# Practice 4: Simulating GW emission from scalar and gauge fields

**Jorge Baeza-Ballesteros and Nicolás Loayza**

$\mathcal{C}$osmo$\mathcal{L}$attice school 2025 - 24th September 2025

Deutsches Elektronen
-Synchroton Zeuthen

IFIC
INSTITUT DE FÍSICA CORPUSCULAR

# 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 six 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 six 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

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

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

- Simulate the evolution of six 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

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

Simulations are performed **analogously to scalar fields**

# Measures of GW observables

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

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

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

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

where $\tilde{\rho}_\mathrm{tot}$ 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**

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

# 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

```
#Times
tOutputFreq = 0.1
tOutputInfreq = 1
tMax = 300


#Spectra options
PS_type = 1
PS_version = 1

#GWs
GWprojectorType = 1
withGWs = false

#IC
kCutOff = 4
initial_amplitudes = 5.6964e18 0  # homogeneous amplitudes in GeV
```

# GWs in CosmoLattice: input parameters

`src/model/parameter-files/lphi4.in`

```
#Times
tOutputFreq = 0.1
tOutputInfreq = 1
tMax = 300


#Spectra options
PS_type = 1
PS_version = 1

#GWs
GWprojectorType = 1
withGWs = false

#IC
kCutOff = 4
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`

$$\Omega_{\rm GW} \rightarrow \frac{1}{\tilde{\rho}_{\rm tot}}\frac{d\tilde{\rho}_{\rm GW}}{d\log k}$$

| $\kappa$ | $\Omega_{\rm GW}$ | Multiplicity |
|---|---|---|
| 0.4 | 3.455976932183128e-29 | 13 |
| 0.8 | 1.135499488356703e-28 | 37 |
| 1.2 | 1.653950018181714e-28 | 57 |
| $\vdots$ | $\vdots$ | $\vdots$ |

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 | 1.26699487198562e-27 | 5.29787700163311e-29 |
| 2 | 2.75524033399837e-27 | 1.24679240886064e-29 |
| $\vdots$ | $\vdots$ | $\vdots$ |

# GWs in CosmoLattice: equation of motion

src/include/CosmoInterface/evolvers/leapfrog.h

```
        if (model.fldGWs != nullptr) kickGWs(model, weight);
```

```
    template<class Model>
    void kickGWs(Model& model, T w) {
        ForLoop(n, 0, Model::NGWs - 1,
                (*model.piGWs)(n) += (w * model.dt) * GWsKernels::get(model,n);
        );
    }
```

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

```
        if (model.fldGWs != nullptr) driftGWs(model);
```

```
    template<class Model>
    void driftGWs(Model& model) {

        (*model.fldGWs) +=  pow(model.aSI, model.alpha - 3) * (model.dt * (*model.piGWs) );

    }
```

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

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

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

```cpp
        template <class Model, int N>
        static auto get(Model& model, Tag<N> n){

            return (pow(model.aI, 1 + model.alpha) *
                    LatLapl<Model::NDim>( (*model.fldGWs)(n))
                     + pow(model.aI, 1 + model.alpha) * 2. * (PITensor::totalTensor(model,n)));
        }
```

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

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

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

```
        template <class Model, int N>
        static auto get(Model& model, Tag<N> n){

            return (pow(model.aI, 1 + model.alpha) *
                   LatLapl<Model::NDim>( (*model.fldGWs)(n))
                    + pow(model.aI, 1 + model.alpha) * 2. *
                    ↪  (PITensor::totalTensor(model,n)));
        }
```

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

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

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

```
        template <class Model, int N>
        static auto get(Model& model, Tag<N> n){

            return (pow(model.aI, 1 + model.alpha) *
                LatLapl<Model::NDim>( (*model.fldGWs)(n))
                 + pow(model.aI, 1 + model.alpha) * 2. *
                 ↪  (PITensor::totalTensor(model,n)));
        }
```

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

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

src/include/CosmoInterface/definitions/PITensor.h

