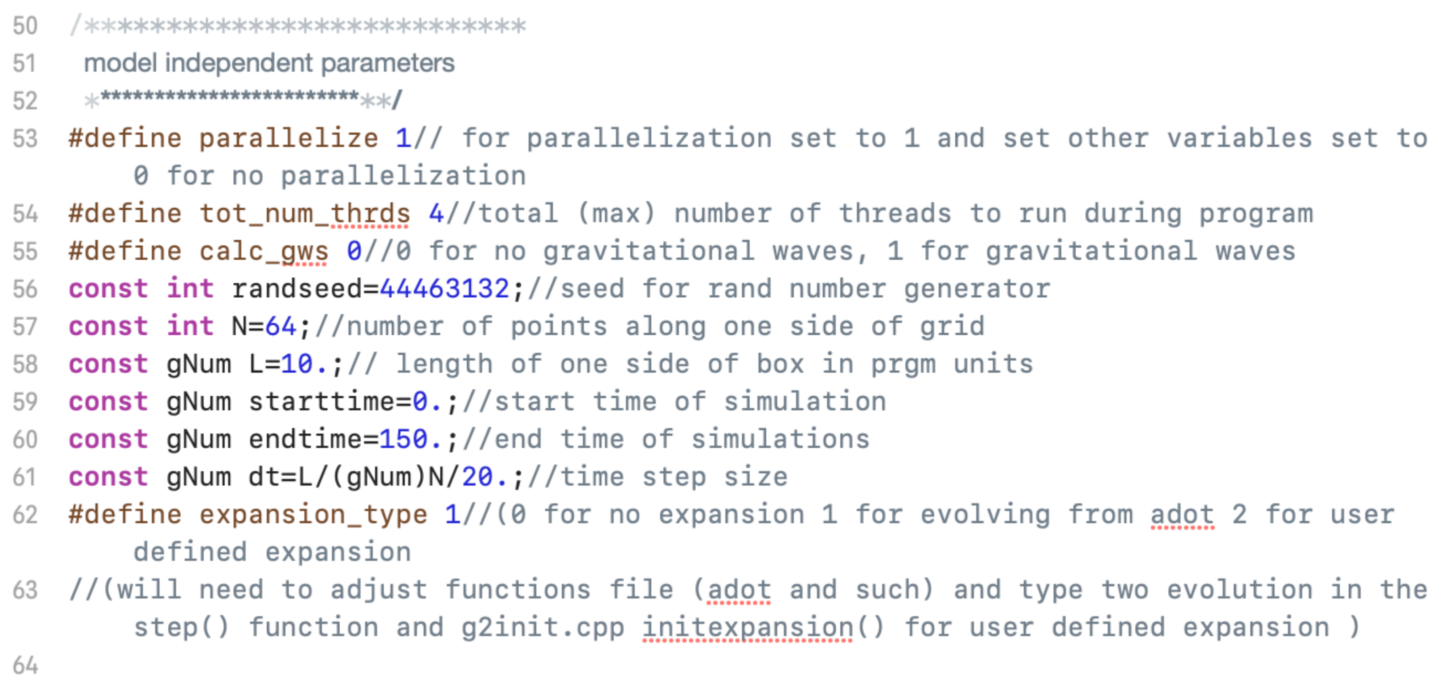
```cpp
inline gNum effMass(int s, int fld)//the effective mass used for random initial conditions
{
    gNum avemass=0.;
    int i,j,k;
    switch (fld) {
        case 0:
            LOOP{
                avemass += (1. + COUP*CHI[i][j][k]*CHI[i][j][k]);
            }
            return avemass/gridsize;
    case 1:
            LOOP{
                avemass += (COUP*PHI[i][j][k]*PHI[i][j][k]);
            }
            return avemass/gridsize;
        default://sets mass as 1(rescaled)if there is no case structure
            return 1.;
            break;
    }
}
```

