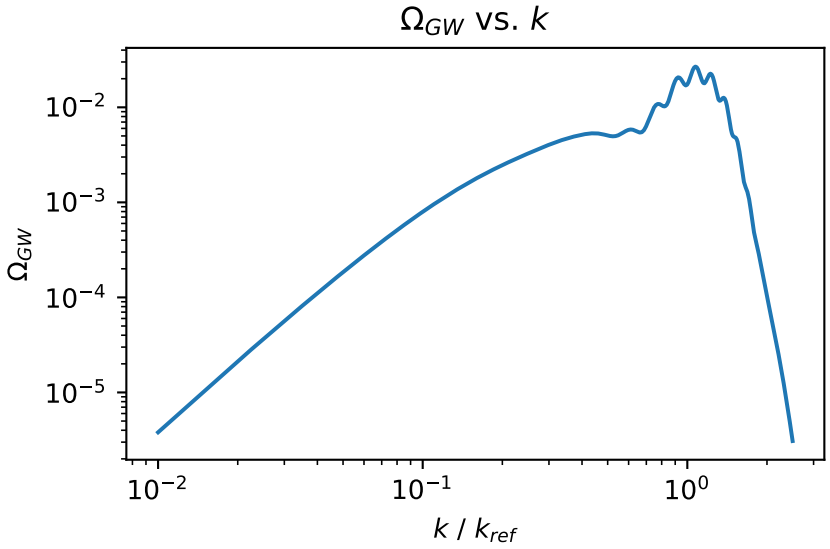
SIGWfast Scientific User Guide

Lukas T. Witkowski

August 2022

Abstract

SIGWfast is a python code to compute the **S**calar-**I**nduced **G**ravitational **W**ave spectrum from a primordial scalar power spectrum that can be given in analytical or numerical form. SIGWfast was written with the aim of being easy to install and use, and to produce results fast, typically in a matter of a few seconds. To this end the code employs vectorization techniques within python, but there is also the option to compile a C++ module to perform the relevant integrations, further accelerating the computation. The python-only version should run on all platforms that support python3. The version employing the C++ module is only available for Linux and MacOS systems.



Contents

1 Introduction

The nascent era of gravitational wave astronomy offers new ways of testing cosmic inflation through its signature in the stochastic gravitational wave background [1, 2]. An important observable in this context is the spectrum of scalar-induced gravitational waves (SIGW), see [3] for a recent review, which is produced when scalar fluctuations that exited the horizon during inflation re-enter in the post-inflationary era.

For the SIGW signal to be potentially detectable by the upcoming generation of gravitational wave observatories, the sourcing scalar fluctuations need to be sufficiently large. At large scales ($\gtrsim$ 1Mpc) the amplitude of scalar fluctuations is bounded by Cosmic Microwave Background (CMB) and Large Scale Structure (LSS) data, and the corresponding SIGW signal will be undetectable. However, these bounds do not apply at small scales ($\ll$ 1Mpc), allowing for a significant enhancement of scalar fluctuations, rendering the SIGW signal potentially observable. As a result, the SIGW spectrum is particularly suited for testing inflation at small scales, thus complementing CMB and LSS measurements, which give information about large scales.

Inflation models that enhance scalar fluctuations have been studied extensively in recent years. One reason is the increased interest in production mechanisms for primordial black holes (see [4–6] for overviews of the topic), which are generated as overdensities sourced by large scalar fluctuations collapse. In addition, such models have also been studied in their own right as completions of inflation at small scales, where constraints from CMB and LSS data are not available, and inflation can depart from the single-field slow-roll paradigm favoured at large scales. In all these models the spectrum of SIGW is an important observable, whose detection could give important information about the mechanism of inflation.[1]

As a result, the computation of the spectrum $\Omega_{\rm GW}(k)$ of the SIGW is in the process of becoming a routine task for inflationary model builders with an interest in mechanisms beyond single-field

[1] To give an example, a characteristic signature of a departure of inflation from the single-field slow-roll paradigm is an oscillatory modulation of the SIGW spectrum, whose properties can be directly related to inflationary-era quantities.

slow-roll models. The leading contribution to the SIGW spectrum is computed as a convolution-like double integral over two factors of the scalar power spectrum $\mathcal{P}_\zeta(k)$ and a transfer function $\mathcal{T}$, i.e. $\Omega_{\rm GW} \sim \iint \mathcal{T}\mathcal{P}_\zeta\mathcal{P}_\zeta$ [7–10]. Even though the transfer functions are known explicitly for various post-inflationary histories [9–12], the integration typically has to be performed numerically, and analytic results can only be derived for idealised choices of $\mathcal{P}_\zeta$ (like δ-distributions [10, 13] or lognormal peaks [14]), or in certain limits (e.g. in the IR limit $k \to 0$). Thus every researcher on this topic is eventually confronted with the task of computing this integration numerically. While performing a double integral numerically is not a priori difficult, there a couple of subtleties to be considered in this case. Firstly, the transfer function $\mathcal{T}$ typically exhibits a divergence, and the dominant contribution to the integral comes from integrating over the vicinity of the divergent locus. Hence all-purpose numerical integration routines like `NIntegrate` of Wolfram Mathematica or `scipy.integrate.quad` in python have to be used with care to treat the singular locus correctly. In the worst case, the computation produces an error or, even worse, wrong results. Secondly, one of the integrations is unbounded above, so that an upper limit has to be set in practice. This has to be chosen correctly to not affect the result. Finally, as the double integral has to be evaluated for every value of k, an unoptimized code can run significantly longer (by a factor of 100 or more!) than a version that has been designed for high speed.

These considerations are strong motivations for a public code to compute the SIGW spectrum that is *(i)* easy to use, *(ii)* robust against possible pitfalls, and *(iii)* computes $\Omega_{\rm GW}(k)$ fast. The package 'SIGWfast' has been written with these three goals in mind. To be easy to use, SIGWfast has a python interface where the computation can be configured and through which the input scalar power spectrum $\mathcal{P}_\zeta(k)$ is provided. It can also be run on all systems that have a functioning installation of python3. For robustness, SIGWfast sets the integration cutoff automatically. It also makes very little demands on how the scalar power spectrum is to be provided.[2] It also outputs plots which allow the user to check that the computation proceeded correctly. Regarding speed, SIGWfast comes as a python-only version, which uses vectorization and list comprehension to minimise computation time. It also provides the option to use a compiled C++ module for performing the integration, which reduces the computation time by another 20%-25%. This option is however restricted to systems running on Linux or MacOS. Overall, on a modern laptop or desktop machine a computation of $\Omega_{\rm GW}(k)$ with a quality suitable for publication typically takes $\mathcal{O}(1)$ seconds for the python-only version as well as the version using the compiled C++ module.

