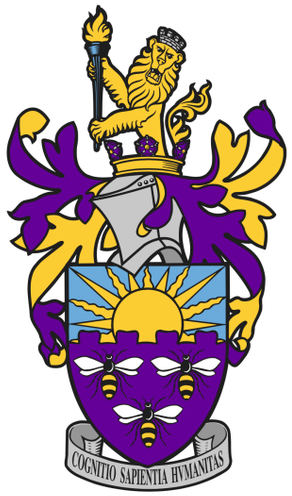
Bard School
Computing Lab

Cosmic Microwave Background

Boris Bolliet

Jodrell Bank Centre for Astrophysics
The University of Manchester

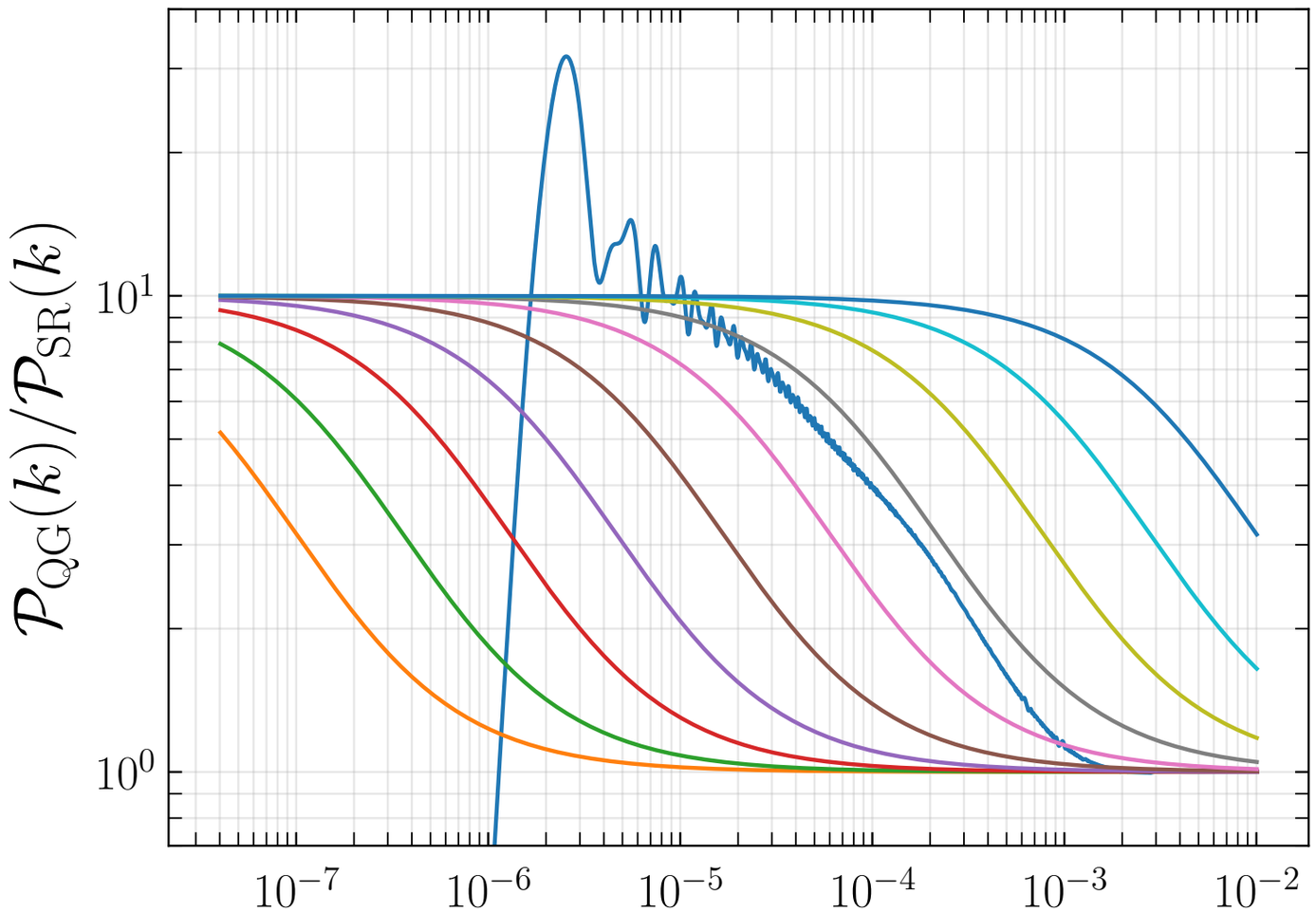


Day 2:

- Intro + questions
- Phenomenological parameterisation for primordial power spectrum
- Analytical power spectrum in CLASS
- Intro to MontePython