```
template<class Model>
static inline auto totalTensor(Model& model, Tag<0>)
{
    return totalTensor(model, 1_c, 1_c);
}
```

```
template<class Model, int I, int J>
static inline auto totalTensor(Model& model, Tag<I> i, Tag<J> j)
{
    return scalarSinglet(model, i, j) + complexScalar(model, i, j) + electricU1(model, i, j) + magneticU1(model, i, j);
}
```

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

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

`src/include/CosmoInterface/definitions/PITensor.h`

```cpp
        template<class Model>
        static inline auto totalTensor(Model& model, Tag<0>)
        {
            return totalTensor(model, 1_c, 1_c);
        }
```

```cpp
        template<class Model, int I, int J>
        static inline auto totalTensor(Model& model, Tag<I> i, Tag<J> j)
        {
            return scalarSinglet(model, i, j) + complexScalar(model, i, j) + electricU1(model, i, j) + magneticU1(model, i, j);
        }
```

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

$$\boxed{\Pi_{ij}^{\phi} = \sum_a \tilde{\nabla}_i^+ \tilde{\phi} \tilde{\nabla}_i^+ \tilde{\phi}}$$

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

src/include/CosmoInterface/definitions/GWsProjector.h

```
template<class Model, class Looper, int I, int J, typename T = double>
inline auto Pr(Model& model, Looper& it, Tag<I> i, Tag<J> j) {

    auto pVec = it.getVec();

    size_t N = GetNGrid::get(model.getOneField());

    // Note that we force the vector to have 3 compone
    ↪  so the code compiles for any NDim. The GW modu
    ↪  If NDim != 3, the code does not run and returns an error.
    auto klattice = MakeVector(l, 1, 3,
    mType == 1 ? std::sin(2 * Constants::pi<T> * pVec[l-1] / N) :
    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])) :
    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.));

    T klatticeSquare = Total(k, 1, 3, abs((klattice(k))*conj(klattice(k))));

    if(i == j) return (std::complex<T>)(1) - conj(klattice(i)) * (klattice(j)) /
    ↪  klatticeSquare;
    return - conj(klattice(i)) * (klattice(j)) / klatticeSquare;
```

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

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

src/include/CosmoInterface/definitions/GWsProjector.h

```
        template<class Model, class Looper>
        auto projectedGW_complex(Model& model, Looper& it) {

            auto P11 = Pr(model, it, 1_c, 1_c);
            auto P12 = Pr(model, it, 1_c, 2_c);
            auto P13 = Pr(model, it, 1_c, 3_c);
            auto P21 = conj(P12);
            auto P22 = Pr(model, it, 2_c, 2_c);
            auto P23 = Pr(model, it, 2_c, 3_c);
            auto P31 = conj(P13);
            auto P32 = conj(P23);
            auto P33 = Pr(model, it, 3_c, 3_c);

            auto u11 = GetValue::get(model.pi_GWtensor(1_c,1_c).inFourierSpace(), it());
            auto u12 = GetValue::get(model.pi_GWtensor(1_c,2_c).inFourierSpace(), it());
            auto u13 = GetValue::get(model.pi_GWtensor(1_c,3_c).inFourierSpace(), it());
            auto u21 = u12;
            auto u22 = GetValue::get(model.pi_GWtensor(2_c,2_c).inFourierSpace(), it());
            auto u23 = GetValue::get(model.pi_GWtensor(2_c,3_c).inFourierSpace(), it());
            auto u31 = u13;
            auto u32 = u23;
            auto u33 = GetValue::get(model.pi_GWtensor(3_c,3_c).inFourierSpace(), it());
```

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

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

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

src/include/CosmoInterface/definitions/GWsProjector.h