SIGWfast is free software that is distributed under the MIT License: This means that you can use, copy, modify, merge, publish and distribute, sublicense, and/or sell copies of it, and to permit persons to whom it is furnished to do so it under the terms of the MIT License. The rest of this User Guide is structured as follows. In sec. 2 we explain in detail what SIGWfast computes. Sec. 3 is dedicated to instruction for how to install, configure and use SIGWfast. In sec. 4 we show a couple of example computations. Some potential issues and their solution are listed in sec. 5. A link to the precise wording of the MIT License can be found in sec. 6. We hope that you will find SIGWfast useful!

2 What does SIGWfast compute?

The code computes the energy density fraction $\Omega_{\rm GW}(k)$ in gravitational waves today that was induced by scalar fluctuations with primordial power spectrum $\mathcal{P}_\zeta(k)$. As this depends on the equation of state of the universe when the gravitational waves are produced, the package SIGWfast contains two python scripts, `SIGWfast.py` and `SIGWfastEOS.py`, to allow for exploring different expansion histories of the universe after inflation:

- `SIGWfast.py` computes the gravitational wave spectrum induced during a period of radiation domination, which we expect to be the result of interest for most users.
- `SIGWfastEOS.py` computes the gravitational wave spectrum induced during a period of the

[2] For example, if $\mathcal{P}_\zeta$ is to be given by an analytic formula, this does not have to be a vectorizable function.

universe with equation of state w which can be set by the user. As the integration kernel is more involved to allow for different values of w, this script is slightly slower than `SIGWfast.py`.

In the following, we describe in detail the output produced by the two scripts.

2.1 `SIGWfast.py`: gravitational waves induced during radiation domination

For gravitational waves induced during a period of radiation domination, the leading contribution to the fraction of energy density in gravitational waves is given by [7, 8]:[3]

$$\Omega_{\rm GW}^{\rm rad}(k) = \mathcal{N} \int_0^1 {\rm d}d \int_1^\infty {\rm d}s \, \mathcal{T}_{\rm rad}(d,s) \, \mathcal{P}_\zeta\left(\frac{k}{2}(s+d)\right) \mathcal{P}_\zeta\left(\frac{k}{2}(s-d)\right). \tag{1}$$

with [9, 10]

$$\mathcal{T}_{\rm rad}(d,s) = 12 \frac{\left(d^2-1\right)^2\left(s^2-1\right)^2\left(d^2+s^2-6\right)^4}{\left(s^2-d^2\right)^8} \left[\left(\ln\frac{3-d^2}{|s^2-3|} + \frac{2\left(s^2-d^2\right)}{d^2+s^2-6}\right)^2 + \pi^2\Theta\left(s-\sqrt{3}\right)\right]. \tag{2}$$

To get the spectrum of gravitational waves today, the prefactor $\mathcal{N}$ needs to be set to $\mathcal{N} = c_g \Omega_{\rm r,0}$, where $\Omega_{\rm r,0} = 8 \cdot 10^{-5}$ corresponds to the energy density fraction in radiation today and c_g depends on the number of relativistic species during the time of gravitational waves generation vs. today as

$$c_g = \frac{g_{*,\rm rad}}{g_{*,0}} \left(\frac{g_{*,0}}{g_{*S,\rm rad}}\right)^{4/3}. \tag{3}$$

If the matter content of the universe is that of the Standard Model only, and during gravitational wave production all particles were relativistic, one finds $c_g \approx 0.4$. To allow for different values, in `SIGWfast.py` the prefactor $\mathcal{N}$ is implemented as the variable `norm` that can be set by the user.[4]

The principal input for `SIGWfast.py` is the primordial power spectrum $\mathcal{P}_\zeta(k)$ that can be provided in numerical or analytical form. The script then computes the result of expression (1), i.e.

$$\text{Output } \Omega_{\rm GW}^{\rm out} \text{ of } \texttt{SIGWfast.py}: \quad \Omega_{\rm GW}^{\rm out}(k) = \Omega_{\rm GW}^{\rm rad}(k)\,. \tag{4}$$

The result is saved in a `.npz` file in the 'data/' subdirectory containing the user-specified array of k-values, and the computed corresponding values $\Omega_{\rm GW}^{\rm out}$. In addition, both $\mathcal{P}_\zeta(k)$ and $\Omega_{\rm GW}(k)$ are plotted.

2.2 `SIGWfastEOS.py`: gravitational waves induced during a phase with equation of state w

If inflation is not immediately followed by a transition to radiation domination, scalar-induced gravitational waves can also be produced during an era where the universe is not dominated by radiation. One possibility is to consider a universe that before the epoch of radiation domination is described by a constant equation of state, parameterised by w. For gravitational waves induced during such a period, the contribution to the energy density fraction in gravitational waves is given by [3, 11, 12]:[5]

$$\Omega_{\rm GW}^{w}(k) = \mathcal{N} \left(\frac{k}{k_{\rm rh}}\right)^{-2b} \int_0^1 {\rm d}d \int_1^\infty {\rm d}s \, \mathcal{T}_{w}(d,s) \, \mathcal{P}_\zeta\left(\frac{k}{2}(s+d)\right) \mathcal{P}_\zeta\left(\frac{k}{2}(s-d)\right), \tag{5}$$

[3] Strictly speaking, the leading scalar-induced contribution to $\Omega_{\rm GW}$ depends on the four-point function of scalar fluctuations $\langle\zeta\zeta\zeta\zeta\rangle$. This can then be split into two pieces, one involving $\mathcal{P}_\zeta\mathcal{P}_\zeta$, which is what is shown in (1), and an additional piece involving the primordial trispectrum [15], which we ignore here. The reason is that for most well-behaved inflation models (i.e. models that remain within the bounds of perturbative control) the piece with $\mathcal{P}_\zeta\mathcal{P}_\zeta$ dominates so that the contribution from the trispectrum can be safely ignored [16].

[4] That is, setting `norm = 1` in `SIGWfast.py` corresponds to setting $\mathcal{N} = 1$ in (1).