Phenomenological parameterisation of primordial power spectrum

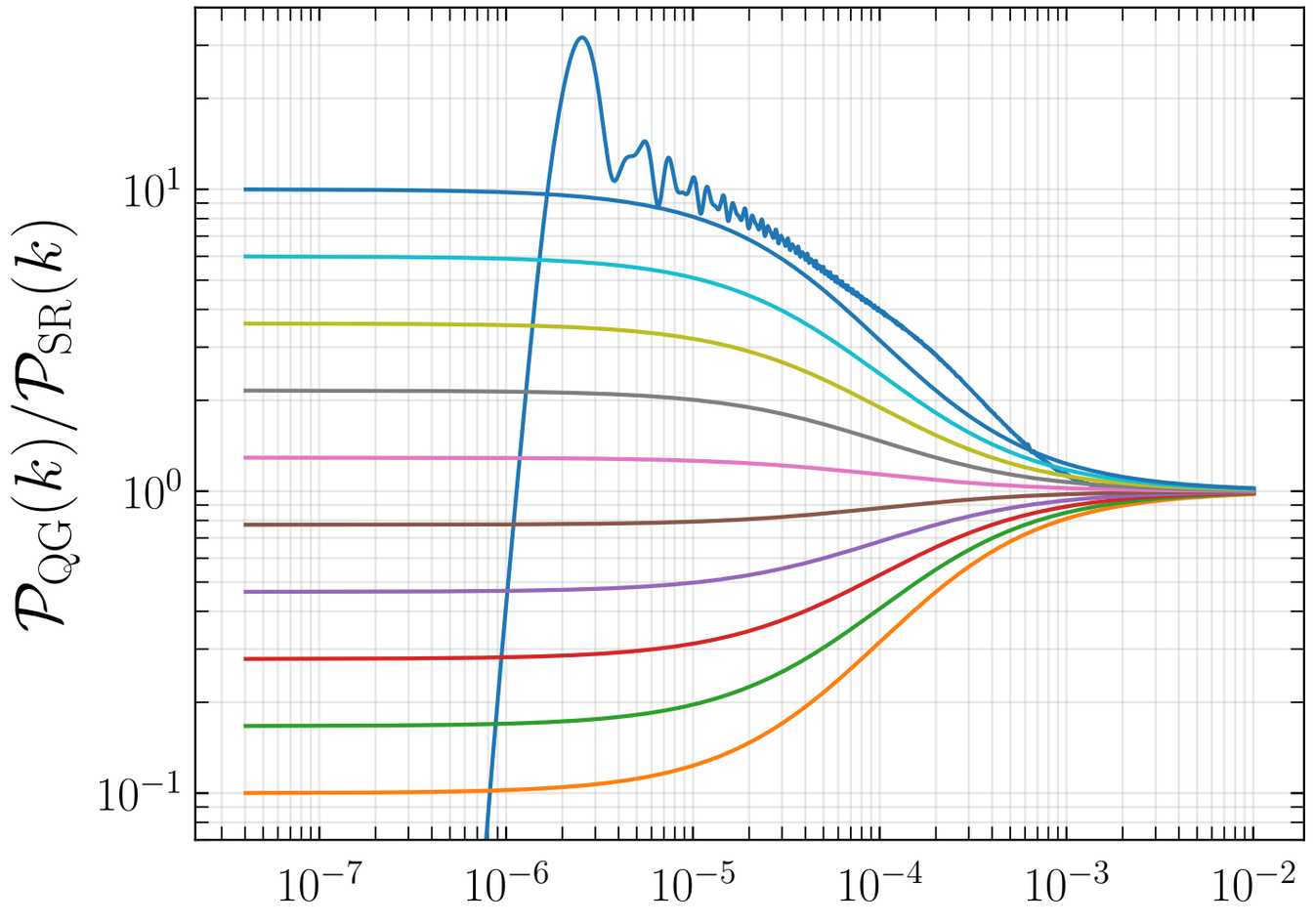
$$\frac{\mathcal{P}_{\rm QG}}{\mathcal{P}_{\rm SR}} = \exp\left(\frac{\ln\alpha}{1+\left(\frac{k}{k_{\rm QG}}\right)^{\beta}}\right)$$



Varying k_QG

Phenomenological parameterisation of primordial power spectrum

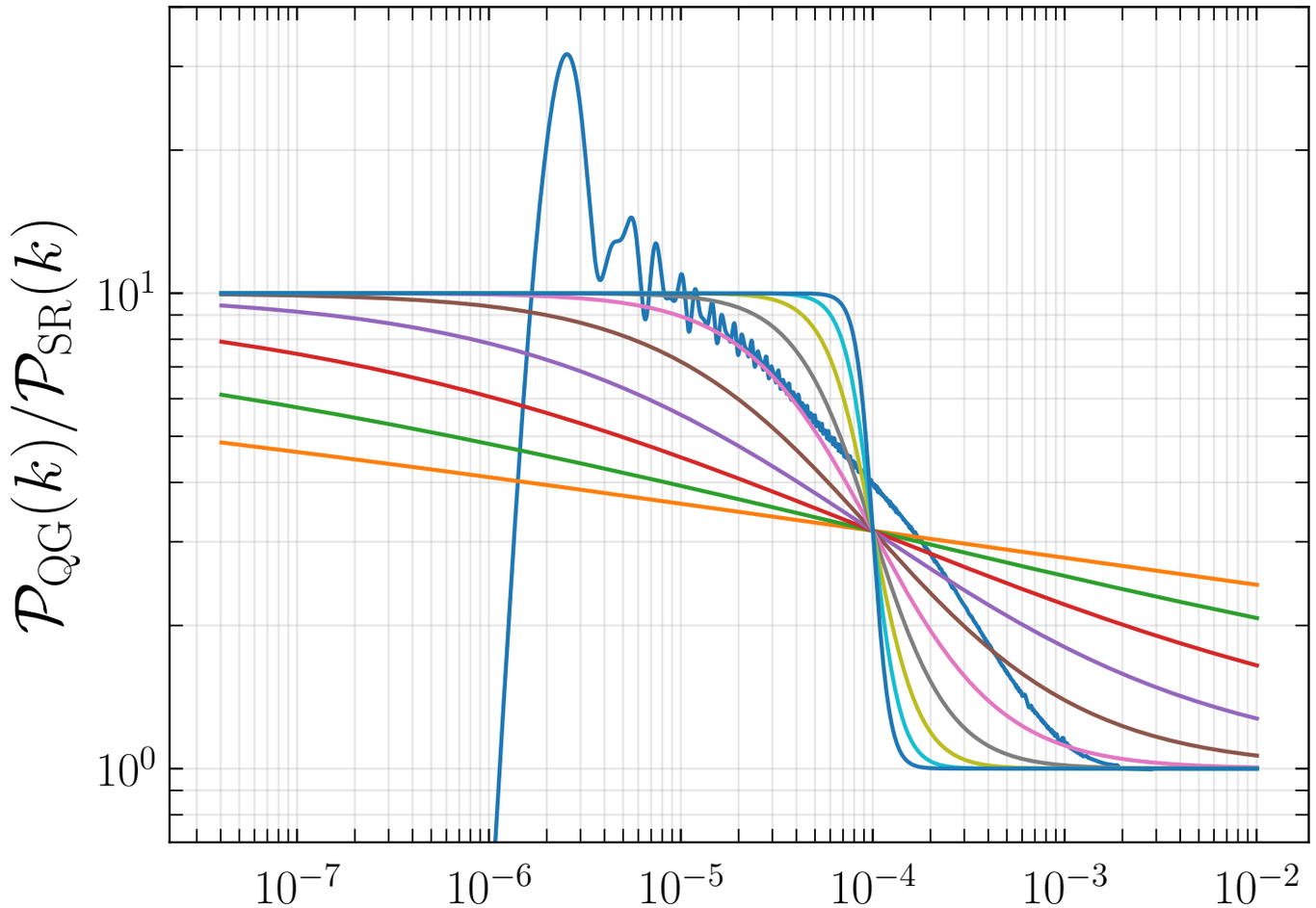
$$\frac{\mathcal{P}_{\rm QG}}{\mathcal{P}_{\rm SR}} = \exp\left(\frac{\ln\alpha}{1+\left(\frac{k}{k_{\rm QG}}\right)^{\beta}}\right)$$



