Practice 4: Simulating GW emission from scalar and gauge fields

Nicolás Loayza
in collaboration with Jorge Baeza-Ballesteros

Simulating the early universe: $\mathcal{C}$osmo$\mathcal{L}$attice Workshop 2026

Overview

1. Numerical implementation in $\mathcal{C}$osmo$\mathcal{L}$attice
2. Diving into the GWs module
3. Running simulations with GWs
4. Physical applications

Overview

1. Numerical implementation in $\mathcal{C}$osmo$\mathcal{L}$attice
2. Diving into the GWs module
3. Running simulations with GWs
4. Physical applications

GWs equation of motion in $\mathcal{C}$osmo$\mathcal{L}$attice

- Simulate the evolution of **five** unphysical u-fields
- In $\mathcal{C}$osmo$\mathcal{L}$attice, we use **program variables**

$$\begin{aligned} \tilde{u}_{ij} &= \left(\frac{m_{\rm p}}{f_\star}\right)^2 u_{ij} \\ (\tilde{\pi}_u)_{ij} &= \frac{1}{\omega_\star}\left(\frac{m_{\rm p}}{f_\star}\right)(\pi_u)_{ij} = \frac{1}{\omega_\star}\left(\frac{m_{\rm p}}{f_\star}\right) a^{3-\alpha} u'_{ij} \end{aligned}$$

GWs equation of motion in $\mathcal{C}$osmo$\mathcal{L}$attice

- Simulate the evolution of **five** unphysical u-fields
- In $\mathcal{C}$osmo$\mathcal{L}$attice, we use **program variables**

$$\begin{aligned} \tilde{u}_{ij} &= \left(\frac{m_{\rm p}}{f_\star}\right)^2 u_{ij} \\ (\tilde{\pi}_u)_{ij} &= \frac{1}{\omega_\star}\left(\frac{m_{\rm p}}{f_\star}\right)(\pi_u)_{ij} = \frac{1}{\omega_\star}\left(\frac{m_{\rm p}}{f_\star}\right) a^{3-\alpha} u'_{ij} \end{aligned}$$

- Evolve following the equations of motion

$$\begin{aligned} \tilde{\partial}_0 \tilde{u}_{ij} &= a^{\alpha-3}(\tilde{\pi}_u)_{ij} \\ \tilde{\partial}_0 (\tilde{\pi}_u)_{ij} &= a^{1+\alpha}\tilde{\nabla}_i^- \tilde{\nabla}_i^+ u_{ij} + 2a^{1+\alpha}\left(\tilde{\Pi}_{ij} - \tfrac{1}{3}\delta_{ij}\tilde{\Pi}_{kk}\right) \end{aligned}$$

GWs equation of motion in $\mathcal{C}$osmo$\mathcal{L}$attice

- Simulate the evolution of **five** unphysical u-fields
- In $\mathcal{C}$osmo$\mathcal{L}$attice, we use **program variables**

$$\begin{aligned} \tilde{u}_{ij} &= \left(\frac{m_{\rm p}}{f_\star}\right)^2 u_{ij} \\ (\tilde{\pi}_u)_{ij} &= \frac{1}{\omega_\star}\left(\frac{m_{\rm p}}{f_\star}\right)(\pi_u)_{ij} = \frac{1}{\omega_\star}\left(\frac{m_{\rm p}}{f_\star}\right) a^{3-\alpha} u'_{ij} \end{aligned}$$

- Evolve following the equations of motion

$$\begin{aligned} \tilde{\partial}_0 \tilde{u}_{ij} &= a^{\alpha-3}(\tilde{\pi}_u)_{ij} \\ \tilde{\partial}_0 (\tilde{\pi}_u)_{ij} &= a^{1+\alpha}\tilde{\nabla}_i^- \tilde{\nabla}_i^+ u_{ij} + 2a^{1+\alpha}\left(\tilde{\Pi}_{ij} - \tfrac{1}{3}\delta_{ij}\tilde{\Pi}_{kk}\right) \end{aligned}$$

Simulations are performed **analogously to scalar fields**

Measures of GW observables

- Physical degrees of freedom are **only constructed when measuring**

$$h_{ij}(\tilde{\boldsymbol{k}}, \tilde{\eta}) = \left(\frac{f_\star}{m_{\rm p}}\right)^2 \Lambda_{ij,kl}(\tilde{\boldsymbol{k}}) u_{kl}(\tilde{\boldsymbol{k}}, \tilde{\eta})$$

- In $\mathcal{C}$osmo$\mathcal{L}$attice we measure the **fractional energy density power spectrum** and the **total energy** of GWs

$$\begin{aligned} \Omega_{\rm GW}(\tilde{k}, \tilde{\eta}) &= \frac{1}{\tilde{\rho}_{\rm tot}} \frac{{\rm d}\tilde{\rho}_{\rm GW}}{{\rm d}\log\tilde{k}} \\ \tilde{\rho}_{\rm GW} &= \tilde{\rho}_{\rm tot} \sum_k \Omega(\tilde{k}, \tilde{\eta}) \Delta \log k \end{aligned}$$

where $\tilde{\rho}_{\rm tot}$ is the **total energy of the matter fields**

Source of gravitational waves

- GWs are sourced by the **anisotropic stress tensor**
- $\mathcal{C}$osmo$\mathcal{L}$attice is prepared to simulate GWs sourced by **real and complex scalar fields** and **U(1) gauge fields**

$$\begin{aligned}\tilde{\Pi}_{ij} &= \sum_a \tilde{\nabla}_i^+ \tilde{\phi}_a \tilde{\nabla}_i^+ \tilde{\phi}_a + \sum_b 2\mathrm{Re}\left[\tilde{\nabla}_i^+ \tilde{\varphi}_b \tilde{\nabla}_i^+ \tilde{\varphi}_b\right] \\ &- \left(\frac{\omega_\star}{f_\star}\right)^2 \left[a^{-2\alpha} \tilde{E}_i \tilde{E}_j + a^{-2} \tilde{B}_i \tilde{B}_j\right]\end{aligned}$$

Overview

1. Numerical implementation in $\mathcal{C}$osmo$\mathcal{L}$attice
2. Diving into the GWs module
3. Running simulations with GWs
4. Physical applications

GW module in $\mathcal{C}$osmo$\mathcal{L}$attice

Please, open the following files:

- `src/model/parameter-files/lphi4.in`
- `src/include/CosmoInterface/evolvers/leapfrog.h`
- `src/include/CosmoInterface/evolvers/kernels/gwskernels.h`
- `src/include/CosmoInterface/definitions/PITensor.h`
- `src/include/CosmoInterface/definitions/GWsProjector.h`
- `src/include/CosmoInterface/measurements/gwspowerspectrum.h`

GWs in $\mathcal{C}$osmo$\mathcal{L}$attice: input parameters

src/model/parameter-files/lphi4.in

```
13 #Times
14 tOutputFreq = 0.1
15 tOutputInfreq = 1
16 tMax = 300
17
18
19 #Spectra options
20 PS_type = 1
21 PS_version = 1
22
23 #GWs
24 GWprojectorType = 1
25 withGWs = false
26
27 #IC
28 kCutOff = 4
29 initial_amplitudes = 5.6964e18 0  # homogeneous amplitudes in GeV
```

GWs in $\mathcal{C}$osmo$\mathcal{L}$attice: input parameters

src/model/parameter-files/lphi4.in

```
13 #Times
14 tOutputFreq = 0.1
15 tOutputInfreq = 1
16 tMax = 300
17
18
19 #Spectra options
20 PS_type = 1
21 PS_version = 1
22
23 #GWs
24 GWprojectorType = 1
25 withGWs = false
26
27 #IC
28 kCutOff = 4
29 initial_amplitudes = 5.6964e18 0  # homogeneous amplitudes in GeV
```