[5] Again, we ignore the contribution from the primordial trispectrum for reasons as explained in footnote 3.

with

$$b \equiv \frac{1-3w}{1+3w}\,. \tag{6}$$

The integration kernel $\mathcal{T}_w(d,s)$ takes the form:

$$\begin{aligned}\mathcal{T}_w(d,s) = \;& \mathcal{F}(b)\left(\frac{(d^2-1)(s^2-1)}{d^2-s^2}\right)^2 \frac{|1-y^2|^b}{(s^2-d^2)^2} \times \\ & \times \Bigg\{ \left[\mathsf{P}_b^{-b}(y) + \frac{2+b}{1+b}\mathsf{P}_{b+2}^{-b}(y)\right]^2 \Theta\big(s-c_s^{-1}\big) \\ & + \frac{4}{\pi^2}\left[\mathsf{Q}_b^{-b}(y) + \frac{2+b}{1+b}\mathsf{Q}_{b+2}^{-b}(y)\right]^2 \Theta\big(s-c_s^{-1}\big) \\ & + \frac{4}{\pi^2}\left[\mathcal{Q}_b^{-b}(-y) + 2\frac{2+b}{1+b}\mathcal{Q}_{b+2}^{-b}(-y)\right]^2 \Theta\big(c_s^{-1}-s\big)\Bigg\}\,,\end{aligned} \tag{7}$$

with

$$y \equiv \frac{s^2+d^2-2c_s^{-2}}{s^2-d^2}\,, \quad \mathcal{F}(b) = \frac{1}{3}\left(\frac{4^{1+b}(b+2)}{(1+b)^{1+b}\,(2b+3)c_s^2}\,\Gamma^2\Big[b+\tfrac{3}{2}\Big]\right)^2\,, \tag{8}$$

where $\mathsf{P}_\nu^\mu(x)$ and $\mathsf{Q}_\nu^\mu(x)$ are the Ferrer's function of the first and second kind, respectively, and $\mathcal{Q}_\nu^\mu(x)$ is the associated Legendre function of the second kind. We recorded their definitions in appendix A.

The factor $(k/k_{\rm rh})^{-2b}$ accounts for the fact that before the transition to radiation domination, gravitational waves redshift differently from the bulk. The 'reheating scale' $k_{\rm rh}$ corresponds to the comoving wavenumber that entered the horizon at the transition to radiation domination, i.e. $k_{\rm rh} = a_{\rm rh}H_{\rm rh}$, and is in general different for different cosmological models. As in `SIGWfast.py`, the normalisation factor $\mathcal{N}$ is implemented as the variable `norm` whose value can be specified by the user. To obtain at the energy density fraction in gravitational waves today, this should be set to $\mathcal{N} = c_g\Omega_{\rm r,0}$, see the discussion in sec. 2.1 for more details.

The principal inputs for `SIGWfastEOS.py` are the value of w and the primordial power spectrum $\mathcal{P}_\zeta(k)$, which can be provided in numerical or analytical form. As the kernel in (7) is valid for $0 < w < 1$, only values of w in this range are allowed as inputs. The code `SIGWfastEOS.py` then computes the gravitational spectrum induced for either a universe described by an adiabatic perfect fluid ($c_s^2 = w$) or a universe dominated by a canonically normalized scalar field $c_s^2 = 1$. The user can choose between these two options by setting the flag `cs_equal_one` to `False` or `True`, respectively. For example, to reproduce the results of `SIGWfast.py` for radiation domination, in `SIGWfastEOS.py` one would need to set $c_s^2 = w = 1/3$ (i.e. `w = 1/3` and `cs_equal_one = False`).[6] Instead of providing $k_{\rm rh}$ as a further input, the script `SIGWfastEOS.py` computes (5), but with the $k_{\rm rh}$-dependent term scaled out, i.e.

$$\text{Output } \Omega_{\rm GW}^{\rm out} \text{ of } \texttt{SIGWfastEOS.py}\text{:}\quad \Omega_{\rm GW}^{\rm out}(k) = \big(k_{\rm ref}/k_{\rm rh}\big)^{2b} \times \Omega_{\rm GW}^{w}(k)\,. \tag{9}$$

Here $k_{\rm ref}$ is the reference unit for wavenumbers k as chosen by the user.[7] As the factor $\big(k_{\rm ref}/k_{\rm rh}\big)^{2b}$ is a constant, its only effect is an overall rescaling which does not affect the spectral shape of $\Omega_{\rm GW}^w$ as a function of k.

Again, the result is saved in a `.npz` file in the 'data/' subdirectory containing the user-specified array of k-values, and the computed corresponding values $\Omega_{\rm GW}^{\rm out}$. In addition, both $\mathcal{P}_\zeta(k)$ and $\Omega_{\rm GW}(k)$ are plotted.

[6] For $c_s^2 = w = 1/3$ the kernel $\mathcal{T}_w$ in (7) exhibits divergent pieces, which one can show to cancel analytically, so that the kernel reduces to $\mathcal{T}_{\rm rad}$ in (2) as expected. To deal with these unphysical divergences numerically, when inputting a value w with $|w-1/3| < \epsilon$ in `SIGWfastEOS.py`, the value of w is redefined automatically to $w = 1/3 - \epsilon$ with $\epsilon = 10^{-4}$. This avoids the divergences without changing the result in a significant way.

[7] The scale $k_{\rm ref}$ is defined in the sense that writing `k = 0.1` in the python script corresponds to $k = 0.1k_{\rm ref}$.

3 Prerequisites, installation and configuration

3.1 Prerequisites

SIGWfast can be used on any system that supports python3. We recommend using python environments and a package manager such as 'conda'. The following python modules are required:[8]

math, matplotlib, numpy, os, scipy, sys, time, tqdm.

Optional C++ extension: This is only supported on systems running on Linux or MacOS. Compiling the C++ extension further requires the modules:

distutils, platform, shutil, subprocess,

and a working C++ compiler.

3.2 Installation

Download `SIGWfast.zip`. After decompression the necessary files and directory structure are already in place in the parent directory 'SIGWfast'.

3.3 File content

The parent directory 'SIGWfast' contains two python scripts `SIGWfast.py` and `SIGWfastEOS.py`, whose execution computes the result. Their respective outputs are described in detail in sec. 2.

The subdirectory 'libraries' contains files necessary for performing the computation and which do not need to be modified by the user.:

- `sdintegral.py` contains the definitions and kernels for computing the relevant integrals.
- `SIGWfast.cpp` is a C++ script to perform the integrals in the computation of $\Omega_{\rm GW}$.
- `setup.py` is the code that configures and executes the compilation of the python module that will execute the C++ code contained in `SIGWfast.cpp`. The resulting python module named 'sigwfast' is also deposited in the 'libraries' subdirectory. The script `setup.py` is only called if the option to use the C++ extension is chosen by the user.

The subdirectory 'data' will receive the result data for $\Omega_{\rm GW}(k)$ in a `.npz` file. Also, if the input is a scalar power spectrum in numerical form, this needs to be provided in the 'data' subdirectory as a `.npz` file. As an example, the file `P_of_k.npz` with data for $\mathcal{P}_\zeta(k)$ for the inflation model (11) is included.