Varying alpha

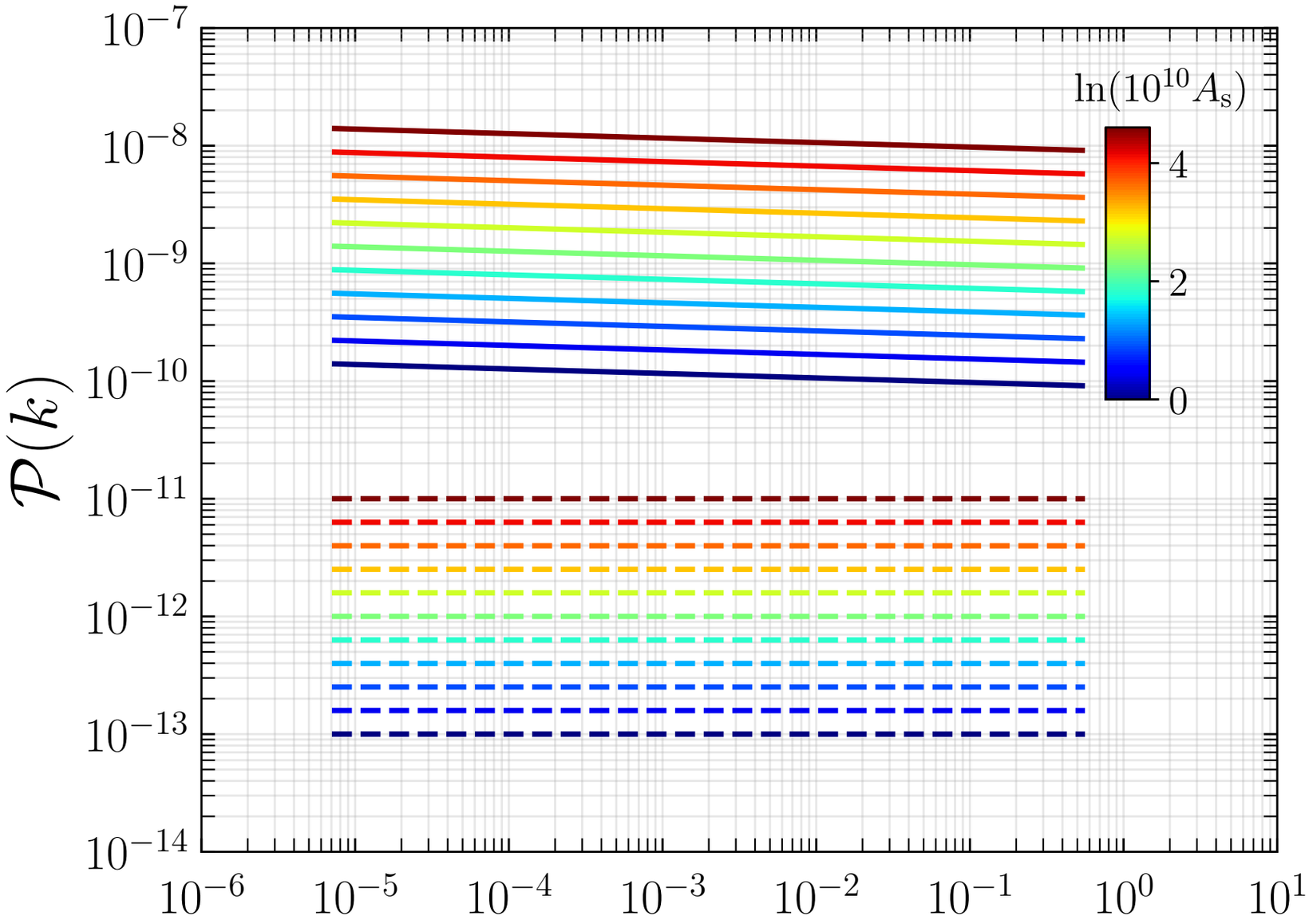
Phenomenological parameterisation of primordial power spectrum

$$\frac{\mathcal{P}_{\rm QG}}{\mathcal{P}_{\rm SR}} = \exp\left(\frac{\ln\alpha}{1+\left(\frac{k}{k_{\rm QG}}\right)^{\beta}}\right)$$



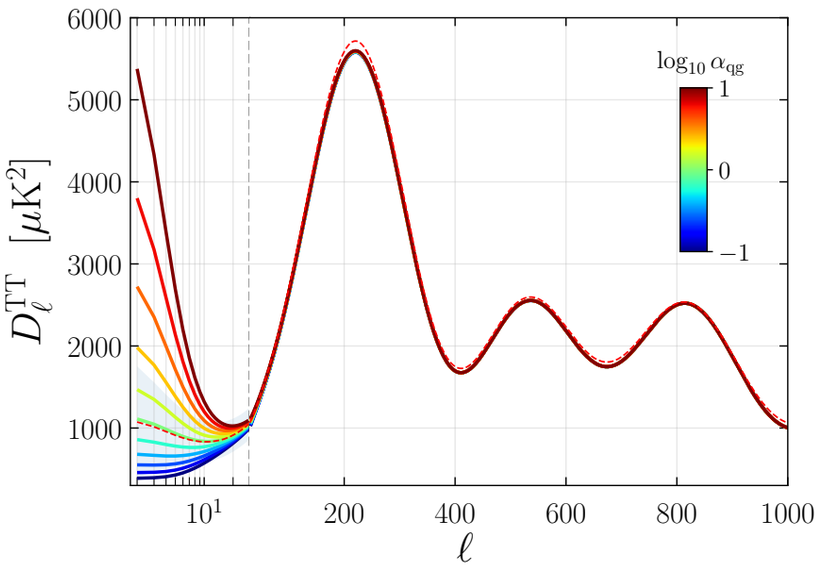
Varying beta

Computation with CLASS

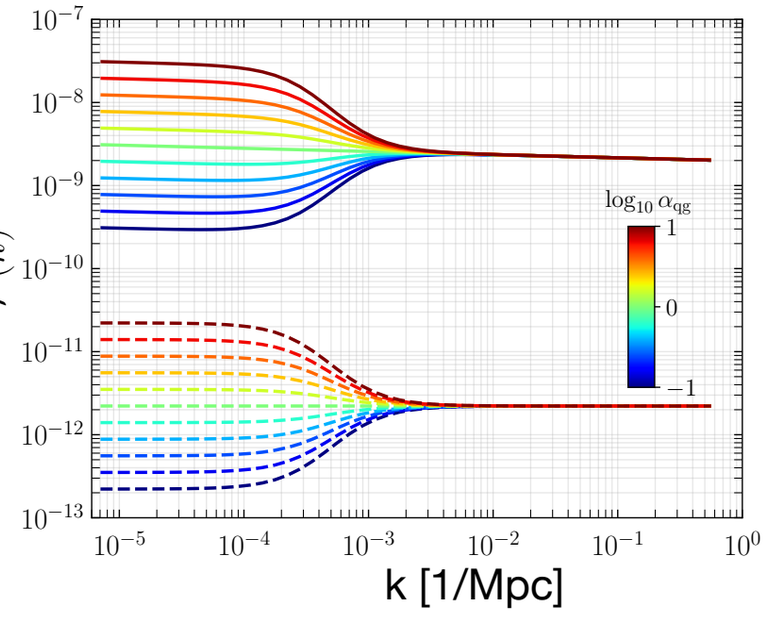


Exercise 1: use CLASS and Jupyter notebook to plot the primordial power Spectrum of both scalar and tensor perturbations for many values of As, ns and r.