The GWs module is controlled by two parameters:

1. `withGWs`: boolean used to turn on/off GWs
2. `GWprojectorType`: choose between different projectors
   - `GWprojectorType = 1`: neutral derivative
   - `GWprojectorType = 2`: forward derivative (default)
   - `GWprojectorType = 3`: backward derivative

GWs in $\mathcal{C}$osmo$\mathcal{L}$attice: output results

We get two different output files

1. `spectra_gws.txt` → $\dfrac{1}{\tilde{\rho}_{\rm tot}}\dfrac{{\rm d}\tilde{\rho}_{\rm GW}}{{\rm d}\log k}$

κ	$\Omega_{\rm GW}$	Multiplicity
0.4	3.455976932183128e-29	13
0.8	1.13549488356703e-28	37
1.2	1.65395001818171 4e-28	57
⋮	⋮	⋮

2. `energy_gws.txt`

$\tilde{\eta}$	$\tilde{\rho}_{\rm GW}/\tilde{\rho}_{\rm tot}$	$\tilde{\rho}_{\rm GW} = \rho_{\rm GW}/\omega_*^2 f_*^2$
0	0	0
1	1.2669487198562e-27	5.2978770016331 1e-29
2	2.75524033399837e-27	1.24679240886064e-29
⋮	⋮	⋮

GWs in $\mathcal{C}$osmo$\mathcal{L}$attice: equation of motion

```
136     template<class Model>
137     void kickGWs(Model& model, T w) {
138         ForLoop(n, 0, Model::NGWs - 1,
139                 (*model.piGWs)(n) += (w * mod
                ↪  GWsKernels::get(model,n)
140         );
141     }
```

$$u'_{ij} = a^{\alpha-3}(\pi_u)_{ij}$$

```
203     template<class Model>
204     void driftGWs(Model& model) {
205
206         (*model.fldGWs) +=  pow(model.a
            ↪  (model.dt * (*model.piGWs) );
207
208     }
```

$$(\pi_u)'_{ij} = \mathcal{K}_{ij}[u_{ij}, \{\tilde{\phi}\}, a]$$

GWs in $\mathcal{C}$osmo$\mathcal{L}$attice: $\mathcal{K}_{ij}[u_{ij}, \{\tilde{\phi}\}, a]$

src/include/CosmoInterface/evolvers/kernels/gwskernels.h

```
30      template <class Model, int N>
31      static auto get(Model& model, Tag<N> n){
32
33          return (pow(model.aI, 1 + model.alpha) *
34                 LatLapl<Model::NDim>( (*model.fldGWs)(n))
35                  + pow(model.aI, 1 + model.alpha) * 2. *
                    ↪  (PITensor::totalTensor(model,n)));
36      }
```

$$\mathcal{K}_{ij}[u_{ij}, \{\tilde{\phi}\}, a] = a^{1+\alpha}\tilde{\nabla}^2_{\mathrm{L}} u_{ij} + 2a^{1+\alpha}\left(\frac{f_*}{m_{\mathrm{Pl}}}\right)^2\left(\tilde{\Pi}_{ij} - \frac{1}{3}\delta_{ij}\tilde{\Pi}_{kk}\right)$$

GWs in $\mathcal{C}$osmo$\mathcal{L}$attice: $\mathcal{K}_{ij}[u_{ij}, \{\tilde{\phi}\}, a]$

src/include/CosmoInterface/evolvers/kernels/gwskernels.h

```
30      template <class Model, int N>
31      static auto get(Model& model, Tag<N> n){
32
33          return (pow(model.aI, 1 + model.alpha) *
34                 LatLapl<Model::NDim>( (*model.fldGWs)(n))
35                  + pow(model.aI, 1 + model.alpha) * 2. *
                    ↪  (PITensor::totalTensor(model,n)));
36      }
```

$$\mathcal{K}_{ij}[u_{ij}, \{\tilde{\phi}\}, a] = a^{1+\alpha}\tilde{\nabla}^2_{\mathrm{L}} u_{ij} + 2a^{1+\alpha}\left(\frac{f_*}{m_{\mathrm{Pl}}}\right)^2\left(\tilde{\Pi}_{ij} - \frac{1}{3}\delta_{ij}\tilde{\Pi}_{kk}\right)$$

GWs in $\mathcal{C}$osmo$\mathcal{L}$attice: $\mathcal{K}_{ij}[u_{ij}, \{\tilde{\phi}\}, a]$

src/include/CosmoInterface/evolvers/kernels/gwskernels.h

```
30      template <class Model, int N>
31      static auto get(Model& model, Tag<N> n){
32
33          return (pow(model.aI, 1 + model.alpha) *
34                 LatLapl<Model::NDim>( (*model.fldGWs)(n))
35                  + pow(model.aI, 1 + model.alpha) * 2. *
                    ↪  (PITensor::totalTensor(model,n)));
36      }
```

$$\mathcal{K}_{ij}[u_{ij}, \{\tilde{\phi}\}, a] = a^{1+\alpha}\tilde{\nabla}^2_{\mathsf{L}} u_{ij} + 2a^{1+\alpha}\left(\frac{f_*}{m_{\mathsf{Pl}}}\right)^2\left(\tilde{\Pi}_{ij} - \frac{1}{3}\delta_{ij}\tilde{\Pi}_{kk}\right)$$

GWs in $\mathcal{C}$osmo$\mathcal{L}$attice: anisotropic tensor

src/include/CosmoInterface/definitions/PITensor.h

```
43    template<class Model> requires (Model::NDim == 3)
44    static inline auto effectiveAnisotropicTensor(Model &model)
45    {
46      return ConstructSymTraceless(effectiveAnisotropicTensor(model, 1_c, 1_c),
47                                   effectiveAnisotropicTensor(model, 1_c, 2_c),
48                                   effectiveAnisotropicTensor(model, 1_c, 3_c),
49                                   effectiveAnisotropicTensor(model, 2_c, 2_c),
50                                   effectiveAnisotropicTensor(model, 2_c, 3_c),
51                                   effectiveAnisotropicTensor(model, 3_c, 3_c));
52
53    }
```

```
56    template <class Model, int I, int J> static inline auto
      ↪  effectiveAnisotropicTensor(Model &model, Tag<I> i, Tag<J> j)
57    {
58      return scalarSingletContribution(model, i, j) +
        ↪  complexScalarContribution(model, i, j) + electricU1Contribution(model,
        ↪  i, j) +
59            magneticU1Contribution(model, i, j);
60    }
```

```
62    template <class Model, int I, int J> static inline auto
      ↪  scalarSingletContribution(Model &model, Tag<I> a, Tag<J> b)
63    {
64      return Total(i, 0, Model::Ns - 1, forwDiff(model.fldS(i), a) *
        ↪  forwDiff(model.fldS(i), b));
65    }
```

GWs in $\mathcal{C}$osmo$\mathcal{L}$attice: anisotropic tensor

src/include/CosmoInterface/definitions/PITensor.h

```
43    template<class Model> requires (Model::NDim == 3)
44    static inline auto effectiveAnisotropicTensor(Model &model)
45    {
46      return ConstructSymTraceless(effectiveAnisotropicTensor(model, 1_c, 1_c),
47                                   effectiveAnisotropicTensor(model, 1_c, 2_c),
48                                   effectiveAnisotropicTensor(model, 1_c, 3_c),
49                                   effectiveAnisotropicTensor(model, 2_c, 2_c),
50                                   effectiveAnisotropicTensor(model, 2_c, 3_c),
51                                   effectiveAnisotropicTensor(model, 3_c, 3_c));
52
53    }
```

```
56    template <class Model, int I, int J> static inline auto
      ↪  effectiveAnisotropicTensor(Model &model, Tag<I> i, Tag<J> j)
57    {
58      return scalarSingletContribution(model, i, j) +
        ↪  complexScalarContribution(model, i, j) + electricU1Contribution(model,
        ↪  i, j) +
59            magneticU1Contribution(model, i, j);
60    }
```

$$\Pi^{\phi}_{ij} = \sum_a \tilde{\nabla}^{+}_i \tilde{\phi} \tilde{\nabla}^{+}_i \tilde{\phi}$$

```
62    template <class Model, int I, int J> static inline auto
      ↪  scalarSingletContribution(Model &model, Tag<I> a, Tag<J> b)
63    {
64      return Total(i, 0, Model::Ns - 1, forwDiff(model.fldS(i), a) *
        ↪  forwDiff(model.fldS(i), b));
65    }
```