3.4 Quick guide

Set flags and values for input parameters in the block of code titled 'Configuration'. Provide the scalar power spectrum either by defining a function `Pofk(k)` in the block of code titled 'Primordial scalar power spectrum' or in form of numerical data in a file 'data/filenamePz.npz'. See the more detailed guide below for how this file is to be prepared. After this, you're good to go!

3.5 Configuration of `SIGWfast.py` and `SIGWfastEOS.py`: step-by-step guide

This is a set of detailed instructions for configuring `SIGWfast.py` and `SIGWfastEOS.py`. The first seven steps are universal to both scripts. To this end open `SIGWfast.py` or `SIGWfastEOS.py` and go to the block of code labeled 'Configuration', which is the first block after the header where the necessary modules are imported. In `SIGWfast.py` the relevant section of code looks like in fig. 1. This is where you can adjust the script for your purposes. The necessary steps are as follows:

[8] The modules 'time' and 'tqdm' are for timing the computation and displaying a progress bar, respectively, and could be dispensed with by commenting out appropriate lines of code.

```
#============================================================================#
                                  # CONFIGURATION #
#============================================================================#

# Name of file where Omega_GW(k) is to be stored as a .npz file in the data
# subdirectory. The k-values and Omega-GW-values will be stored with keywords
# 'karray' and 'OmegaGW' respectively.
filenameGW = 'OmegaGW_of_k'

# Choose whether to regenerate the data. If True, Omega_GW(k) is recomputed in
# every run. If False, the data in data/filenameGW.npz is plotted and new data
# is only computed if this file is absent. This is a safety-measure to not
# overwrite previous data.
regenerate = True #False

# Choose whether to compute Omega_GW from a primordial scalar power spectrum
# given in numerical or analytical form. Set Num_Pofk = True for using
# numerical data as input for P(k). Set Num_Pofk = False for using an anlytical
# formula for P(k) to be defined below.
Num_Pofk = False #True

# If numerical data is to be used as input for P(k) (Num_Pofk = True), declare
# the name of the .npz file containing the data. This file is to be placed in
# the data subdrectory and prepared in a way so that k-values and associated
# P(k)-values are accessed via the keywords 'karray' and 'Pzeta'. This file is
# not needed if P(k) is provided via an analytic formula instead and no name
# needs to be declared.
filenamePz = 'P_of_k'

# Choose whether to compile a C++ module to perform the integration. If set to
# False the entire computation is peformed using existing python modules only.
# Note that Use_Cpp = True will only work for Linux / MacOS, but not Windows.
Use_Cpp = False #True

# Set the normalisation factor that multiplies Omega_GW.
norm = 1

# Set limits kmin and kmax of the interval in k for which Omega_GW is to be
# computed. Also set the number nk of entries of the k-array.
kmin = 0.01 # | in some arbitrary reference units
kmax = 2.50 # | denoted by k_{ref} in the plots.
nk   = 200

# Fill the array of k-values for which Omega_GW is to be computed.
komega = np.linspace(kmin,kmax,nk,dtype=np.float64) # linear spacing
#komega = np.geomspace(kmin,kmax,nk,dtype=np.float64) # logarithmic spacing
```

Figure 1: Configuration section of `SIGWfast.py` for the computation of the result in fig. 3. To compute the same result using the compiled C++ module, set the flag `Use_Cpp = True`.

1. Set `filenameGW`. Choose a name for the `.npz` file that will contain the results for $\Omega_{\rm GW}(k)$ and that will be deposited in the 'data' subdirectory. The k-values and corresponding $\Omega_{\rm GW}$-values in this file will be accessible via the keywords 'karray' and 'OmegaGW'.[9] Note that if a file with this name already exists, a new run will in general overwrite the old file. To avoid this, see the next step.

2. Set the flag `regenerate`. If this is set to `True`, a run of the code will execute a new computation and save the result in the file 'data/'+filenameGW+'.npz', possibly overwriting an old file of the same name. If the flag is set to `False`, after hitting run, the code checks whether a file 'data/'+filenameGW+'.npz' already exists. If this is the case, no new computation is performed and instead the data in the existing file is plotted. This is a safety-measure to avoid existing data to be overwritten by accident. If however 'data/'+filenameGW+'.npz' does not exist, the code proceeds to performing the computation and saving the new data.

3. Set the flag `Num_Pofk`. The code computes the scalar-induced gravitational wave spectrum using the primordial scalar power spectrum $\mathcal{P}_\zeta(k)$ as input. $\mathcal{P}_\zeta(k)$ can be provided in terms of numerical data or an analytic formula, the choice of which is declared by specifying the flag `Num_Pofk`. If set to `True`, $\Omega_{\rm GW}$ will be computed from a scalar power spectrum given by the numerical data in a `.npz` file in the 'data' subdirectory. The name of the file can be specified in the next step. If the flag `Num_Pofk` is instead set to `False`, $\Omega_{\rm GW}$ will be computed from a scalar power spectrum that needs to declared explicitly as a function `Pofk(k)`. See sec. 3.6 for for detailed instructions on how this is to be done.

4. Optional: declare `filenamePz`. In case $\Omega_{\rm GW}(k)$ is to be computed from numerical data (`Num_Pofk = True`), give here the name of the `.npz` file located in the 'data' subdirectory. The file is to

[9] That is, to access the data in this file, load it via `data = numpy.load('data/'+filename+'.npz')`. The arrays with the values of k and $\Omega_{\rm GW}$ are then given by `data['karray']` and `data['OmegaGW']`, respectively.

be prepared so that k-values and associated $\mathcal{P}_\zeta(k)$-values are to be accessed via the keywords 'karray' and 'Pzeta', respectively. If $\mathcal{P}_\zeta(k)$ is to be provided via an analytic formula instead (`Num_Pofk = False`), no input file is needed and this line of code is ignored.

5. Set the flag `Use_Cpp`. If set to `True`, the code imports methods from the compiled module 'sigwfast' to perform the integration, or, if the module does not yet exist, initiates a compilation of it from 'libraries/SIGWfast.cpp'. This option is only available for Linux and MacOS systems and requires a functioning C++ compiler in addition to python. The module 'sigwfast' only needs to be compiled once and can then be used for all further computations.[10] That is, if the input power spectrum $\mathcal{P}_\zeta(k)$ is changed (and/or in the case of `SIGWfastEOS.py` the parameter w) the originally compiled module can still be used and does *not* need to be recompiled. If `Use_Cpp` is instead set to `False`, the entire computation is done within python, using only the modules listed above under 'Prerequisites'.