Exercise 2: Find a three-parameters function that models a primordial power spectrum that has the same amplitude as SR power spectrum on small scales, but converges towards a different amplitude on large scales.



$$\frac{\mathcal{P}_{\rm QG}}{\mathcal{P}_{\rm SR}} = \exp\left(\frac{\ln\alpha}{1+\left(\frac{k}{k_{\rm QG}}\right)^{\beta}}\right)$$



Exercise 3: Plot the phenomenological power spectrum for many different values of the parameters, alpha, beta and k_QG.

Exercise 4: Implement the new phenomenological parameterisation into CLASS, and obtain the temperature anisotropy power spectrum for many different primordial power spectra.

Step-by-step solution:

- In source/input.c:

```
if (ppt->has_ad == _TRUE_) {

  class_read_double("n_s",ppm->n_s);
  class_read_double("alpha_s",ppm->alpha_s);

  class_read_int("has_qg",ppm->has_qg); //bard

  class_read_double("alpha_qg",ppm->alpha_qg); //bard
  class_read_double("beta_qg",ppm->beta_qg); //bard
  class_read_double("k_qg_over_k_pivot",ppm->k_qg_over_k_pivot); //bard

}
```

```
/** - primordial structure */

ppm->primordial_spec_type = analytic_Pk;
ppm->k_pivot = 0.05;
ppm->A_s = 2.215e-9;
ppm->n_s = 0.9619;
ppm->alpha_s = 0.;
ppm->has_qg = 0; //bard
ppm->alpha_qg = 1.; //bard
ppm->beta_qg = 1.; //bard
ppm->k_qg_over_k_pivot = 0.;  //bard
```

Step-by-step solution:

- In include/primordial.h:

```
/* - parameters describing the case primordial_spec_type = analytic_Pk : amplitudes, tilts, runnings,
    cross-correlations, ... */

double A_s;  /**< usual scalar amplitude = curvature power spectrum at pivot scale */
double n_s;  /**< usual scalar tilt = [curvature power spectrum tilt at pivot scale -1] */
double alpha_s; /**< usual scalar running */
double beta_s;  /**< running of running */

int has_qg;  //bard
double alpha_qg;  //bard
double beta_qg;  //bard
double k_qg_over_k_pivot;  //bard
```

```
//bard
int primordial_pheno_pk_qg_over_pr_sr(
                                      struct primordial * ppm
                                      double k,
                                      double * ratio
                                      );
```

Step-by-step solution:

- In source/primordial.c:

```
int primordial_analytic_spectrum(
                                 struct primordial * ppm,
                                 int index_md,
                                 int index_ic1_ic2,
                                 double k,
                                 double * pk
                                 ) {

  if (ppm->is_non_zero[index_md][index_ic1_ic2] == _TRUE_) {
    *pk = ppm->amplitude[index_md][index_ic1_ic2]
      *exp((ppm->tilt[index_md][index_ic1_ic2]-1.)*log(k/ppm->k_pivot)
           + 0.5 * ppm->running[index_md][index_ic1_ic2] *
               pow(log(k/ppm->k_pivot), 2.));


    if (ppm->has_qg == 1) {//bard
        double ratio_qg;
        class_call(primordial_pheno_pk_qg_over_pr_sr(ppm,k,&ratio_qg),
                   ppm->error_message,
                   ppm->error_message);
        *pk *= ratio_qg;
    }

}
else {
  *pk = 0.;
}

return _SUCCESS_;
```

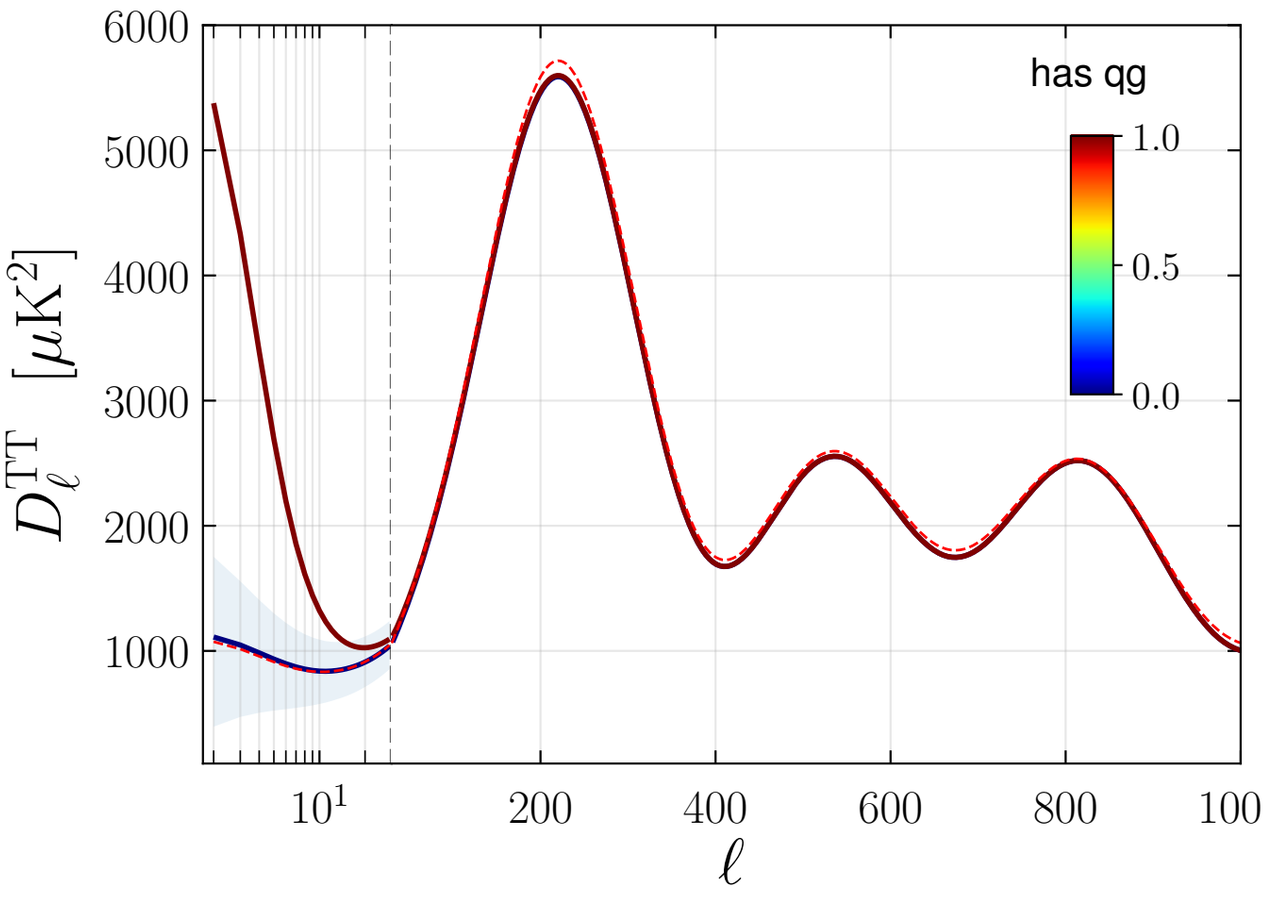
Step-by-step solution:

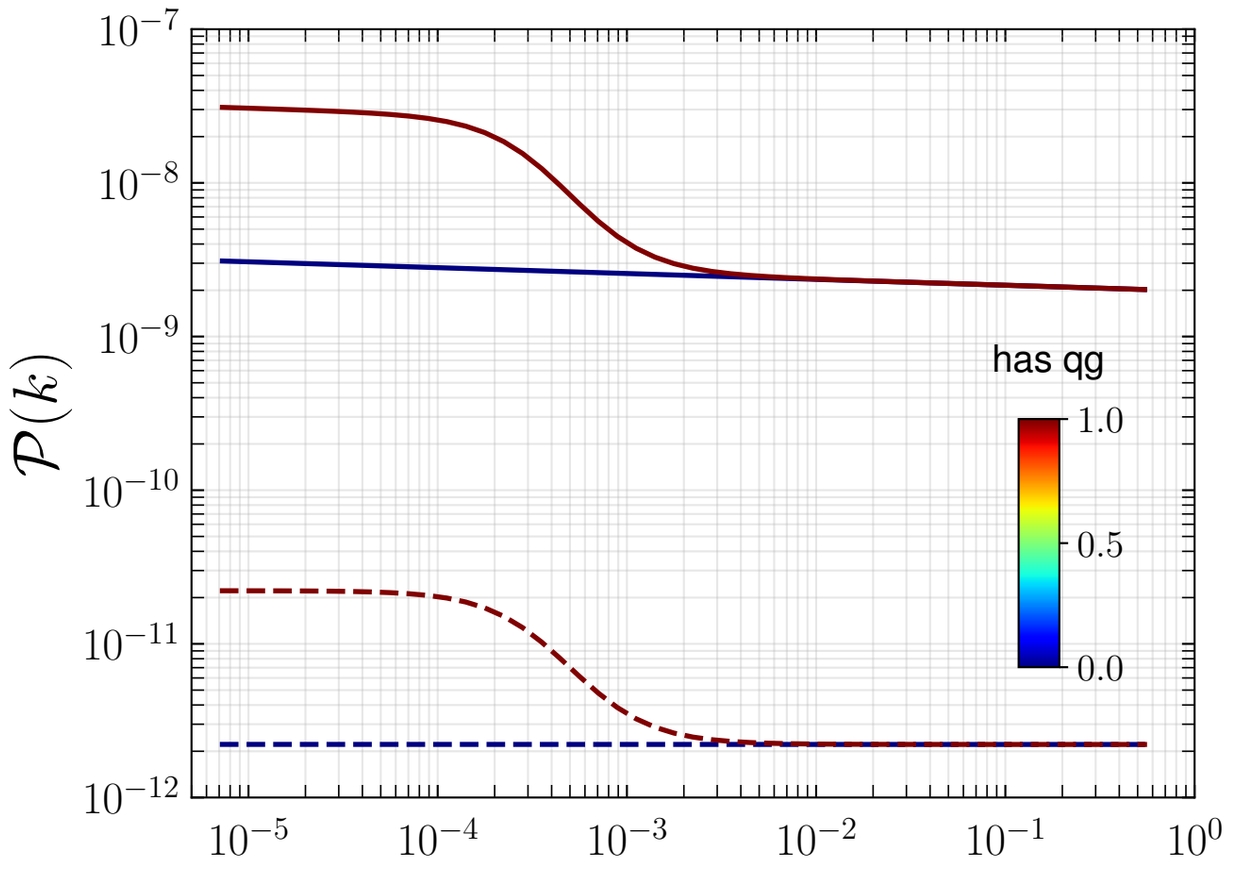
- In source/primordial.c:

```
//bard
int primordial_pheno_pk_qg_over_pr_sr(
                                      struct primordial * ppm
                                      double k,
                                      double * ratio
                                      ) {

    *ratio =
        exp(log(ppm->alpha_qg)/(1
        .+pow(k/ppm->k_pivot/ppm->k_qg_over_k_pivot,ppm->beta_qg)));
    return _SUCCESS_;
}
```

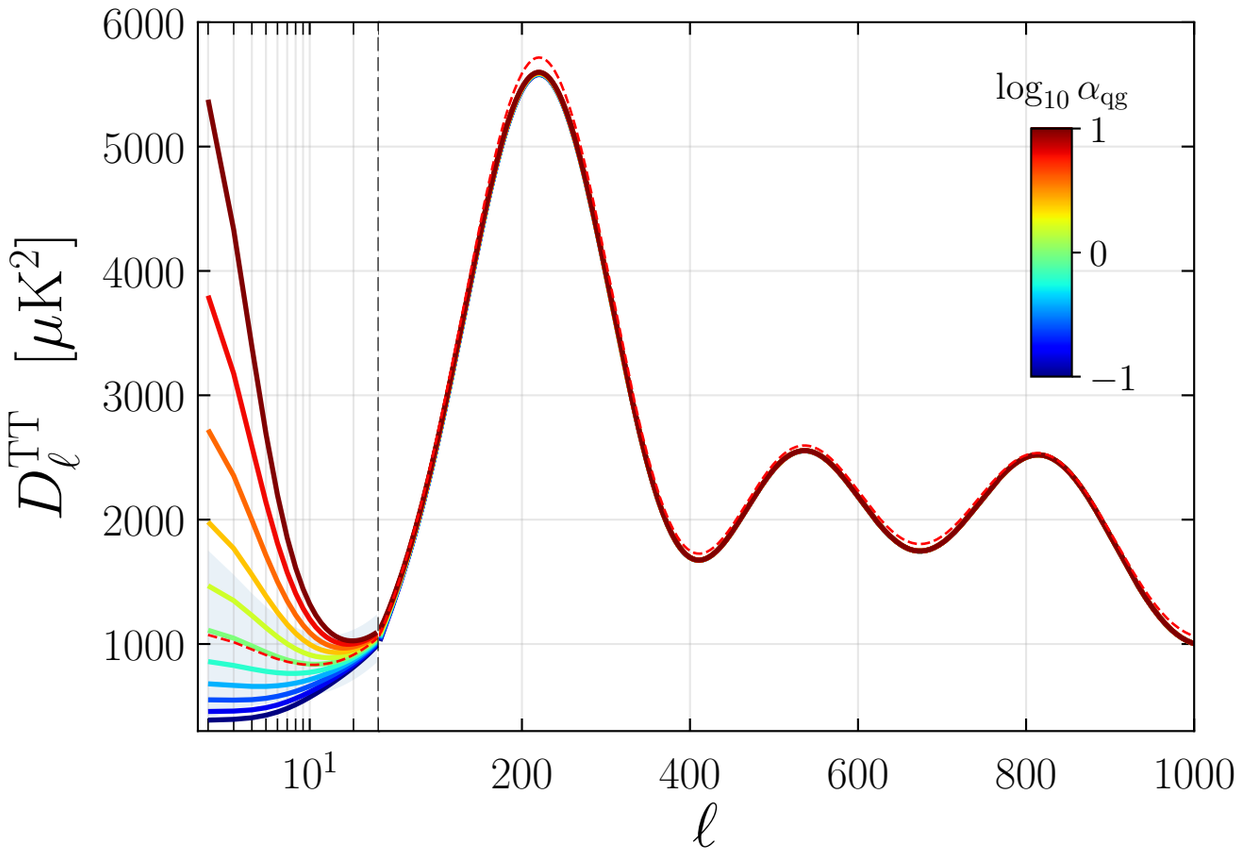
```
has_qg =1
alpha_qg = 1.e1
beta_qg = 2.
k_qg_over_k_pivot = 1.e-2
```