g2parameters.cpp



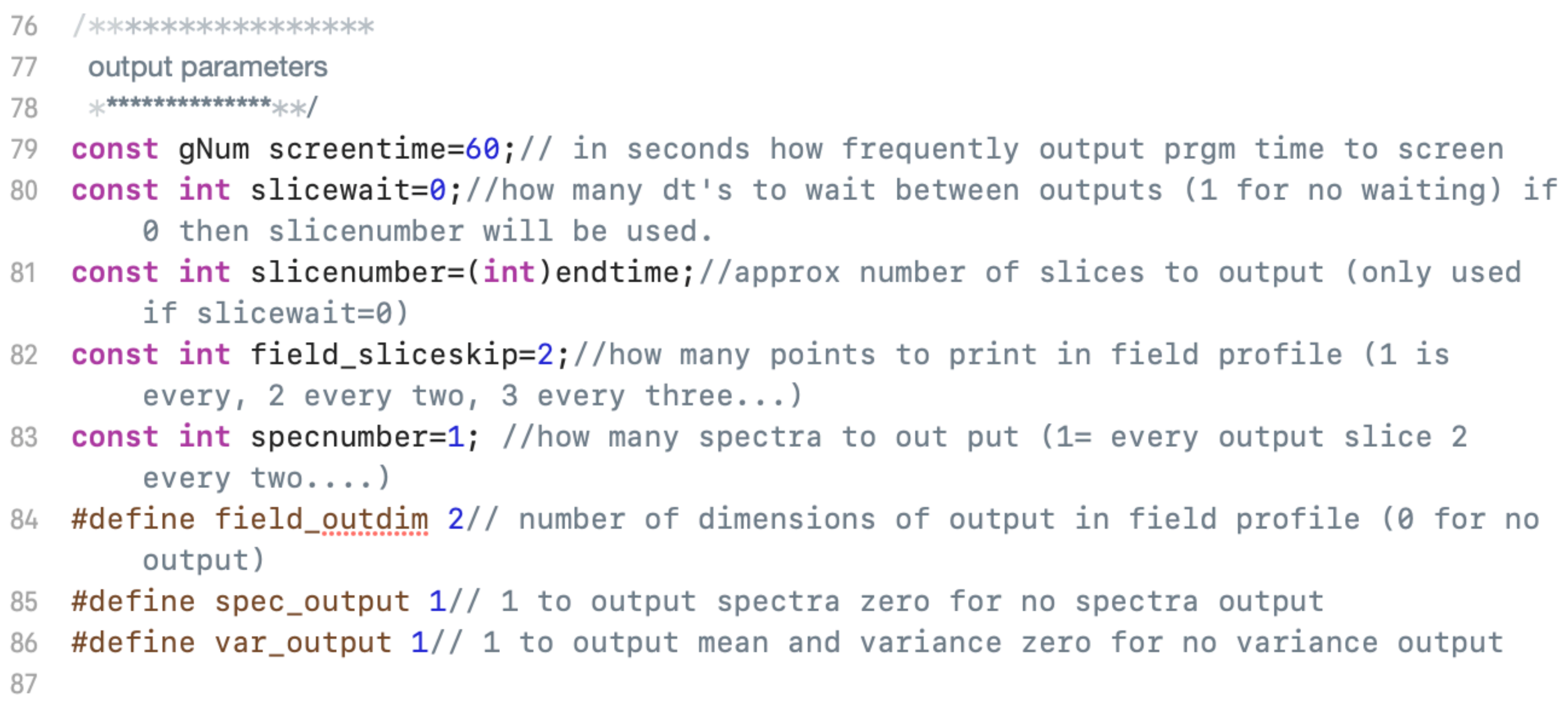
```cpp
#define num_flds 2// number of fields
const gNum mphi=1.e-6;//mass of phi field
const gNum phi0=0.193;//initial avg phi field value
const gNum gsq=2.5e-7;//g^2 value for phi chi coupling
const gNum f0[2]={phi0,0.};//array storing initial phi and chi field values
const gNum df0[2]={-0.142231,0.};//array storing initial phi and chi field derivative values
```

g2parameters.cpp



```
/***************************
 model independent parameters
 ***************************/
#define parallelize 1// for parallelization set to 1 and set other variables set to 0 for no parallelization
#define tot_num_thrds 4//total (max) number of threads to run during program
#define calc_gws 0//0 for no gravitational waves, 1 for gravitational waves
const int randseed=44463132;//seed for rand number generator
const int N=64;//number of points along one side of grid
const gNum L=10.;// length of one side of box in prgm units
const gNum starttime=0.;//start time of simulation
const gNum endtime=150.;//end time of simulations
const gNum dt=L/(gNum)N/20.;//time step size
#define expansion_type 1//(0 for no expansion 1 for evolving from adot 2 for user defined expansion
//(will need to adjust functions file (adot and such) and type two evolution in the step() function and g2init.cpp initexpansion() for user defined expansion )

```

g2parameters.cpp



```cpp
/*****************
 output parameters
 *****************/
const gNum screentime=60;// in seconds how frequently output prgm time to screen
const int slicewait=0;//how many dt's to wait between outputs (1 for no waiting) if 0 then slicenumber will be used.
const int slicenumber=(int)endtime;//approx number of slices to output (only used if slicewait=0)
const int field_sliceskip=2;//how many points to print in field profile (1 is every, 2 every two, 3 every three...)
const int specnumber=1; //how many spectra to out put (1= every output slice 2 every two....)
#define field_outdim 2// number of dimensions of output in field profile (0 for no output)
#define spec_output 1// 1 to output spectra zero for no spectra output
#define var_output 1// 1 to output mean and variance zero for no variance output

```

On to the challenge!