6. Set `norm`. This is the "normalization" factor $\mathcal{N}$ in (1) and (5). For $\mathcal{N} = 1$ (i.e. `norm = 1`) the output of `SIGWfast.py` and `SIGWfastEOS.py` corresponds to the energy density fraction in gravitational waves at the time of radiation domination. To get the corresponding result for today requires setting $\mathcal{N} = c_g\Omega_{\mathrm{r},0}$, see sec. 2 above.

7. Declare the range of wavenumbers k for which $\Omega_{\mathrm{GW}}(k)$ is to be computed. The corresponding array is called `komega` accordingly. In SIGWfast, the default way of doing this is to define both a lower limit `kmin`, an upper limit `kmax` and setting the number of entries `nk` of `komega`. The values of k can be specified in any unit of choice. The default option is then to fill `komega` with values that are linearly spaced (`numpy.linspace`), but logarithmic spacing (`numpy.geomspace`) or any other method are also allowed, with the only restriction that `komega` should be a numpy array.

   Note that for an analytic primordial power spectrum as input (`Num_Pofk = False`) a good guideline is to choose `komega` such that $\mathcal{P}_\zeta(k)$, when sampled over `komega`, exhibits all relevant features of the full scalar power spectrum.[11] The reason is that for better robustness and performance, $\mathcal{P}_\zeta(k)$ is first discretized over an array `kpzeta`, before an interpolation is used in the computation. By default, in SIGWfast `kpzeta` is constructed from `komega` by extending its range and increasing the density of points, which should be sufficient for most applications. For specialised applications `kpzeta` can also be defined independently from `komega`, see sec. 3.6 for details, so that `komega` can be chosen freely.

There are two further configuration steps needed for `SIGWfastEOS.py`:

8. `In SIGWfastEOS.py` we also need to set a value for the equation of state parameter w for the era during which the gravitational waves are induced. In `SIGWfast.py` this value is hard-coded to $w = 1/3$ corresponding to radiation domination. In `SIGWfastEOS.py` this parameter can be specified through the variable `w`, which can take values in $0 <$ `w` < 1, corresponding to the range of validity of the transfer functions used.

9. Set the flag `cs_equal_one`. `In SIGWfastEOS.py` the computation of $\Omega_{\mathrm{GW}}(k)$ can be done for a universe behaving like a perfect adiabatic fluid ($c_s^2 = w$), or a universe whose energy is dominated by a canonically normalised scalar field ($c_s^2 = 1$), see sec. 2. By setting `cs_equal_one = True` the computation is performed for the canonical scalar field case ($c_s^2 = 1$), while for `cs_equal_one = False` it is the adiabatic perfect fluid ($c_s^2 = w$) result that is computed.

[10] The reason is that 'sigwfast' contains methods to perform in the integrations over d and s in (1) and (5), but is ignorant of the integrands.

[11] This is also good practice from the physics point-of-view, as one expects $\Omega_{\mathrm{GW}}(k)$ and $\mathcal{P}_\zeta(k)$ to exhibit structure on roughly the same scales.

3.6 How to input the primordial scalar power spectrum $\mathcal{P}_\zeta(k)$

These instructions apply to both `SIGWfast.py` and `SIGWfastEOS.py` and concern the block of code after 'Configuration' and titled 'Primordial scalar power spectrum'.

The principal input for SIGWfast is the primordial scalar power spectrum $\mathcal{P}_\zeta(k)$. This can be provided as an analytical formula or in terms of numerical data:

- **Analytical formula** (`Num_Pofk = False`)**:** If an analytic scalar power spectrum is to be used as input, this is to be defined here as the function `Pofk(k)`. This should take a single argument which is the wavenumber k and return the corresponding value of $\mathcal{P}_\zeta$. Additional parameters of the power spectrum need to be declared as either global or local variables with given values. To keep the code as general as possible, there are no further restrictions on how `Pofk(k)` is to be defined. As long as calling the function `Pofk(k)` with a float argument returns a float, the script should run without any problems. The default example included with the code is the scalar power spectrum obtained for a strong sharp turn in the inflationary trajectory, see eq. (2.25) in [17] and sec. 4 below.

  In SIGWfast, `Pofk(k)` is first discretized by evaluating it on an array of k-values, before an interpolation function is then used in the computation.[12] The discretization is performed by evaluating `Pofk(k)` on k-values given in the array `kpzeta` which by default is an extended and denser version of `komega`.[13] If needed, `kpzeta` can be defined here by the user in any other way.[14] For SIGWfast to produce meaningful results, `kpzeta` needs to be sufficiently dense so that the discretization of `Pofk(k)` faithfully captures the relevant features of $\mathcal{P}_\zeta(k)$.

- **Numerical data** (`Num_Pofk = True`)**:** If numerical input is to be used for the scalar power spectrum, this is to be provided in a file 'data/'+filenamePz+'.npz'. Here, 'filenamePz' refers to any name chosen for this file by the user and which can be declared in step 4 of the configuration, see sec. 3.5. The `.npz` file should be prepared to contain an array of k-values and an array of corresponding $\mathcal{P}_\zeta$-values, which should be accessible via the keywords 'karray' and 'Pzeta', respectively. That is, after loading the data in this file via the command `Pdata = numpy.load('data/'+filenamePz+'.npz')`, the arrays of k-values and $\mathcal{P}_\zeta$-values should be given by `Pdata['karray']` and `Pdata['Pzeta']`, respectively.[15]

The script is now be ready to be run!

4 Examples

A great model for illustrating the functionality of SIGWfast is the contribution to the scalar power spectrum due to a strong sharp turn in the inflationary trajectory. This produces a localized peak in $\mathcal{P}_\zeta(k)$ that exhibits further substructure in the form of oscillations. For a constant turn rate, the

[12] The reason is that SIGWfast uses 'vectorization' to accelerate the computation. This means that a function, when provided with an array of values as input, outputs the corresponding result values also as an array. For vectorization to be applicable to `Pofk(k)` directly, certain rules in the definition of `Pofk(k)` would have to be followed. For example, piecewise-defined functions could not be defined via `if` statements, but constructions using e.g. `numpy.where` would have to be used. Also, some functions that are provided in the `scipy` package are not vectorizable by default. The method of discretizing and then interpolating is thus used to not burden the user with such restrictions: `Pofk` can be defined without any vectorization requirements as the interpolation function is vectorizable in any case. Thus, for example, `if` statements can be used in the definition of `Pofk`, as can be seen in the default example provided.

[13] By default, `kpzeta` ranges from `kmin/2` to `2*kmax` and contains $4\times$ as many entries as `komega`.

[14] Consider the following situation where the default definition of `kpzeta` would lead to wrong results. Take the case where $\mathcal{P}_\zeta(k)$ is peaked around a value $k = k_\star$. For SIGWfast to produce meaningful results, `kpzeta` should span an interval that contains $k_\star$. At the same time one may be exclusively interested in the IR tail of $\Omega_{\rm GW}(k)$, hence choosing `komega` to only contain values $k \ll k_\star$. In this case `kpzeta` should be defined independently from `komega`.