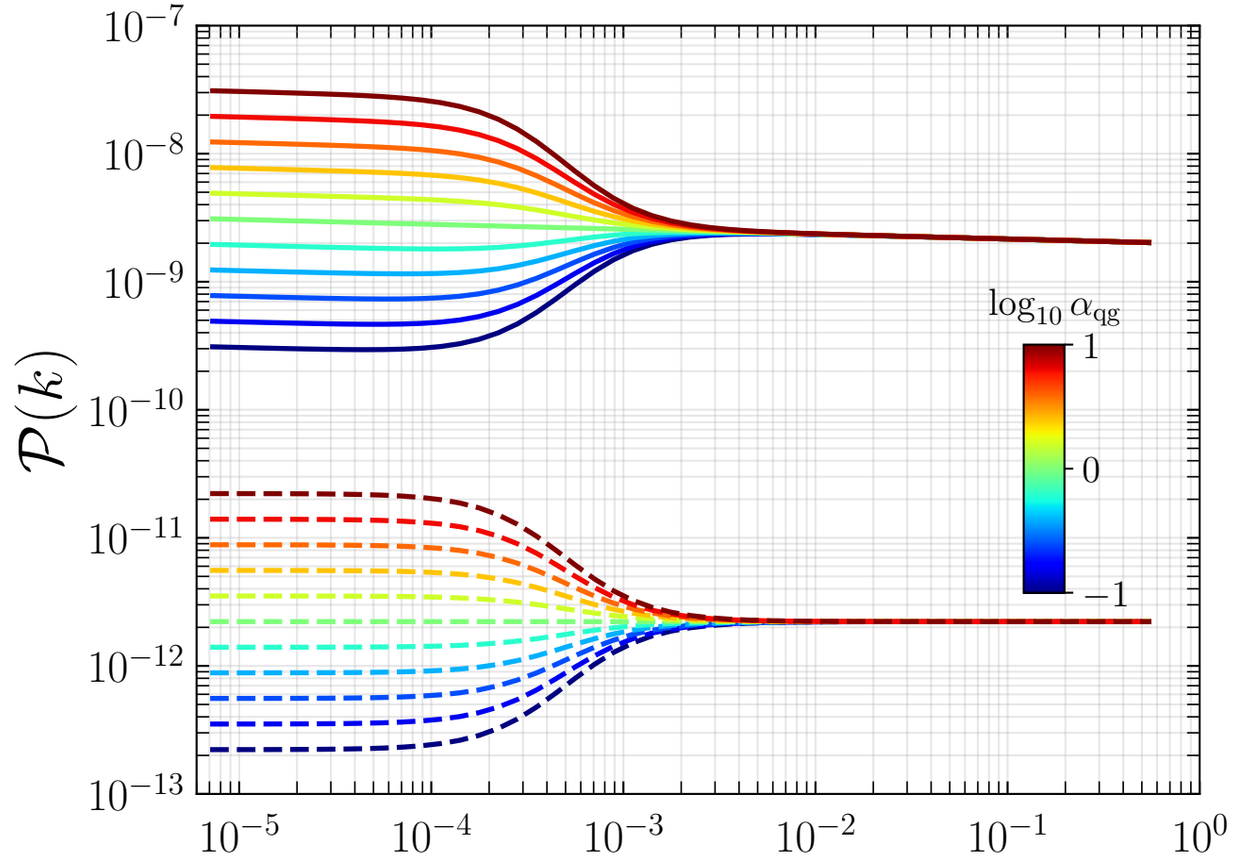




$$\frac{\mathcal{P}_{\rm QG}}{\mathcal{P}_{\rm SR}} = \exp\left(\frac{\ln\alpha}{1+\left(\frac{k}{k_{\rm QG}}\right)^{\beta}}\right)$$