Challenge 1

Part A

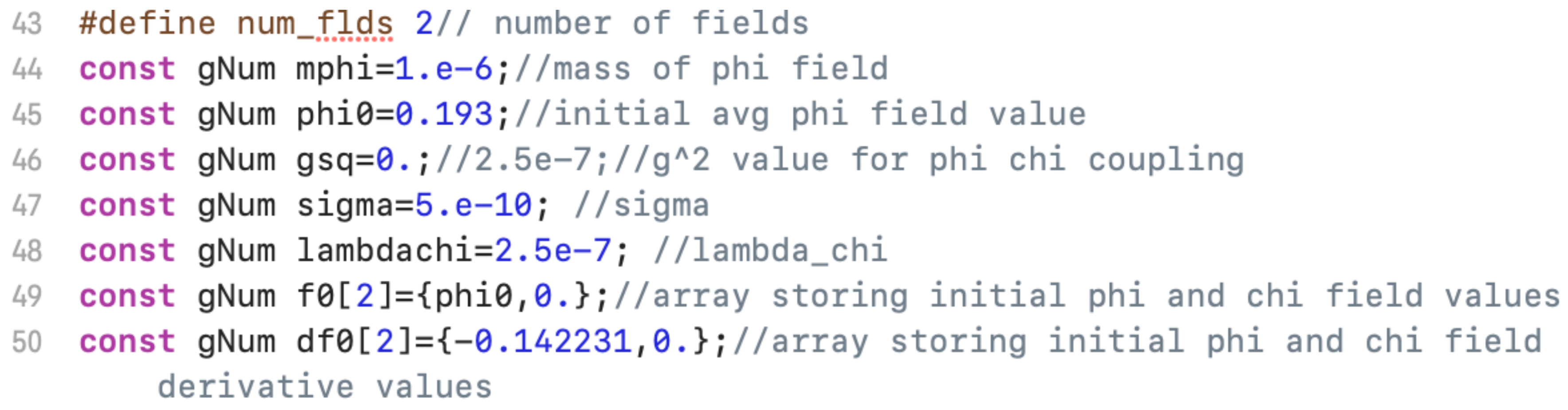
- When I calculate $V_{\rm pr}$, I get:

$$V_{\rm pr} = \frac{1}{2}\left(\phi'_{\rm pr}\right)^2 + \frac{\sigma m_{\rm pl}}{m^2}\sigma_{\rm pr}\phi_{\rm pr}\chi^2_{\rm pr} + \frac{\lambda_\chi m^2_{\rm pl}}{m^2}\frac{1}{4}\chi^4_{\rm pr}$$

- Which has two new dimensionless couplings.

Challenge 1

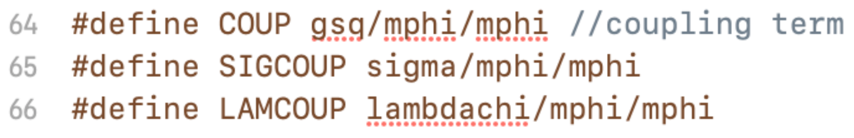
Part B



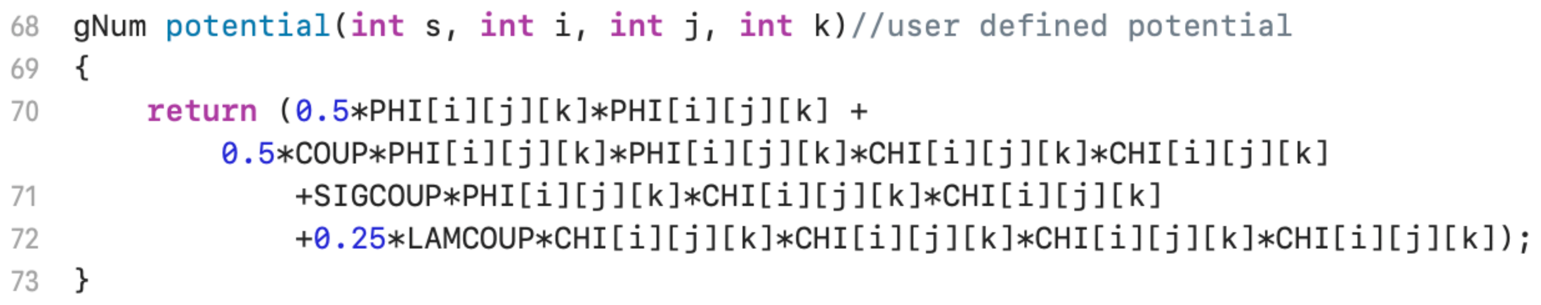
```
#define num_flds 2// number of fields
const gNum mphi=1.e-6;//mass of phi field
const gNum phi0=0.193;//initial avg phi field value
const gNum gsq=0.;//2.5e-7;//g^2 value for phi chi coupling
const gNum sigma=5.e-10; //sigma
const gNum lambdachi=2.5e-7; //lambda_chi
const gNum f0[2]={phi0,0.};//array storing initial phi and chi field values
const gNum df0[2]={-0.142231,0.};//array storing initial phi and chi field derivative values
```