GWs in $\mathcal{C}$osmo$\mathcal{L}$attice: $h'_{ij}h'^{*}_{ij}$ wit complex projector

src/include/CosmoInterface/definitions/GWsProjector.h

```
49          template<class Model, class Looper, int I, int J, typename T = double>
50          inline auto Pr(Model& model, Looper& it, Tag<I> i, Tag<J> j) {
51
52              auto pVec = it.getVec();
53
54              size_t N = GetNGrid::get(model.getOneField());
55
56              // Note that we force the vector to have 3 componer
                ↪  the code compiles for any NDim. The GW module c
                ↪  NDim != 3, the code does not run and returns ar
57              auto klattice = MakeVector(l, 1, 3,
58              mType == 1 ? std::sin(2 * Constants::pi<T> * pVec[l-1] / N) :
59              mType == 2 ? std::complex<T>(1) - std::complex<T>(std::cos(2.0 *
                ↪  Constants::pi<T> / N * pVec[l-1]),std::sin(2.0 * Constants::pi<T> / N *
                ↪  pVec[l-1])) :
60              mType == 3 ? std::complex<T>(1) - std::complex<T>(std::cos(2.0 *
                ↪  Constants::pi<T> / N * pVec[l-1]),std::sin(-2.0 * Constants::pi<T> / N *
                ↪  pVec[l-1])) : std::complex<T>(1.));
61
62              T klatticeSquare = Total(k, 1, 3, abs((klattice(k))*conj(klattice(k))));
63
64              if(i == j) return (std::complex<T>)(1) - conj(klattice(i)) * (klattice(j)) /
                ↪  klatticeSquare;
65              return - conj(klattice(i)) * (klattice(j)) / klatticeSquare;
```

$$P_{ij} = \delta_{ij} - \frac{\tilde{k}_{\mathrm{L},i}\tilde{k}^{*}_{\mathrm{L},j}}{\tilde{k}^{2}_{\mathrm{L}}}$$

GWs in $\mathcal{C}$osmo$\mathcal{L}$attice: $h'_{ij}h'^{*}_{ij}$ wit complex projector

src/include/CosmoInterface/definitions/GWsProjector.h

```
76          template<class Model, class Looper>
77          auto projectedGW_complex(Model& model, Looper& it) {
78
79              auto P11 = Pr(model, it, 1_c, 1_c);
80              auto P12 = Pr(model, it, 1_c, 2_c);
81              auto P13 = Pr(model, it, 1_c, 3_c);
82              auto P21 = conj(P12);
83              auto P22 = Pr(model, it, 2_c, 2_c);
84              auto P23 = Pr(model, it, 2_c, 3_c);
85              auto P31 = conj(P13);
86              auto P32 = conj(P23);
87              auto P33 = Pr(model, it, 3_c, 3_c);
88
89              auto u11 = GetValue::get(model.pi_GWtensor(1_c,1_c).inFourierSpace(), it());
90              auto u12 = GetValue::get(model.pi_GWtensor(1_c,2_c).inFourierSpace(), it());
91              auto u13 = GetValue::get(model.pi_GWtensor(1_c,3_c).inFourierSpace(), it());
92              auto u21 = u12;
93              auto u22 = GetValue::get(model.pi_GWtensor(2_c,2_c).inFourierSpace(), it());
94              auto u23 = GetValue::get(model.pi_GWtensor(2_c,3_c).inFourierSpace(), it());
95              auto u31 = u13;
96              auto u32 = u23;
97              auto u33 = GetValue::get(model.pi_GWtensor(3_c,3_c).inFourierSpace(), it());
```

$$P_{ij} = \delta_{ij} - \frac{\tilde{\boldsymbol{k}}_{\mathrm{L},i}\tilde{\boldsymbol{k}}^{*}_{\mathrm{L},j}}{\tilde{\boldsymbol{k}}^{2}_{\mathrm{L}}}$$

$$(\pi_u)_{ij}(\tilde{\boldsymbol{k}}, \tilde{\eta}) = a^{3-\alpha} u'_{ij}$$

GWs in $\mathcal{C}$osmo$\mathcal{L}$attice: $h'_{ij}h'^{*}_{ij}$ wit complex projector

src/include/CosmoInterface/definitions/GWsProjector.h

```
99              auto Pu11 = P11 * u11 + P12 * u21 + P13 * u31;
100             auto Pu12 = P11 * u12 + P12 * u22 + P13 * u32;
101             auto Pu13 = P11 * u13 + P12 * u23 + P13 * u33;
102             auto Pu21 = P21 * u11 + P22 * u12 + P23 * u13;
103             auto Pu22 = P21 * u12 + P22 * u22 + P23 * u23;
104             auto Pu23 = P21 * u13 + P22 * u23 + P23 * u33;
105             auto Pu31 = P31 * u11 + P32 * u21 + P33 * u13;
106             auto Pu32 = P31 * u12 + P32 * u22 + P33 * u23;
107             auto Pu33 = P31 * u13 + P32 * u23 + P33 * u33;
108
109             auto Pu_s11 = P11 * conj(u11) + P12 * conj(u21) + P13 * conj(u31);
110             auto Pu_s12 = P11 * conj(u12) + P12 * conj(u22) + P13 * conj(u32);
111             auto Pu_s13 = P11 * conj(u13) + P12 * conj(u23) + P13 * conj(u33);
112             auto Pu_s21 = P21 * conj(u11) + P22 * conj(u12) + P23 * conj(u13);
113             auto Pu_s22 = P21 * conj(u12) + P22 * conj(u22) + P23 * conj(u23);
114             auto Pu_s23 = P21 * conj(u13) + P22 * conj(u23) + P23 * conj(u33);
115             auto Pu_s31 = P31 * conj(u11) + P32 * conj(u21) + P33 * conj(u13);
116             auto Pu_s32 = P31 * conj(u12) + P32 * conj(u22) + P33 * conj(u23);
117             auto Pu_s33 = P31 * conj(u13) + P32 * conj(u23) + P33 * conj(u33);
```

$a^{3-\alpha} Pu'$

$a^{3-\alpha} P(u')^{*}$

GWs in $\mathcal{C}$osmo$\mathcal{L}$attice: $h'_{ij}{h'_{ij}}^*$ wit complex projector

src/include/CosmoInterface/definitions/GWsProjector.h

```
119             auto Tr1 = (Pu11 * Pu_s11 + Pu12 * Pu_s21 + Pu13 * Pu_s31) +
                ↪  (Pu21 * Pu_s12 + Pu22 * Pu_s22 + Pu23 * Pu_s32) + (Pu31 *
                ↪  Pu_s13 + Pu32 * Pu_s23 + Pu33 * Pu_s33);
120             auto Tr2a = (Pu11 + Pu22 + Pu33);
121             auto Tr2b = (Pu_s11 + Pu_s22 + Pu_s33);
122
123             return pow(model.aI,2*(model.alpha-3)) * abs(Tr1 - .5 * Tr2a
                ↪  * Tr2b);
124         }
```

$$\boxed{h'_{ij}{h'_{ij}}^* = \mathrm{Tr}[Pu'P(u')^*] - \tfrac{1}{2}\mathrm{Tr}[Pu']\mathrm{Tr}[P(u')^*]}$$