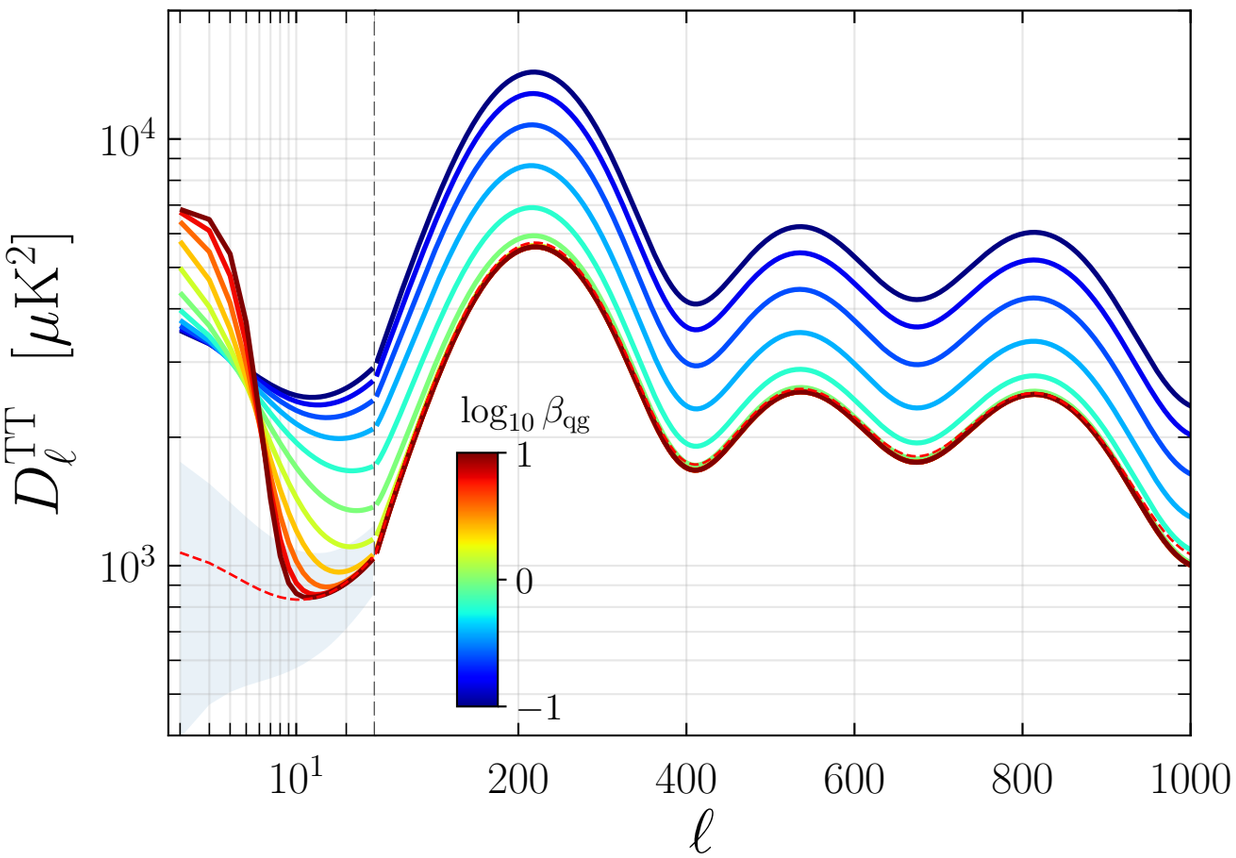
```
beta_qg = 2.
k_qg_over_k_pivot = 1.e-2
```

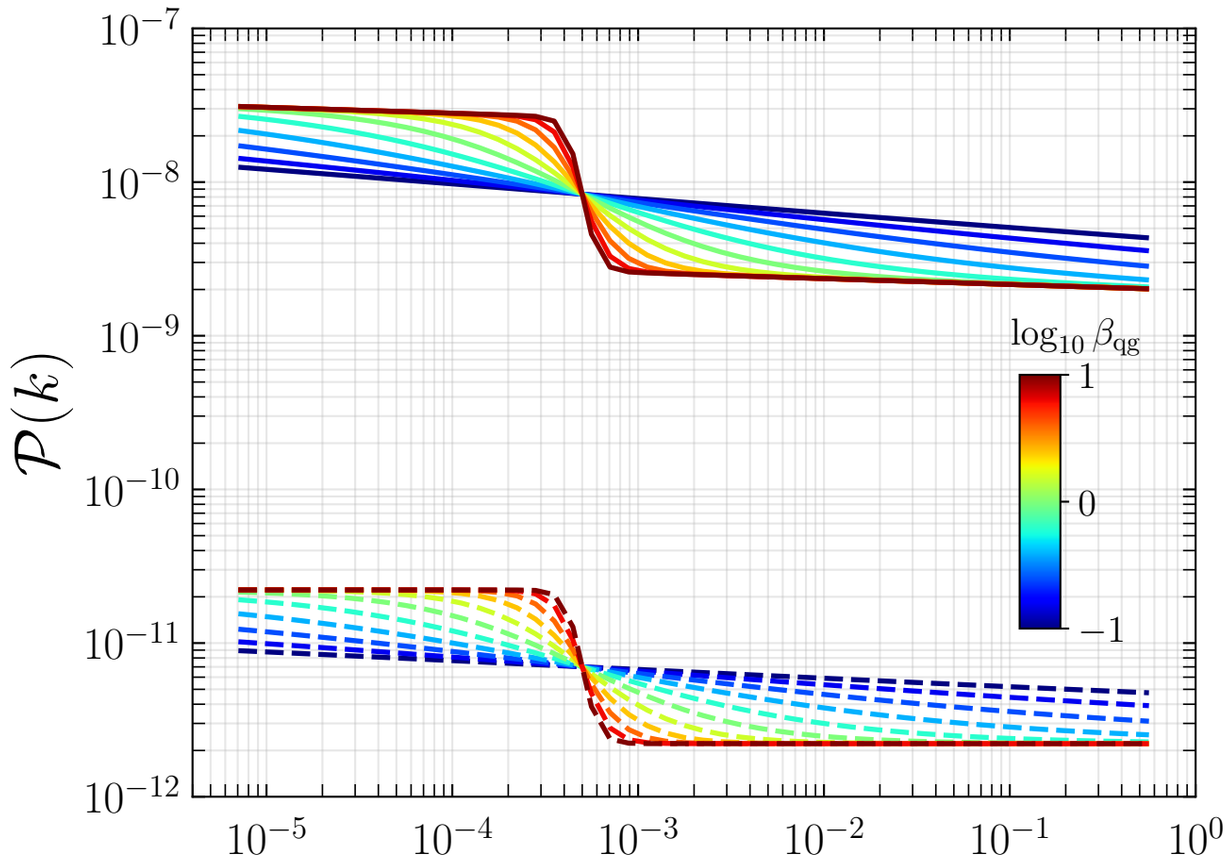




$$\frac{\mathcal{P}_{\rm QG}}{\mathcal{P}_{\rm SR}} = \exp\left(\frac{\ln\alpha}{1+\left(\frac{k}{k_{\rm QG}}\right)^{\beta}}\right)$$