Challenge 1

Part C



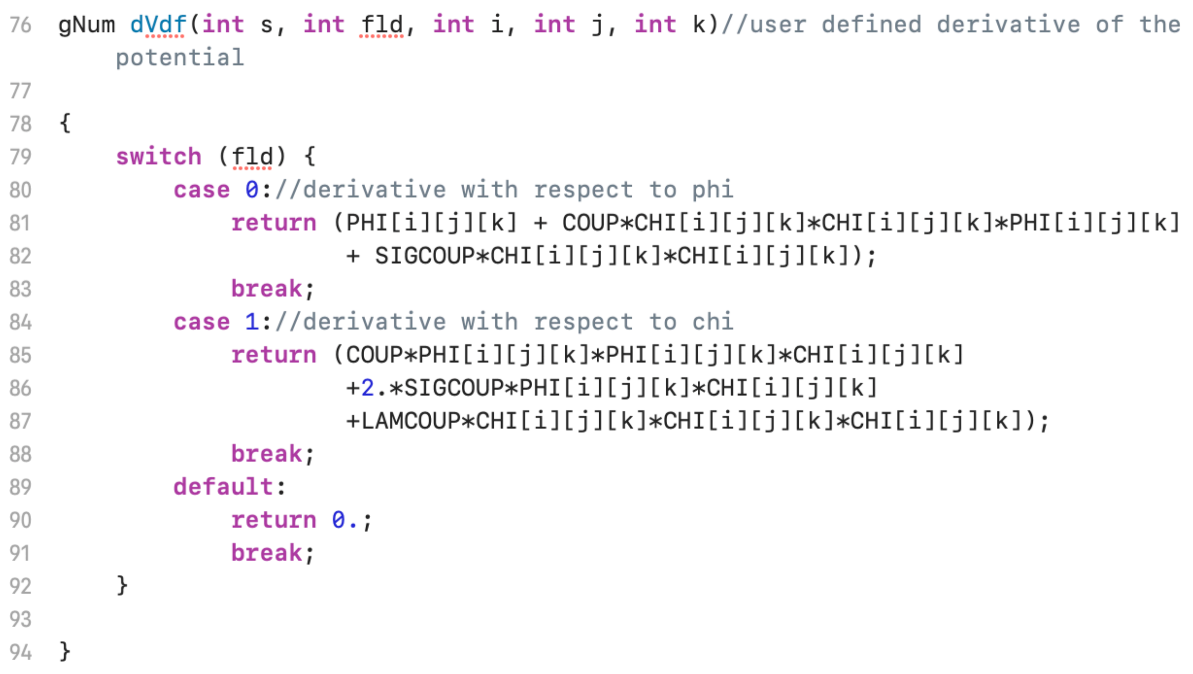
```
#define COUP gsg/mphi/mphi //coupling term
#define SIGCOUP sigma/mphi/mphi
#define LAMCOUP lambdachi/mphi/mphi
```



```
gNum potential(int s, int i, int j, int k)//user defined potential
{
    return (0.5*PHI[i][j][k]*PHI[i][j][k] +
        0.5*COUP*PHI[i][j][k]*PHI[i][j][k]*CHI[i][j][k]*CHI[i][j][k]
            +SIGCOUP*PHI[i][j][k]*CHI[i][j][k]*CHI[i][j][k]
            +0.25*LAMCOUP*CHI[i][j][k]*CHI[i][j][k]*CHI[i][j][k]*CHI[i][j][k]);
}
```