[15] To make the arrays of k-values and $\mathcal{P}_\zeta$-values accessible via these keywords, the file needs to be prepared as `numpy.savez('data/'+filenamePz, karray=kvalues, Pzeta=Pvalues)`, where `kvalues` and `Pvalues` refer to the arrays storing the corresponding values.

```
################################################################################
################ DEFINE YOUR OWN SCALAR POWER SPECTRUM HERE: ################
################################################################################

# Default example: P(k) as arises for a strong sharp turn in the inflationary
# trajectory, see eq. (2.25) in arXiv:2012.02761. This exhibits O(1)
# oscillations modulating a peaked envelope. Here P(k) is normalised by a
# factor np.exp(-2*delta*etap) to avoid excessively large values. To account
# for this, one should multiply the final result for Omega_GW by a factor
# np.exp(4*delta*etap).

# Define the model parameters as global variables with fixed values.
# Duration delta of the turn in e-folds
delta = 0.5
# Strength of the turn, i.e. turn rate in units of the Hubble scale
etap  = 14
# Normalisation of power spectrum
P0    = 1 # 2.4*10**(-9) for CMB value

def Pofk(k):
    # P(k) in eq. (2.25) of arXiv:2012.02761 is valid for 0 < k < 2.
    # For some values of delta and etap one finds unphysical spikes for
    # k -> 0 or k -> 2. We remove these by cutting P(k) at k=kcut and k=2-kcut.
    kcut  = 0.001
    if k > 2-kcut:
        P = 0
    elif k < kcut:
        P = 0
    else:
        # Eq. (2.25) in arXiv:2012.02761 multiplied by np.exp(-2*delta*etap)
        P = np.exp(2*(np.sqrt((2-k)*k)-1)*delta*etap)/4/(2-k)/k*(
            1+(k-1)*np.cos(2*np.exp(-delta/2)*etap*k)
             +np.sqrt(abs(2-k)*k)*np.sin(2*np.exp(-delta/2)*etap*k))
    return P0*P

################################################################################
################################################################################
################################################################################

# Define the array of k-values over which P(k) is to be discretized and then
# interpolated. This should include the entire interval where P(k) is not
# negligible and will hence contribute to Omega_GW(k).
# By default, this interval is defined to be wider than komega by a factor
# fac^2 in log(k)-space and also to contain more entries than komega by a
# factor (int(fac))^2. The default value of this factor is chosen as fac=2.
fac = 2
kpzeta = np.linspace(np.amin(komega)/fac, np.amax(komega)*fac,
                     (len(komega))*(int(fac))**2)
# Uncomment lines below if logarithmic spacing is desired.
#kpzeta = np.geomspace(np.amin(komega)/fac, np.amax(komega)*fac,
#                      (len(komega))*(int(fac))**2)
```

Figure 2: Section of code where the analytical expression for $\mathcal{P}_\zeta(k)$ can be defined as the function `Pofk(k)`, here with the implementation of the formula given in eq. (10) as provided in the download versions of `SIGWfast.py` and `SIGWfastEOS.py`. Note the additional factor of $\exp(-2\eta_\perp\delta)$ compared to (10) to avoid excessively large values. In the lower part the default definition of the array `kpzeta` is shown, over which $\mathcal{P}_\zeta(k)$ is discretized in the computation.

contribution to $\mathcal{P}_\zeta(k)$ can be computed analytically and takes the form [17, 18]:

$$\mathcal{P}_\zeta(\kappa) = \mathcal{P}_{\rm env}(\kappa)\left[1+(\kappa-1)\cos\left(2e^{-\frac{\delta}{2}}\eta_\perp\kappa\right)+\sqrt{(2-\kappa)\kappa}\,\sin\left(2e^{-\frac{\delta}{2}}\eta_\perp\kappa\right)\right]\times\Theta(2-\kappa)\,, \qquad (10)$$

$$\text{with}\quad \mathcal{P}_{\rm env}(\kappa) = \mathcal{P}_0\frac{e^{2\sqrt{(2-\kappa)\kappa}\,\eta_\perp\delta}}{4(2-\kappa)\kappa}\,,\quad \text{and}\quad \kappa = k/k_\star\,,$$

where $\Theta(x)$ denotes the Heaviside theta function and $k_\star$ corresponds to the locus where the 'envelope' $\mathcal{P}_{\rm env}(k)$ of the scalar power spectrum peaks. The other parameters are *(i)* δ, the duration of the turn in e-folds, *(ii)* $\eta_\perp$, the turn rate in Hubble units, and *(iii)* $\mathcal{P}_0$, the 'background value' of the power spectrum in absence of the turn.

The formula (10) is the default power spectrum implemented in the downloadable versions of both `SIGWfast.py` and `SIGWfastEOS.py`, see fig. 2 for the relevant lines of code. To avoid excessively large numbers due to the exponentially enhanced envelope, in the function `Pofk(k)` defined in the python scripts the envelope is normalised to unity at its maximum at $\kappa = 1$. This is done by multiplying (10) by an additional factor of $\exp(-2\eta_\perp\delta)$.[16] The argument of `Pofk(k)` corresponds to the variable κ in (10), as we choose to measure k in units of $k_\star$. The parameters (δ, $\eta_\perp$, $\mathcal{P}_0$) are implemented as the global variables (`delta`, `etap`, `P0`).

As the standard example we consider a moderately sharp turn with $\delta = 0.5$ and choose $\eta_\perp = 14$ to have an $\mathcal{O}(1)$ number of oscillations across the peak-region, see the left panel of fig. 3. We further

[16] This can be 'undone' in the end by multiplying the computed result for $\Omega_{\rm GW}(k)$ by a factor $\exp(4\eta_\perp\delta)$.