GWs in $\mathcal{C}$osmo$\mathcal{L}$attice: $h'_{ij}h'^{*}_{ij}$ wit complex projector

src/include/CosmoInterface/definitions/GWsProjector.h

```
119         auto Tr1 = (Pu11 * Pu_s11 + Pu12 * Pu_s21 + Pu13 * Pu_s31) +
            ↪  (Pu21 * Pu_s12 + Pu22 * Pu_s22 + Pu23 * Pu_s32) + (Pu31 *
            ↪  Pu_s13 + Pu32 * Pu_s23 + Pu33 * Pu_s33);
120         auto Tr2a = (Pu11 + Pu22 + Pu33);
121         auto Tr2b = (Pu_s11 + Pu_s22 + Pu_s33);
122
123         return pow(model.aI,2*(model.alpha-3)) * abs(Tr1 - .5 * Tr2a
            ↪  * Tr2b);
124     }
```

$$h'_{ij}h'^{*}_{ij} = \mathrm{Tr}[Pu'P(u')^{*}] - \tfrac{1}{2}\mathrm{Tr}[Pu']\mathrm{Tr}[P(u')^{*}]$$

Similar for real projector

GWs in $\mathcal{C}$osmo$\mathcal{L}$attice: $h'_{ij}h'^{*}_{ij}$ wit complex projector

src/include/CosmoInterface/definitions/GWsProjector.h

```
119             auto Tr1 = (Pu11 * Pu_s11 + Pu12 * Pu_s21 + Pu13 * Pu_s31) +
                ↪  (Pu21 * Pu_s12 + Pu22 * Pu_s22 + Pu23 * Pu_s32) + (Pu31 *
                ↪  Pu_s13 + Pu32 * Pu_s23 + Pu33 * Pu_s33);
120             auto Tr2a = (Pu11 + Pu22 + Pu33);
121             auto Tr2b = (Pu_s11 + Pu_s22 + Pu_s33);
122
123             return pow(model.aI,2*(model.alpha-3)) * abs(Tr1 - .5 * Tr2a
                ↪  * Tr2b);
124         }
```

$$h'_{ij}h'^{*}_{ij} = \mathrm{Tr}[Pu'P(u')^{*}] - \frac{1}{2}\mathrm{Tr}[Pu']\mathrm{Tr}[P(u')^{*}]$$

GWs in $\mathcal{C}$osmo$\mathcal{L}$attice: $h'_{ij}h'^{*}_{ij}$ wit complex projector

src/include/CosmoInterface/definitions/GWsProjector.h

```
119             auto Tr1 = (Pu11 * Pu_s11 + Pu12 * Pu_s21 + Pu13 * Pu_s31) +
                ↪  (Pu21 * Pu_s12 + Pu22 * Pu_s22 + Pu23 * Pu_s32) + (Pu31 *
                ↪  Pu_s13 + Pu32 * Pu_s23 + Pu33 * Pu_s33);
120             auto Tr2a = (Pu11 + Pu22 + Pu33);
121             auto Tr2b = (Pu_s11 + Pu_s22 + Pu_s33);
122
123             return pow(model.aI,2*(model.alpha-3)) * abs(Tr1 - .5 * Tr2a
                ↪  * Tr2b);
124         }
```

$$h'_{ij}h'^{*}_{ij} = \mathrm{Tr}[Pu'P(u')^{*}] - \frac{1}{2}\mathrm{Tr}[Pu']\mathrm{Tr}[P(u')^{*}]$$

GWs in $\mathcal{C}$osmo$\mathcal{L}$attice: GW power spectrum

```
55          if (PSVersion != 3){
56              auto fk2 = projectRadiallyGW(model, PSVersion == 1, PRJType, PSType,
                ↪  PSVersion).measure(model, nbins, kMaxBins);
57              if (PSType == 2){
58                  return Function(ntilde, pow<3>(kIR * ntilde * dx ) / N3
                    ↪  /(8*pow<2>(Constants::pi<T>)*pow(model.aI,2*model.alpha))
                    ↪  *pow<2>(model.fStar / Constants::reducedMPlanck<T>)/rho) * fk2;
59              }
```

$$\Omega_{\rm GW} = \frac{1}{\tilde{\rho}_{\rm tot}} \frac{\tilde{k}^3}{8\pi^2 a^{2\alpha}} \left(\frac{\delta\tilde{x}}{N}\right)^3 \left(\frac{m_{\rm Pl}}{f_*}\right)^2 \langle h'_{ij} h'^{*}_{ij} \rangle_{R(\tilde{k})}$$

GWs in $\mathcal{C}$osmo$\mathcal{L}$attice: GW power spectrum

```
55          if (PSVersion != 3){
56              auto fk2 = projectRadiallyGW(model, PSVersion == 1, PRJType, PSType,
                ↪  PSVersion).measure(model, nbins, kMaxBins);
57              if (PSType == 2){
58                  return Function(ntilde, pow<3>(kIR * ntilde * dx ) / N3
                    ↪  /(8*pow<2>(Constants::pi<T>)*pow(model.aI,2*model.alpha))
                    ↪  *pow<2>(model.fStar / Constants::reducedMPlanck<T>)/rho) * fk2;
59              }
```

$$\Omega_{\rm GW} = \frac{1}{\tilde{\rho}_{\rm tot}} \frac{\tilde{k}^3}{8\pi^2 a^{2\alpha}} \left(\frac{\delta\tilde{x}}{N}\right)^3 \left(\frac{m_{\rm Pl}}{f_*}\right)^2 \langle h'_{ij} {h'_{ij}}^* \rangle_{R(\tilde{k})}$$

```
60              else if (PSType == 1){
61                  fk2.sumInsteadOfAverage();
62                  return Function(ntilde, kIR * ntilde * dx  / pow<5>(N) /
                    ↪  (8*(Constants::pi<T>)*pow(model.aI,2*model.alpha))
                    ↪  *pow<2>(model.fStar/Constants::reducedMPlanck<T>)/rho) * fk2;
63              }
```

$$\Omega_{\rm GW} = \frac{1}{\tilde{\rho}_{\rm tot}} \frac{\tilde{k}}{8\pi^2 a^{2\alpha}} \left(\frac{\delta\tilde{x}}{N^5}\right) \left(\frac{m_{\rm Pl}}{f_*}\right)^2 \#_{\tilde{k}} \langle h'_{ij} {h'_{ij}}^* \rangle_{R(\tilde{k})}$$

Overview

1. Numerical implementation in $\mathcal{C}$osmo$\mathcal{L}$attice
2. Diving into the GWs module
3. Running simulations with GWs
4. Physical applications

Example 1: ϕ^4 model

Example 1:

$$V(\phi) = \lambda\phi^4$$

```
#Lattice
N = 32
dt = 0.01
kIR = 0.2

#Times
tOutputFreq = 1
tOutputInfreq = 5
tMax = 1000

#Spectra options
PS_type = 1
PS_version = 1
```

```
#GWs
GWprojectorType = 1
withGWs = true

#IC
kCutOff = 4
initial_amplitudes = 5.6964e18
initial_momenta = -4.86735e30

#Model Parameters
lambda = 9e-14
```

Example 1: ϕ^4 model

Example 1:

$$V(\phi) = \lambda\phi^4$$

```
#Lattice
N = 32
dt = 0.01
kIR = 0.2

#Times
tOutputFreq = 1
tOutputInfreq = 5
tMax = 1000

#Spectra options
PS_type = 1
PS_version = 1
```