Challenge 1

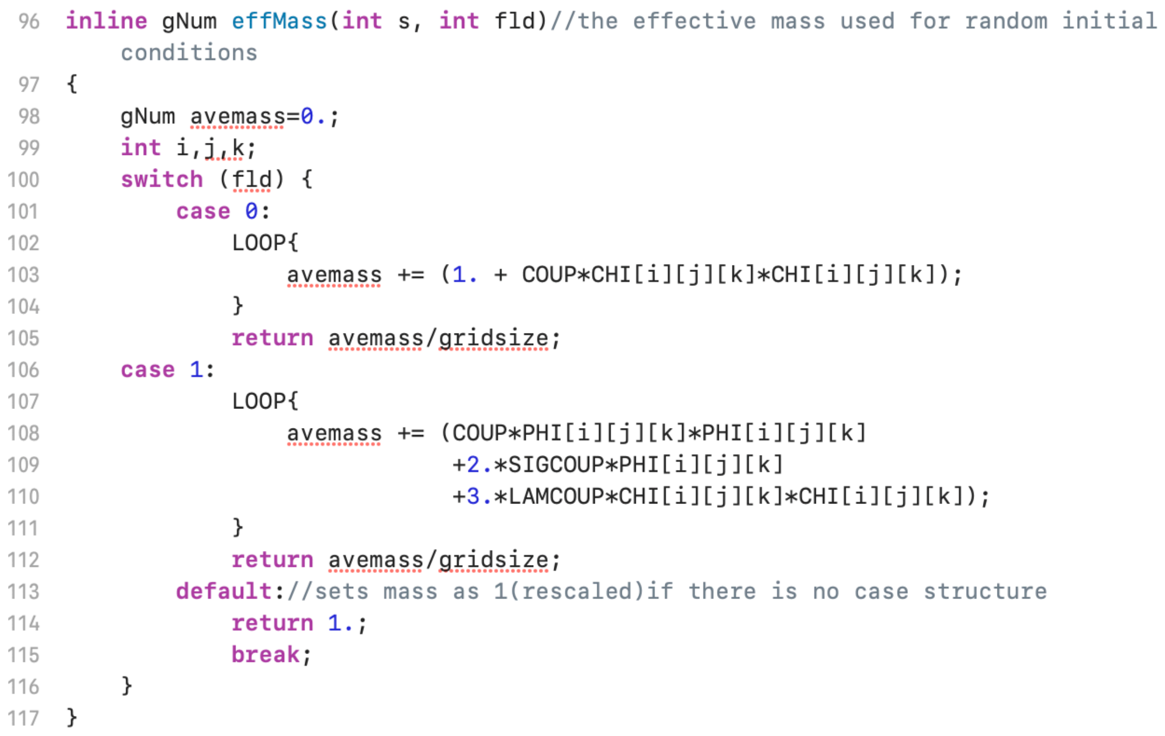
Part C



```
gNum dVdf(int s, int fld, int i, int j, int k)//user defined derivative of the potential

{
    switch (fld) {
        case 0://derivative with respect to phi
            return (PHI[i][j][k] + COUP*CHI[i][j][k]*CHI[i][j][k]*PHI[i][j][k]
                    + SIGCOUP*CHI[i][j][k]*CHI[i][j][k]);
            break;
        case 1://derivative with respect to chi
            return (COUP*PHI[i][j][k]*PHI[i][j][k]*CHI[i][j][k]
                    +2.*SIGCOUP*PHI[i][j][k]*CHI[i][j][k]
                    +LAMCOUP*CHI[i][j][k]*CHI[i][j][k]*CHI[i][j][k]);
            break;
        default:
            return 0.;
            break;
    }

}
```

Challenge 1

Part C



```
inline gNum effMass(int s, int fld)//the effective mass used for random initial conditions
{
    gNum avemass=0.;
    int i,j,k;
    switch (fld) {
        case 0:
            LOOP{
                avemass += (1. + COUP*CHI[i][j][k]*CHI[i][j][k]);
            }
            return avemass/gridsize;
    case 1:
            LOOP{
                avemass += (COUP*PHI[i][j][k]*PHI[i][j][k]
                            +2.*SIGCOUP*PHI[i][j][k]
                            +3.*LAMCOUP*CHI[i][j][k]*CHI[i][j][k]);
            }
            return avemass/gridsize;
        default://sets mass as 1(rescaled)if there is no case structure
            return 1.;
            break;
    }
}
```

Challenge 1

Part D

$$k_* = a\sqrt{-2\sigma\langle\phi\rangle} = 2 \times \sqrt{-2 \times \left(\frac{\sigma m_{\rm pl}}{m^2}\right) \times \langle\phi_{\rm min,pr} m_{\rm pl}\rangle m} = 2 \times \sqrt{2 \times 500 \times (0.05)}\, m$$

$$k_{*\,\rm pr} = \frac{k_*}{m} \approx 14$$

Challenge 1

Part E

$$H_{\mathrm{pr}} = \sqrt{\frac{8\pi}{3}\left[\frac{1}{2}\left(0.193\right)^2 + \frac{1}{2}\left(-0.142\right)^2\right]} \approx 0.5$$