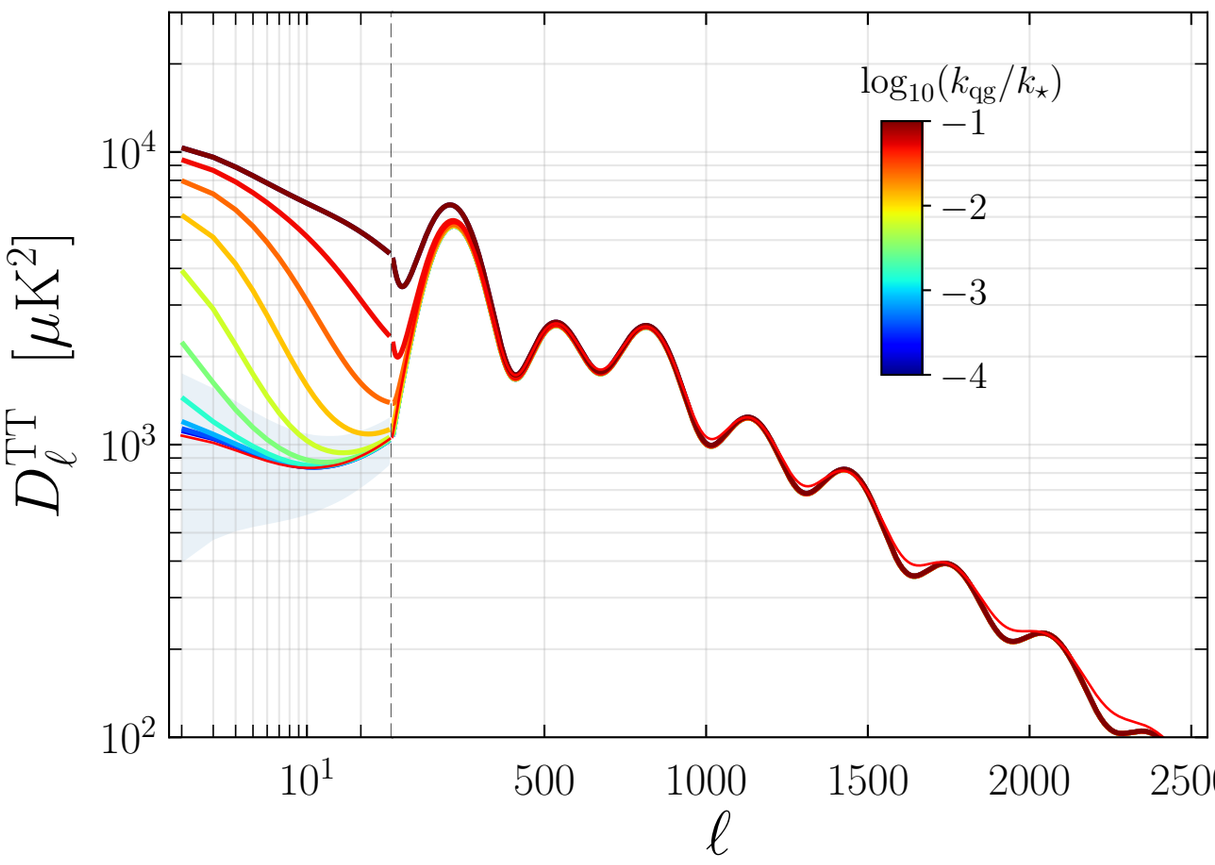
```
alpha_qg = 1.e1
beta_qg =10.0
k_qg_over_k_pivot = 1.e-2
```

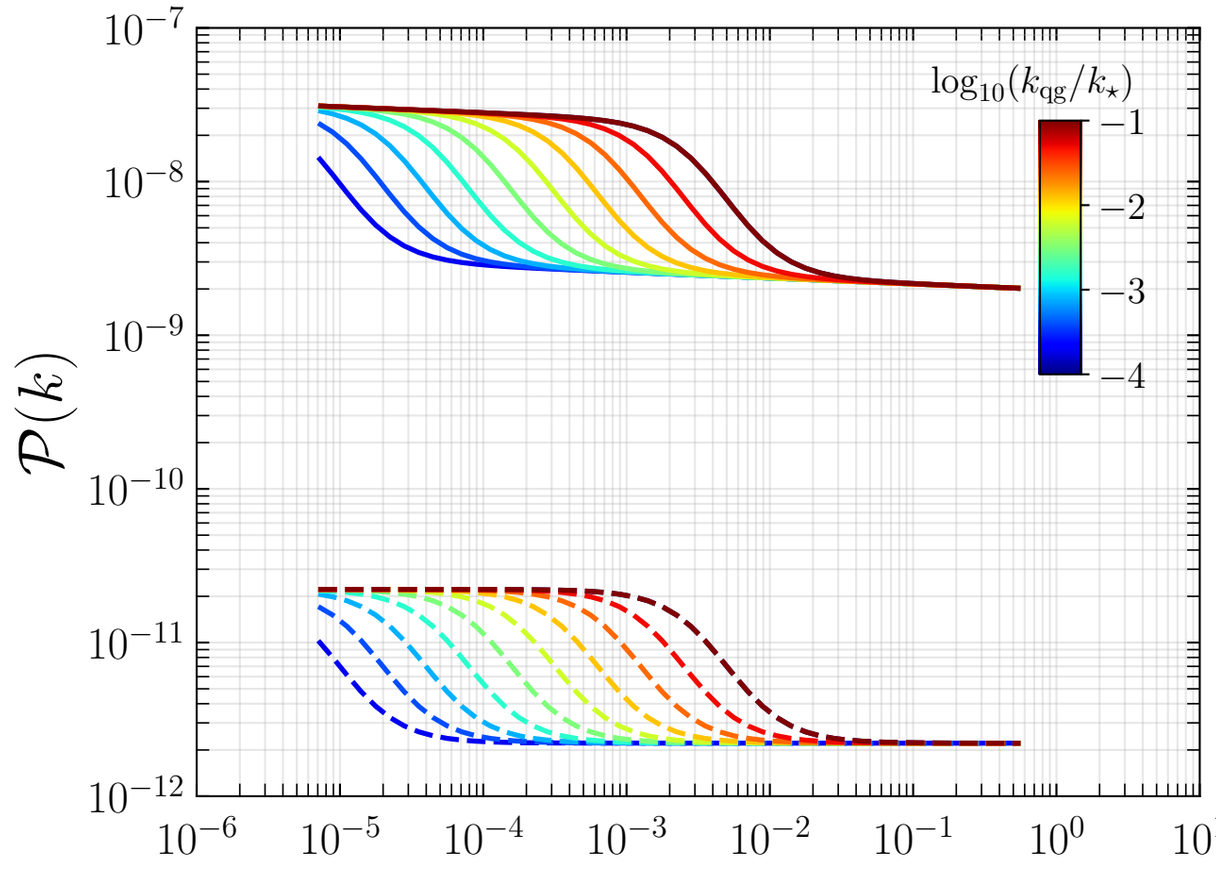




$$\frac{\mathcal{P}_{\rm QG}}{\mathcal{P}_{\rm SR}} = \exp\left(\frac{\ln\alpha}{1+\left(\frac{k}{k_{\rm QG}}\right)^{\beta}}\right)$$

```
alpha_qg = 1.e1
beta_qg = 2.
```





$$\frac{\mathcal{P}_{\rm QG}}{\mathcal{P}_{\rm SR}} = \exp\left(\frac{\ln\alpha}{1+\left(\frac{k}{k_{\rm QG}}\right)^{\beta}}\right)$$