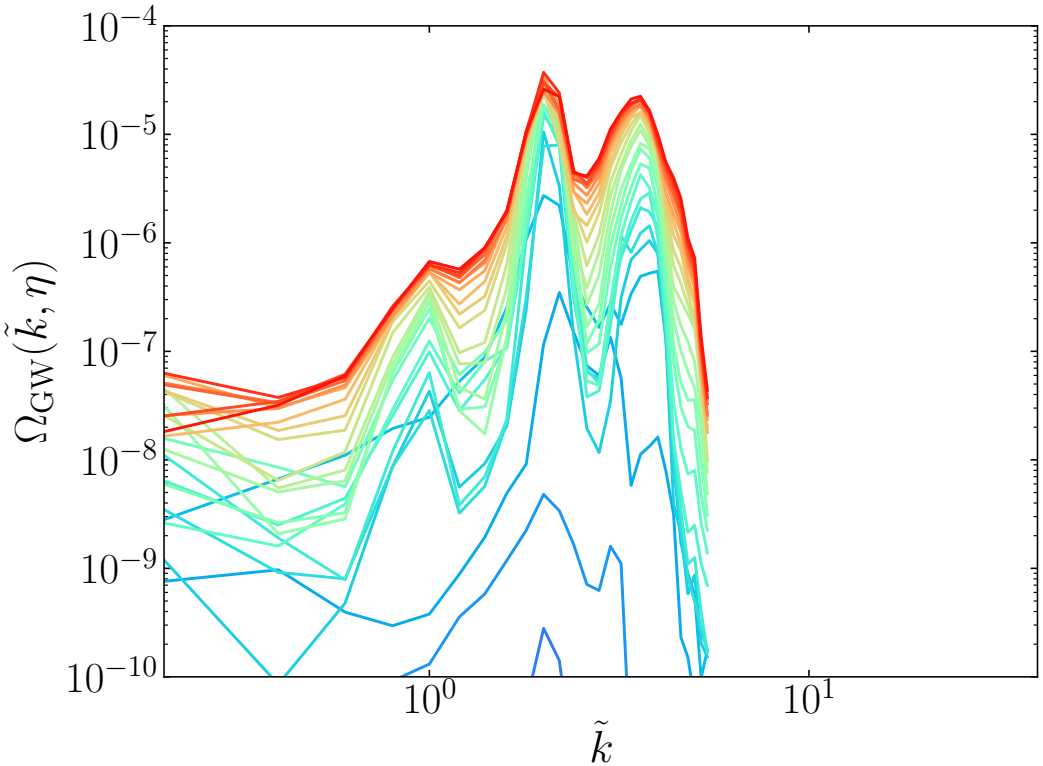
```
#GWs
GWprojectorType = 1
withGWs = true

#IC
kCutOff = 4
initial_amplitudes = 5.6964e18
initial_momenta = -4.86735e30

#Model Parameters
lambda = 9e-14
```

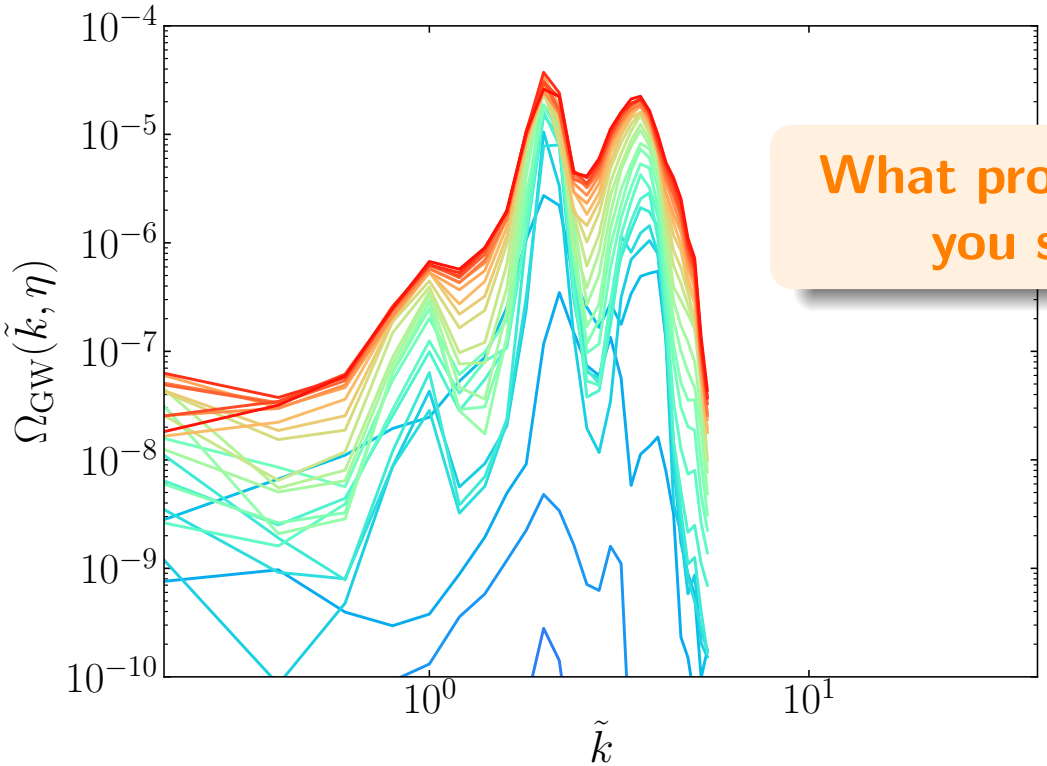
Example 1: ϕ^4 model

Run the model with $N = 32$. You should get something like this:



Example 1: ϕ^4 model

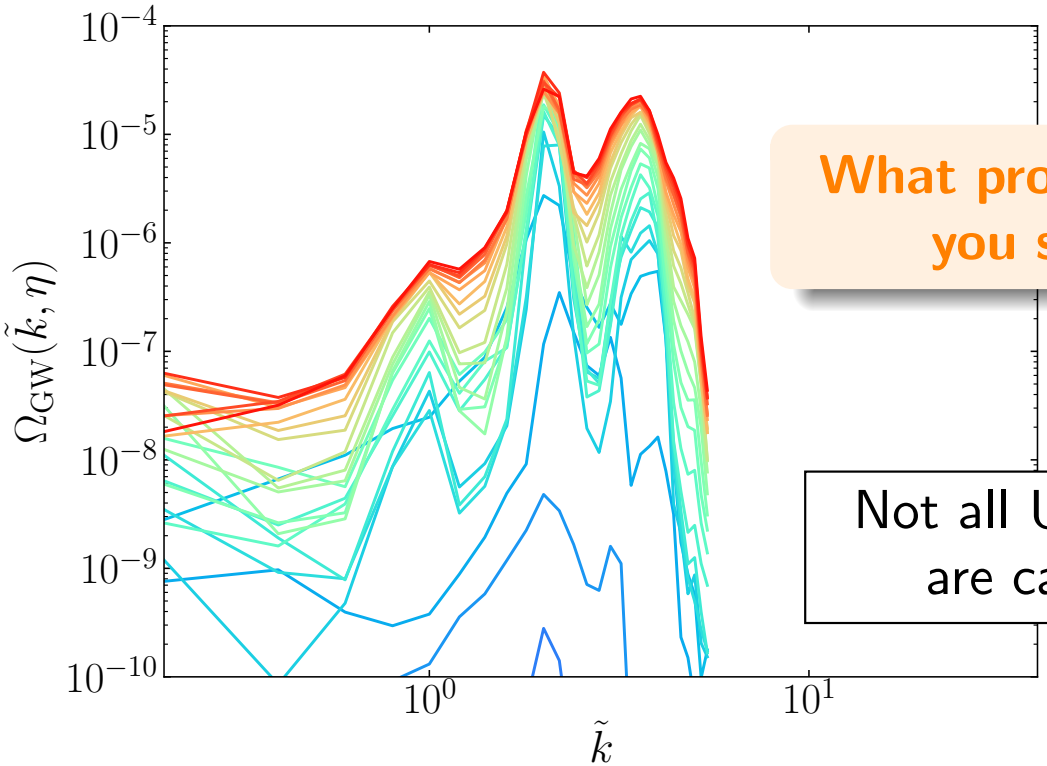
Run the model with $N = 32$. You should get something like this:



What problem do you see?