$$L_{\mathrm{pr}} \sim \frac{2\pi}{H_{\mathrm{pr}}} \approx \pi$$

$$L_{\mathrm{pr}} = 3$$

Challenge 1

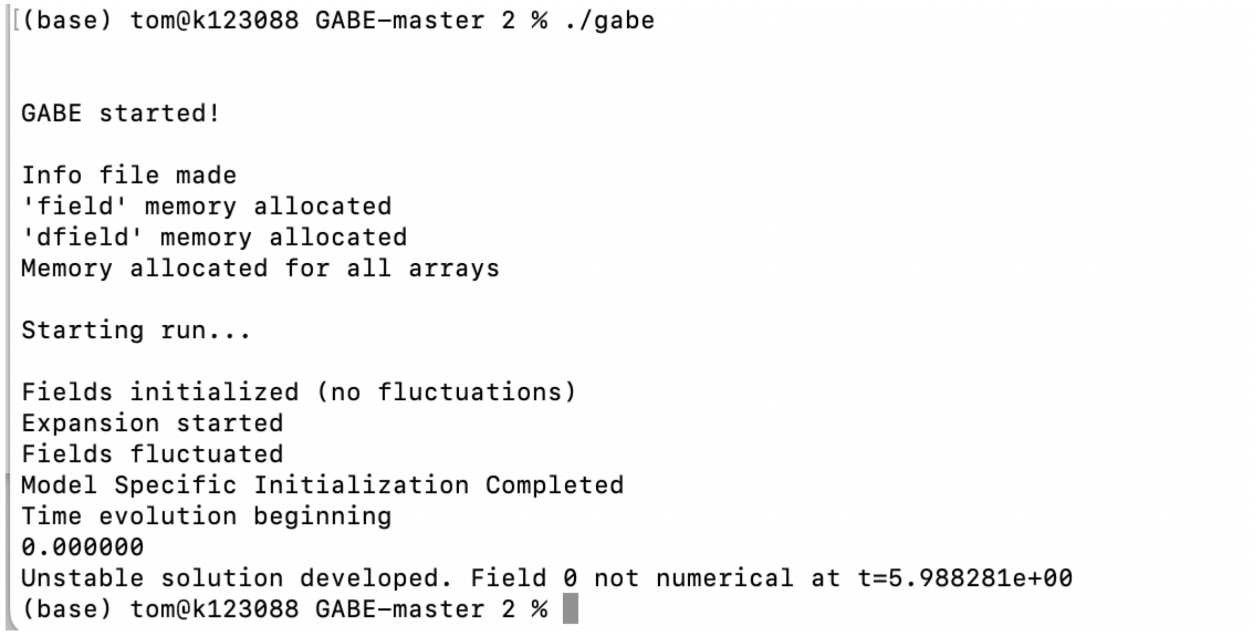
Part E

$$k_{\text{nyquist}} = \frac{2\pi}{L}\frac{\sqrt{3}}{2}N \approx 116\,m$$

When $L \approx 3$ and $N = 64$

Challenge 1

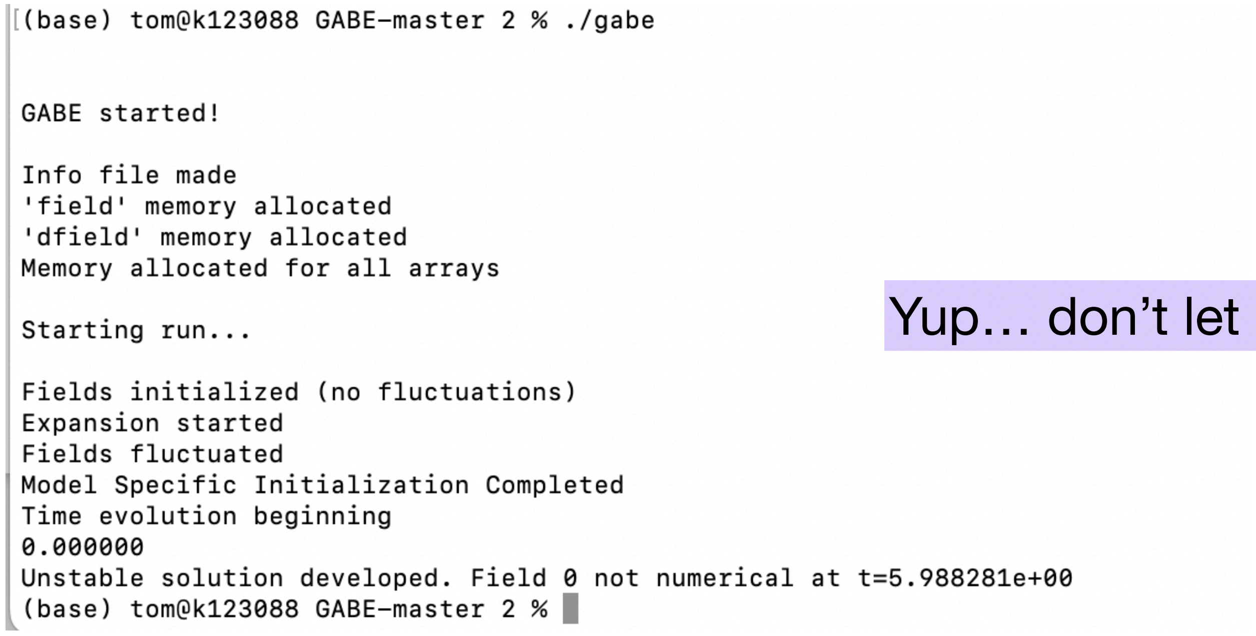
Part F



```
(base) tom@k123088 GABE-master 2 % ./gabe


GABE started!

Info file made
'field' memory allocated
'dfield' memory allocated
Memory allocated for all arrays

Starting run...

Fields initialized (no fluctuations)
Expansion started
Fields fluctuated
Model Specific Initialization Completed
Time evolution beginning
0.000000
Unstable solution developed. Field 0 not numerical at t=5.988281e+00
(base) tom@k123088 GABE-master 2 %
```