```
            auto Pu11 = P11 * u11 + P12 * u21 + P13 * u31;
            auto Pu12 = P11 * u12 + P12 * u22 + P13 * u32;
            auto Pu13 = P11 * u13 + P12 * u23 + P13 * u33;
            auto Pu21 = P21 * u11 + P22 * u12 + P23 * u13;
            auto Pu22 = P21 * u12 + P22 * u22 + P23 * u23;
            auto Pu23 = P21 * u13 + P22 * u23 + P23 * u33;
            auto Pu31 = P31 * u11 + P32 * u21 + P33 * u13;
            auto Pu32 = P31 * u12 + P32 * u22 + P33 * u23;
            auto Pu33 = P31 * u13 + P32 * u23 + P33 * u33;

            auto Pu_s11 = P11 * conj(u11) + P12 * conj(u21) + P13 * conj(u31);
            auto Pu_s12 = P11 * conj(u12) + P12 * conj(u22) + P13 * conj(u32);
            auto Pu_s13 = P11 * conj(u13) + P12 * conj(u23) + P13 * conj(u33);
            auto Pu_s21 = P21 * conj(u11) + P22 * conj(u12) + P23 * conj(u13);
            auto Pu_s22 = P21 * conj(u12) + P22 * conj(u22) + P23 * conj(u23);
            auto Pu_s23 = P21 * conj(u13) + P22 * conj(u23) + P23 * conj(u33);
            auto Pu_s31 = P31 * conj(u11) + P32 * conj(u21) + P33 * conj(u13);
            auto Pu_s32 = P31 * conj(u12) + P32 * conj(u22) + P33 * conj(u23);
            auto Pu_s33 = P31 * conj(u13) + P32 * conj(u23) + P33 * conj(u33);
```

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

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

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

`src/include/CosmoInterface/definitions/GWsProjector.h`

```
            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);
            auto Tr2a = (Pu11 + Pu22 + Pu33);
            auto Tr2b = (Pu_s11 + Pu_s22 + Pu_s33);

            return pow(model.aI,2*(model.alpha-3)) * abs(Tr1 - .5 *
            ↪  Tr2a * Tr2b);
        }
```

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

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

`src/include/CosmoInterface/definitions/GWsProjector.h`

```
            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);
            auto Tr2a = (Pu11 + Pu22 + Pu33);
            auto Tr2b = (Pu_s11 + Pu_s22 + Pu_s33);

            return pow(model.aI,2*(model.alpha-3)) * abs(Tr1 - .5 * Tr2a * Tr2b);
        }
```

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

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`

```
        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);
        auto Tr2a = (Pu11 + Pu22 + Pu33);
        auto Tr2b = (Pu_s11 + Pu_s22 + Pu_s33);

        return pow(model.aI,2*(model.alpha-3)) * abs(Tr1 - .5 *
        ↪  Tr2a * Tr2b);
    }
```

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

# GWs in CosmoLattice: $h'_{ij}h'^{*}_{ij}$ wit complex projector

src/include/CosmoInterface/definitions/GWsProjector.h

```
            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);
            auto Tr2a = (Pu11 + Pu22 + Pu33);
            auto Tr2b = (Pu_s11 + Pu_s22 + Pu_s33);

            return pow(model.aI,2*(model.alpha-3)) * abs(Tr1 - .5 * Tr2a * Tr2b);
        }
```

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

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

`src/include/CosmoInterface/measurements/gwspowerspectrum.h`

```
            if (PSVersion != 3){
                auto fk2 = projectRadiallyGW(model, PSVersion == 1, PRJType, PSType, PSVersion).measure(model, nbins, kMaxBins);
                if (PSType == 2){
                    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;
                }
```

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

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

`src/include/CosmoInterface/measurements/gwspowerspectrum.h`

```
            if (PSVersion != 3){
                auto fk2 = projectRadiallyGW(model, PSVersion == 1, PRJType, PSType, PSVersion).measure(model, nbins, kMaxBins);
                if (PSType == 2){
                    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;
                }
```

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

```
                else if (PSType == 1){
                    fk2.sumInsteadOfAverage();
                    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;
                }
```

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

# 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: $\phi^4$ 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: $\phi^4$ 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
```