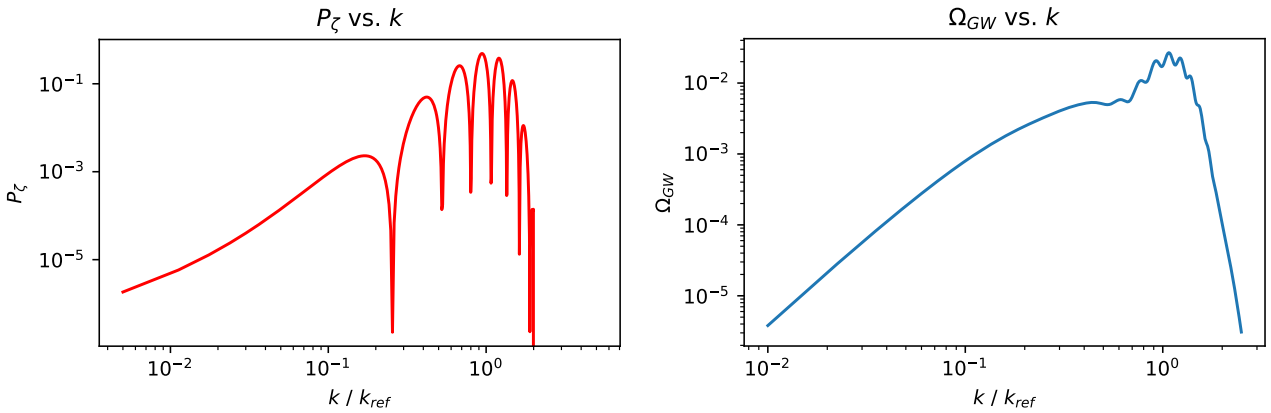


Figure 3: Scalar power spectrum $\mathcal{P}_\zeta(k)$ (left panel) of the standard example in (11) and the corresponding scalar-induced gravitational wave spectrum $\Omega_{\rm GW}(k)$ (right panel) as computed by `SIGWfast.py` (or `SIGWfastEOS.py` with `w = 1/3` and `cs_equal_one = False`). The unit $k_{\rm ref}$ corresponds to $k_\star$, the maxmimum of the envelope of $\mathcal{P}_\zeta(k)$. Using the python-only version (`Use_Cpp = False`) the computation takes 1.3s with `SIGWfast.py` and 3.0s with `SIGWfastEOS.py` on the development machine (Macbook Pro with M1 CPU). Using the compiled C++ module (`Use_Cpp = True`) the same computation takes 0.9s with `SIGWfast.py` and 2.6s with `SIGWfastEOS.py`.

set $\mathcal{P}_0 = 1$ for simplicity, as its effect is just an overall rescaling:

Standard example: (10) with $\delta = 0.5,\ \eta_\perp = 14,\ \mathcal{P}_0 = 1,$ multiplied by a factor $e^{-2\eta_\perp \delta}$. (11)

4.1 Gravitational waves induced during radiation domination

The simplest way of computing $\Omega_{\rm GW}(k)$ in this case is to use `SIGWfast.py`, which has been written for precisely this purpose. However, one can also use `SIGWfastEOS.py` as long as one also sets `w=1/3` and `cs_equal_one = False` in the configuration section.

We choose configuration settings as in fig. 1. We have set `Num_Pofk = False` to use the analytical formula for the scalar power spectrum. We wish to compute $\Omega_{\rm GW}(k)$ for wavenumbers in the range $k \in [0.01k_\star, 2.5k_\star]$, i.e. setting `kmin = 0.01` and `kmax = 2.5`, which encloses the maximum of $\mathcal{P}_{\rm env}$. We then define the array `komega` by covering this interval with `nk = 200` points that are linearly spaced (`numpy.linspace`), which we expect to provide a sufficient resolution of the result. The 'normalization' factor $\mathcal{N}$ in (1) or (5) has been set to unity (`norm = 1`) for simplicity.

When executing `SIGWfast.py` (or `SIGWfastEOS.py`), the script computes $\Omega_{\rm GW}(k)$ from the input scalar power spectrum and saves it in the file 'data/OmegaGW_of_k.npz'. It then displays plots of both $\mathcal{P}_\zeta(k)$ and $\Omega_{\rm GW}(k)$, which we show in fig. 3. The following points are worth highlighting.

- The plot of $\mathcal{P}_\zeta(k)$ is an interpolation of the power spectrum evaluated on `kpzeta`. Here we use the standard definition of `kpzeta`, see footnote 13. As this interpolation is used in the computation of $\Omega_{\rm GW}(k)$, it is important that it faithfully captures the relevant features of $\mathcal{P}_\zeta(k)$, which one can check by inspecting the plot. Here we find that $\mathcal{P}_\zeta(k)$ is indeed resolved sufficiently by the interpolation, including the oscillations, which are hence 'visible' to the computation. The range of `kpzeta` is also wide enough to capture the entire region where $\mathcal{P}_\zeta(k)$ is enhanced, which ensures that all these scales contribute to $\Omega_{\rm GW}(k)$. If this was not the case, we can either increase the range or resolution of `komega` (as this sets `kpzeta` in the default setting) or define a suitable `kpzeta` independently.

- The plot of $\Omega_{\rm GW}(k)$ in the right panel is the main output of SIGWfast, and the corresponding data has been saved in the file 'OmegaGW_of_k.npz' in the 'data' subdirectory. The spectrum $\Omega_{\rm GW}(k)$ exhibits the characteristic morphology of gravitational waves induced by scalar fluctuations with a peak in $\mathcal{P}_\zeta(k)$ at $k = k_\star$, i.e. a broad bump around $k \sim 0.4k_\star$ and principal peak

Number of points `nk` computed	200	400	1000	2000
$t_{\rm comp}$ of `SIGWfast.py` (python-only version)	1.2s	2.5s	6.5s	13.6s
$t_{\rm comp}$ of `SIGWfast.py` (with C++ module)	0.9s	1.9s	5.0s	10.7s
$t_{\rm comp}$ of `SIGWfastEOS.py` (python-only version)	2.8s	4.1s	8.1s	15.3s
$t_{\rm comp}$ of `SIGWfastEOS.py` (with C++ module)	2.5s	3.5s	6.7s	12.4s

Table 1: Comparison of computation times $t_{\rm comp}$ of `SIGSWfast.py` and `SIGWfastEOS.py` (for the python-only version and the version using the compiled C++ module) for different values of `nk`, with the other configuration parameters as in fig. 1. In `SIGWfastEOS.py` we have further set `w=1/3` and `cs_equal_one=False` to replicate the results of `SIGWfastEOS.py`. Computations were done on the development machine (Macbook Pro with M1 CPU) for the standard example (11) as input for $\mathcal{P}_\zeta(k)$. We observe that `SIGWfastEOS.py` is slower than `SIGWfast.py` by a constant offset of $\sim$1.6s, which is the time needed to load the more complicated integration kernel (7) that allows for different values of w. We also find that using the compiled C++ module generally reduces computation times by $\sim$20%-25%.

around $k \simeq 2/\sqrt{3}k_\star$. The most conspicuous property are however the oscillatory modulations of the principal peak, which are a direct consequence of the oscillations in $\mathcal{P}_\zeta(k)$. See [17] for a detailed analysis of the origin and the properties of these modulations in $\Omega_{\rm GW}(k)$. The detection prospects of such an oscillatory gravitational wave spectrum with the upcoming observatory LISA has been studied in [19].