Challenge 1

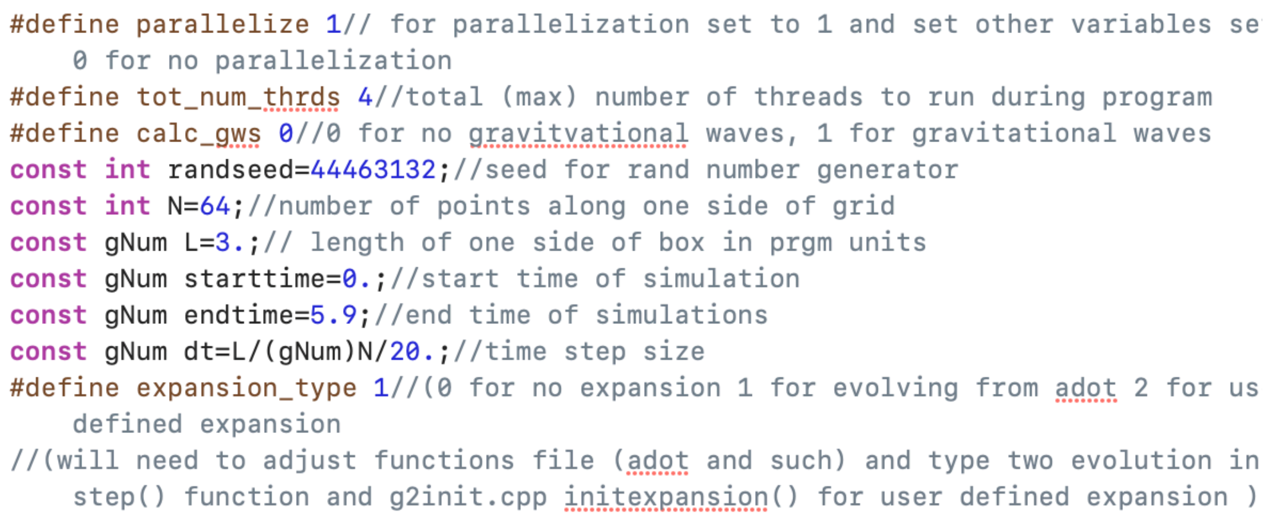
Part F



```
(base) tom@k123088 GABE-master 2 % ./gabe


GABE started!

Info file made
'field' memory allocated
'dfield' memory allocated
Memory allocated for all arrays

Starting run...

Fields initialized (no fluctuations)
Expansion started
Fields fluctuated
Model Specific Initialization Completed
Time evolution beginning
0.000000
Unstable solution developed. Field 0 not numerical at t=5.988281e+00
(base) tom@k123088 GABE-master 2 %
```

Yup… don’t let it get you down!