Example 1: ϕ^4 model

Run the model with $N = 32$. You should get something like this:



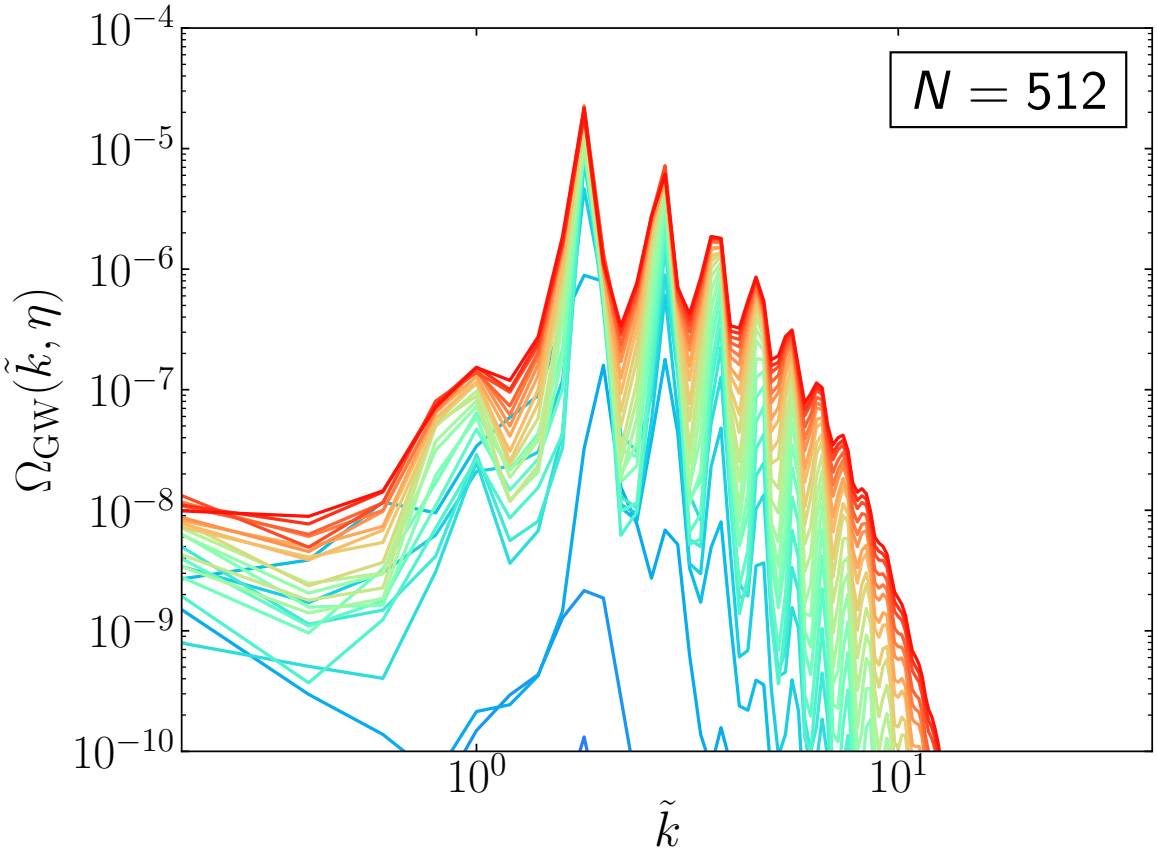
What problem do you see?

Not all UV physics are captured

➤ With $N = 32$ **we cannot capture the UV physics and keep IR resolution!**

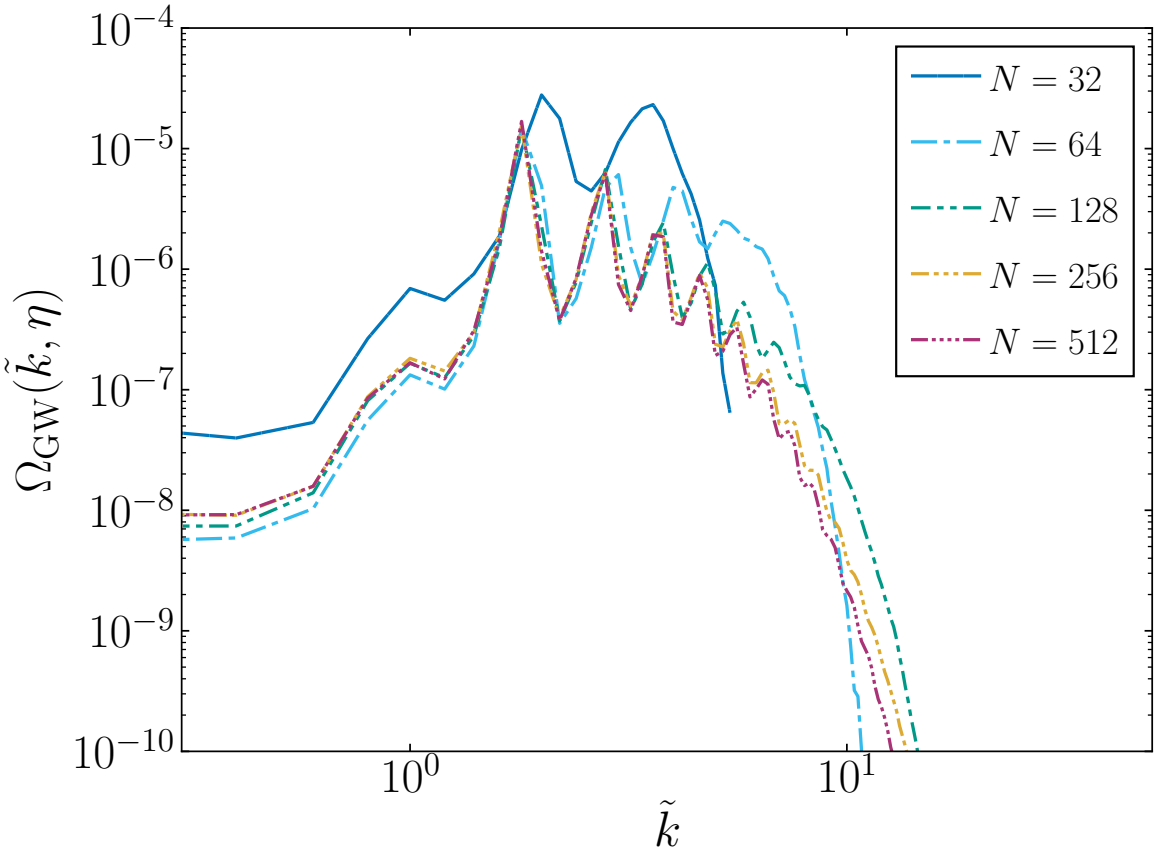
Example 1: ϕ^4 model

To capture capture all physics, we need to go to larger lattices:



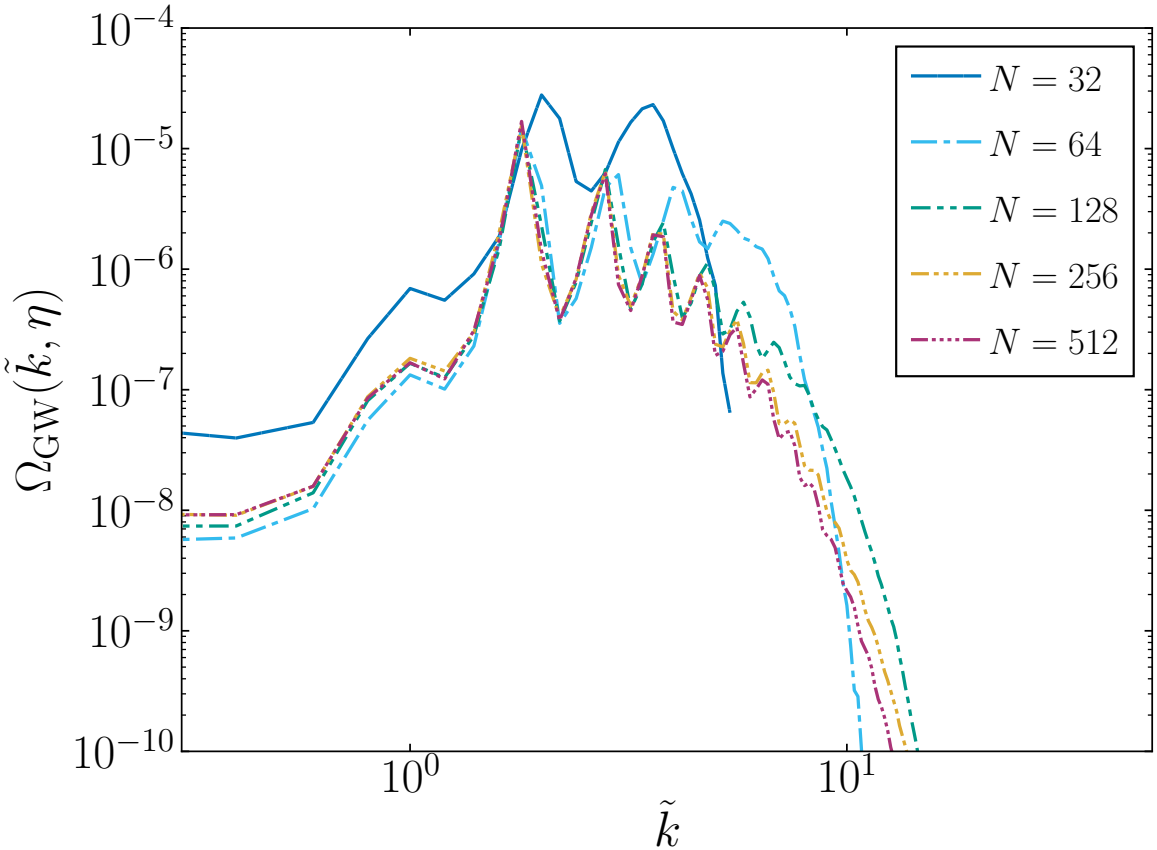
Example 1: ϕ^4 model

To capture capture all physics, we need to go to larger lattices:



Example 1: ϕ^4 model

To capture capture all physics, we need to go to larger lattices:



Large discretization/volume effects if UV/IR physics are not properly captured by small lattices!

Example 2: ϕ^4 model with daughter field

Example 2:

$$V(\phi) = \lambda\phi^4 + q\phi^2\chi^2$$

```
#Lattice
N = 512
dt = 0.01
kIR = 0.2

#Times
tOutputFreq = 1
tOutputInfreq = 5
tMax = 1000

#Spectra options
PS_type = 1
PS_version = 1
```

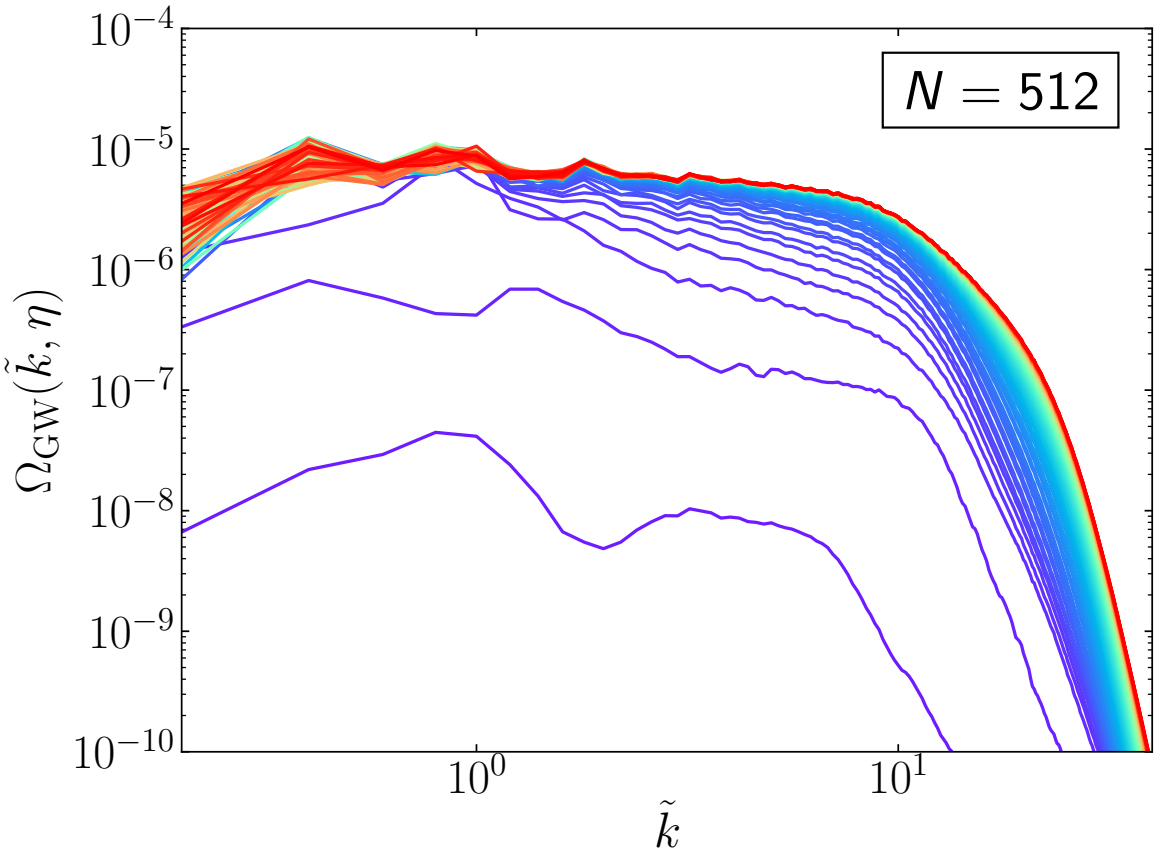
```
#GWs
GWprojectorType = 1
withGWs = true

#IC
kCutOff = 4
initial_amplitudes = 5.6964e18 0
initial_momenta = -4.86735e30 0

#Model Parameters
lambda = 9e-14
q = 100
```

Example 2: ϕ^4 model with daughter field

Interactions to daughter field destroy the self-resonance peaks



Example 3: ϕ^4 model with U(1) fields

Example 3:

$$V(\varphi) = \lambda|\varphi|^4 + \text{U(1) field}$$

```
#Lattice
N = 512
dt = 0.01
kIR = .3

#Times
tOutputFreq = 1
tOutputInfreq = 5
tOutputRareFreq = 10
tMax = 1000

#GWs
withGWs = true
```