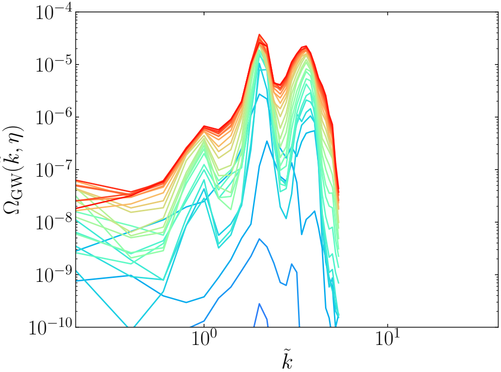
```
#GWs
GWprojectorType = 1
withGWs = true

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

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

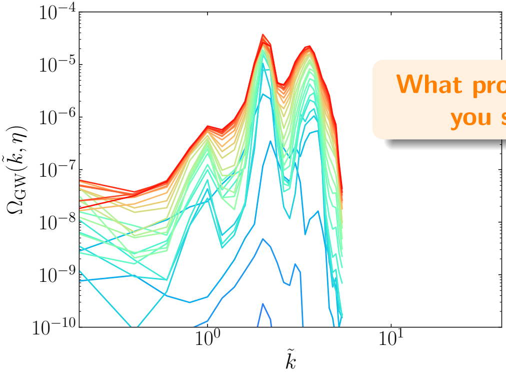
# Example 1: $\phi^4$ model

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



# Example 1: $\phi^4$ model

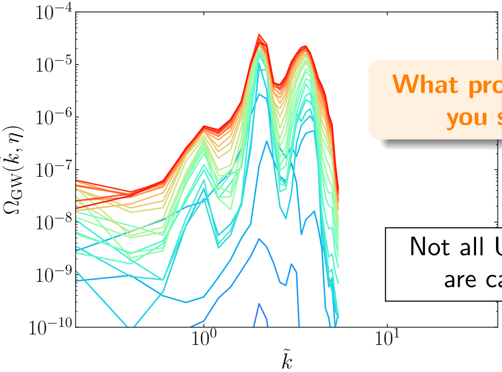
**Run the model with $N = 32$**. You should get something like this:




**What problem do you see?**

# Example 1: $\phi^4$ model

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



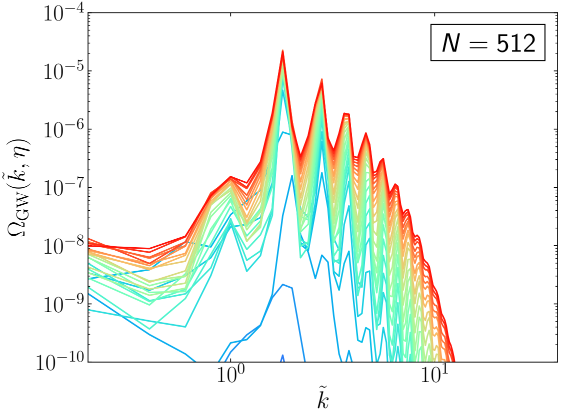

**What problem do you see?**

Not all UV physics are captured

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

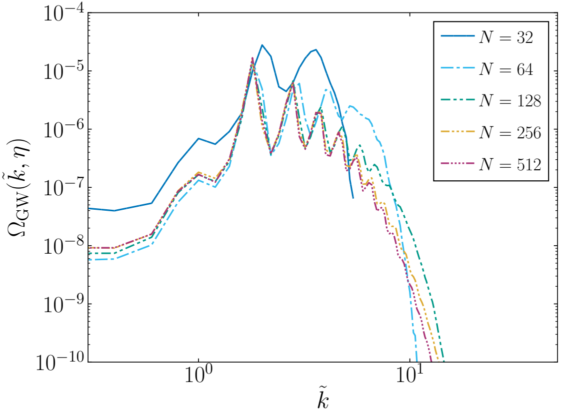
# Example 1: $\phi^4$ model

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



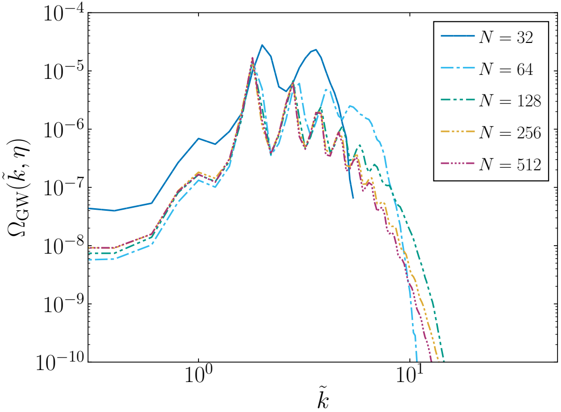
# Example 1: $\phi^4$ model

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



# Example 1: $\phi^4$ 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: $\phi^4$ 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
```