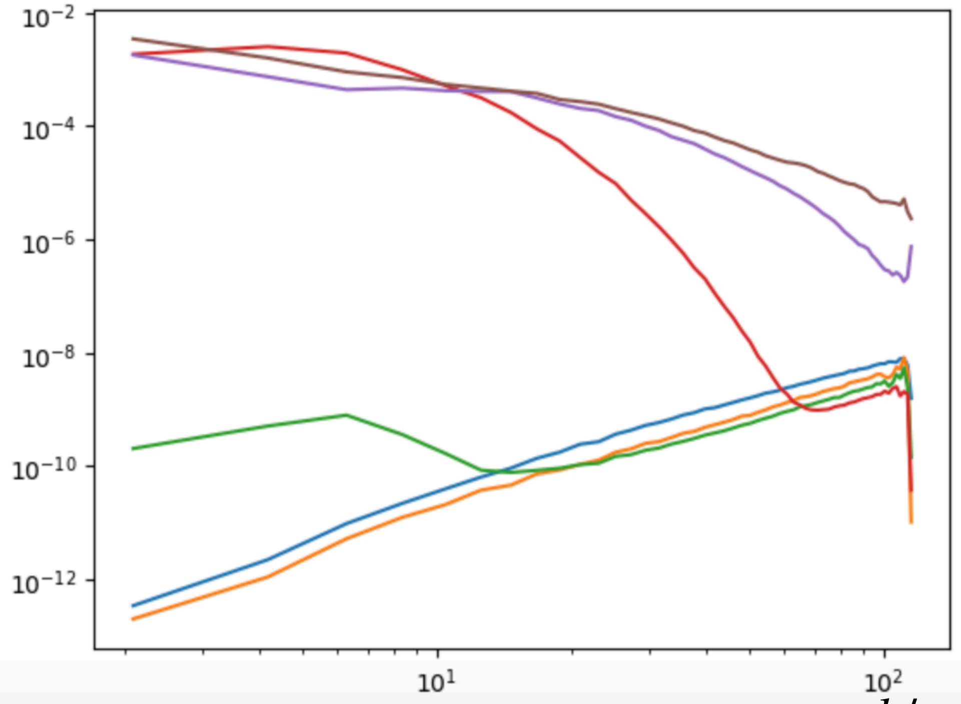
Just so it’s easier to plot



```
#define parallelize 1// for parallelization set to 1 and set other variables se
    0 for no parallelization
#define tot_num_thrds 4//total (max) number of threads to run during program
#define calc_gws 0//0 for no gravitvational waves, 1 for gravitational waves
const int randseed=44463132;//seed for rand number generator
const int N=64;//number of points along one side of grid
const gNum L=3.;// length of one side of box in prgm units
const gNum starttime=0.;//start time of simulation
const gNum endtime=5.9;//end time of simulations
const gNum dt=L/(gNum)N/20.;//time step size
#define expansion_type 1//(0 for no expansion 1 for evolving from adot 2 for us
    defined expansion
//(will need to adjust functions file (adot and such) and type two evolution in
    step() function and g2init.cpp initexpansion() for user defined expansion )
```

The Spectra!

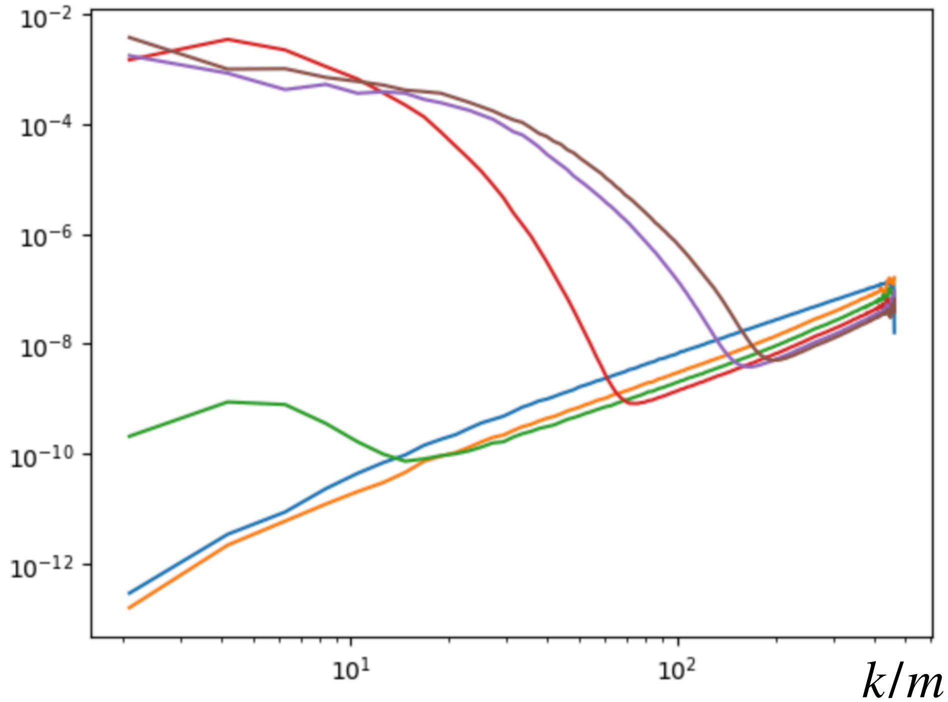
$\Delta^2(k)$



k/m

$N^3 = 64^3$

$\Delta^2(k)$



k/m

$N^3 = 256^3$

See if you can make some oscillons?