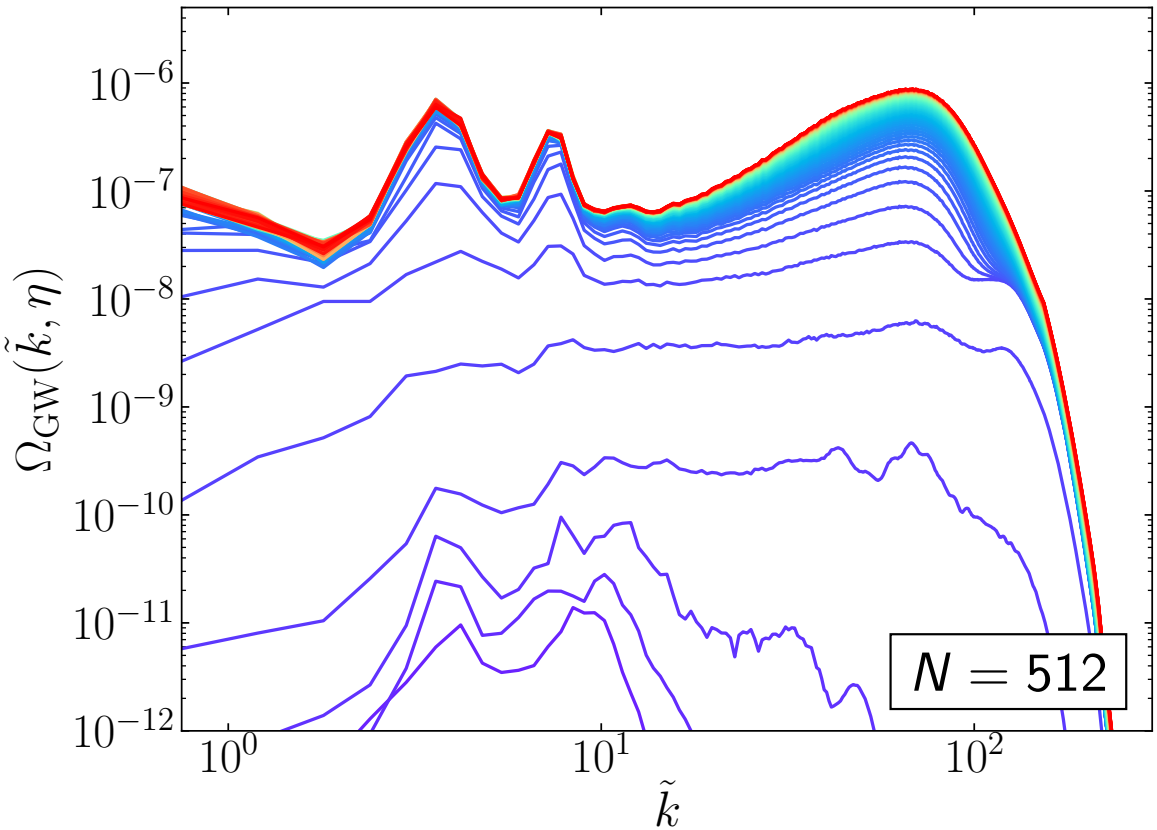
```
#IC
kCutOff = 3
initial_amplitudes = 0
initial_momenta = 0
cmplx_field_initial_norm = 5.6964e18
cmplx_momentum_initial_norm =
↪  -6.88343e30

#Gauge couplings and effectiveCharges
CSU1Charges = 1
gU1s = 1.11037e-05

#Model Parameters
lambda = 9e-14
```

Example 3: ϕ^4 model with U(1) fields

Interactions with U(1) fields leads to different spectrum (2 peaks + buldge)



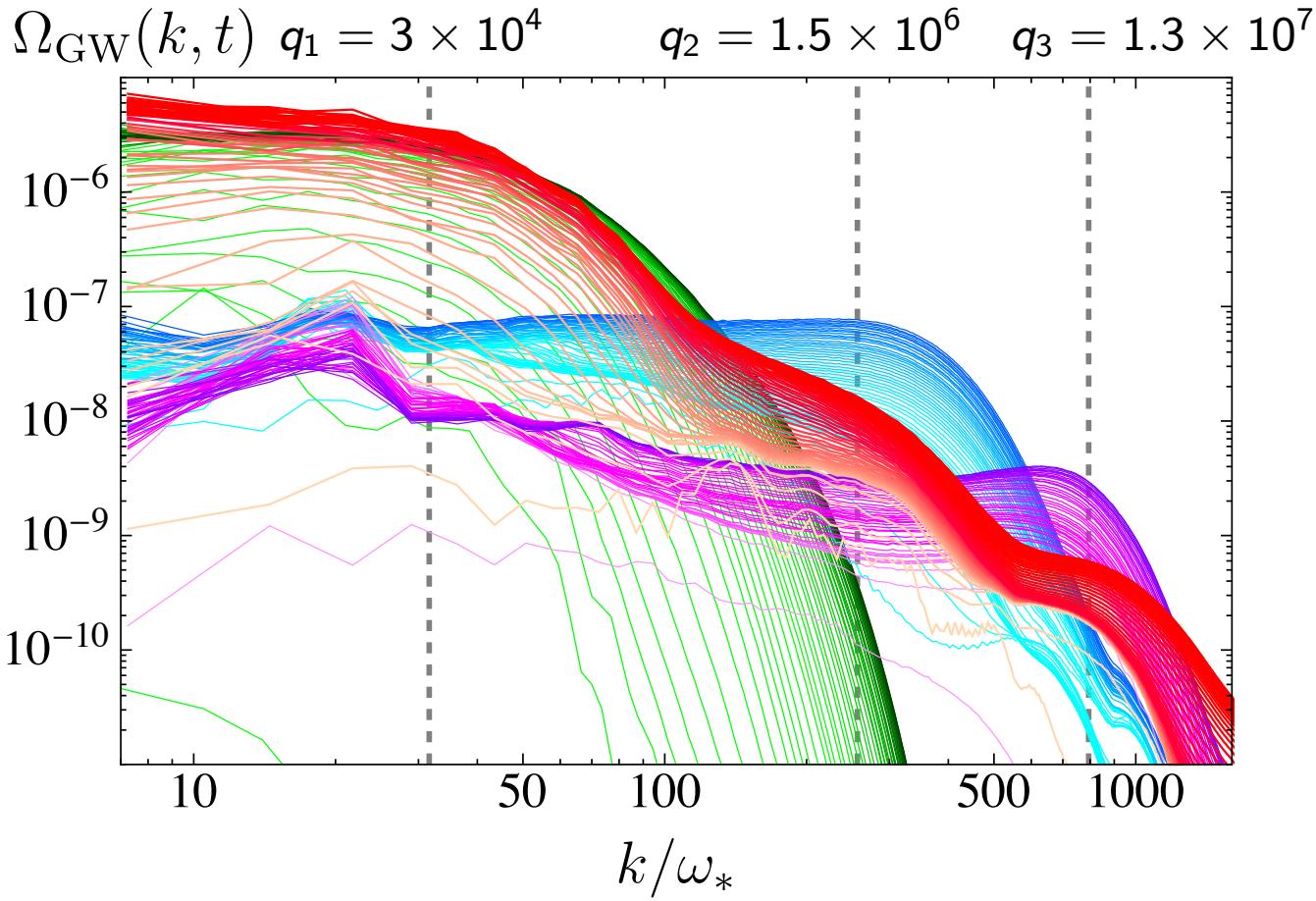
- Spectral shape tells us about underlying particle physics
- **Note**: Simulations with gauge fields + GWs are computationally expensive

Overview

1. Numerical implementation in $\mathcal{C}$osmo$\mathcal{L}$attice
2. Diving into the GWs module
3. Running simulations with GWs
4. Physical applications

Spectroscopy of particle couplings with GWs

[D. G. Figueroa, A. Florio, N. Loayza and M. Pieroni, 2022]



$$V(\phi,\chi_i) = \frac{\Lambda^4}{2}\tanh^2\left(\frac{\phi}{M}\right) + \sum_a \frac{g_a^2}{2}\phi^2\chi_a^2 \qquad q_a = g_a^2\frac{f_*^2}{\omega_*^2}$$

Other applications

The **GW module is $\mathcal{C}$osmo$\mathcal{L}$attice** is being used for **many other applications**:

- GWs from **fluid turbulences**
- GWs from **cosmic topological defects**
  - GWs from cosmic strings
  - GWs from biased domain walls
- GWs from axion inflation

Any questions?