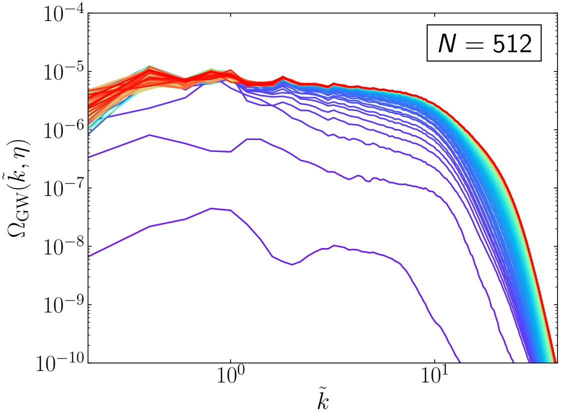
```
#GWs
GWprojectorType = 1
withGWs = true

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

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

# Example 2: $\phi^4$ model with daughter field

Interactions to daughter field destroy the self-resonance peaks



# Example 3: $\phi^4$ model with U(1) fields

**Example 3**:

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

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

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

#GWs
withGWs = true
```

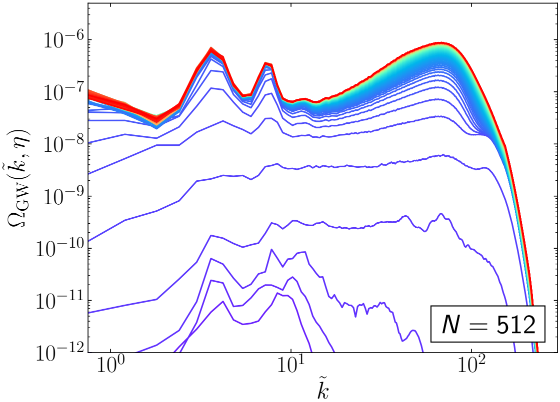
```
#IC
kCutOff = 3
initial_amplitudes = 0
initial_momenta = 0
cmplx_field_initial_norm = 5.6964e18
cmplx_momentum_initial_norm =
↪  -6.88343e30

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

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

# Example 3: $\phi^4$ 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

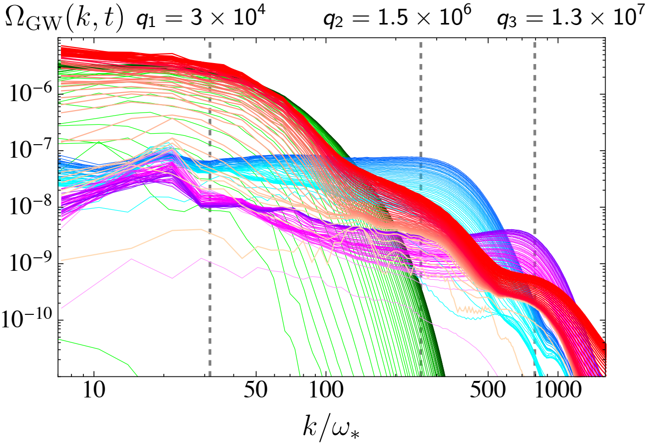
2. Diving into the GWs module

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** (Tomorrow, Practice 5)
- GWs from **cosmic topological defects** (**Friday**, Lecture 11)
  - GWs from cosmic strings
  - GWs from biased domain walls
- GWs from axion inflation (This morning!)

**Any questions?**