- Using the python-only version (`Use_Cpp = False`) the computation takes 1.3s with `SIGWfast.py` and 3.0s with `SIGWfastEOS.py` on the development machine (Macbook Pro with M1 CPU). The same result can also be computed using the compiled C++ module by changing the corresponding flag in the configuration section to `Use_Cpp = True`, which is available for machines running on MacOS or Linux. Executing python script `SIGWfast.py` or `SIGWfastEOS.py`, a module named 'sigwfast' is compiled, which takes $\mathcal{O}(1)$ seconds. The computation of $\Omega_{\rm GW}(k)$ itself is then shortened to 0.9s with `SIGWfast.py` and 2.6s with `SIGWfastEOS.py`. In any further runs with flag setting `Use_Cpp = True` the compiled module is re-used and does not need to be recompiled, even if the input power spectrum or value of w is changed. For a comparison of computation times as the number of points `nk` computed is changed, see the table 1.

We can also reproduce the results in fig. 3 using a numerical scalar power spectrum as input, by setting the flag `Num_Pofk = True`, in which case the data for $\mathcal{P}_\zeta(k)$ contained in the file 'data/P_of_k.npz' is used. In the downloadable version of SIGWfast, this file contains numerical data for the standard example given in (11).

4.2 Gravitational waves induced during a phase with $w = 0.8$

Using the script `SIGWfastEOS.py` we can also explore the spectrum of gravitational waves induced in an era during which the universe evolved with an equation of state parameter w with $0 < w < 1$. We can adjust the equation of state parameter w by giving a value to the variable `w` in the configuration section. By setting the flag `cs_equal_one` we can further consider a universe evolving as an adiabatic perfect fluid (`cs_equal_one = False`) or dominated by a canonical scalar field (`cs_equal_one = True`).

As input, we again consider the standard example for $\mathcal{P}_\zeta(k)$ as given in (11). In fig. 4 we then show plots of $\Omega_{\rm GW}(k)$ produced by `SIGWfastEOS.py` for `w=0.8` and `cs_equal_one = False` (left panel) and

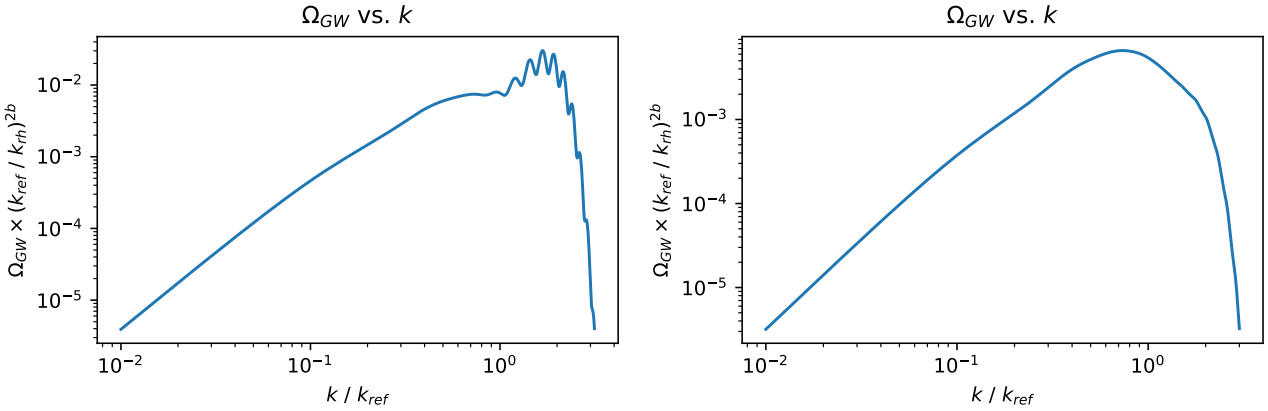


Figure 4: $\Omega_{\rm GW}(k)$ computed with `SIGWfastEOS.py` for `w = 0.8` with `cs_equal_one = False` (left panel) or `cs_equal_one = True` (right panel). The input scalar power spectrum is the same as shown in the left panel of fig. 3.

`cs_equal_one = True` (right panel).[17] On the development machine the computation times are ∼3s (left panel plot) and ∼1s (right panel plot), respectively. Recall that for $w \neq 1/3$ (i.e. $b \neq 0$) the output of `SIGWfastEOS.py` is not $\Omega_{\rm GW}(k)$ today, but $\Omega_{\rm GW}(k) \times (k_{\rm ref}/k_{\rm rh})^{2b}$, see sec. 2.2.

The plots in fig. 4 show that the equation of state of the universe has an important effect on the scalar-induced gravitational wave spectrum. In the left panel one finds that in the adiabatic perfect fluid case the oscillations in $\Omega_{\rm GW}(k)$ for $w = 0.8$ are even more pronounced than for radiation domination, cf. the right panel of fig. 3. Inspecting the right panel of fig. 4 one observes that in the canonical scalar field case $\Omega_{\rm GW}(k)$ does not exhibit any visible oscillations. For further reading on this topic we refer readers to [20], where the effect of the expansion history of the universe on oscillatory signatures in $\Omega_{\rm GW}(k)$ has been analysed in detail.

5 Troubleshooting

SIGWfast has been written for python3 and will not work with python2. It has been developed using python 3.9.7 and 'conda' for environment and package management, but has also been tested on python 3.8. The development machine was a Macbook Pro with a M1 CPU and running MacOS 12.1 Monterey. Python was installed in its x86 version and was running on the M1 chip via the Rosetta2 translator.[18] SIGWfast has also been tested on Ubuntu 20.04.4 running on a CPU with Intel x86 architecture.

5.1 Compiling the C++ module

One possible source of errors is the compilation of the C++ module. This is activated by setting the flag `Use_Cpp = True` in the block of code titled 'Configuration' and its use leads to a 20%-25% reduction in computation times. This option is only available for systems running on Linux and MacOS. When trying to use the C++ option on Windows, the code automatically reverts to the python-only version.

On an older system running python 3.8 on MacOS 10.12 Sierra we encountered the problem that the automatic compilation of the C++ from the code was not initiated. As a result the module 'sigwfast' could not be found and the computation ended with an error. To overcome this, the module 'sigwfast' can be compiled by hand from the command line. It can then be used indefinitely, as it only has to be compiled only once. To do so, open the terminal and go to the 'libraries' subfolder in the parent directory. For definiteness, here we assume that the parent directory 'SIGWfast' is located in the home directory '∼'. Hence, on the command line enter:

[17] We slightly adjusted the limits of `komega` compared to those in the configuration in fig. 1 for better aesthetics.
[18] See the following blog post on running python on Apple Silicon chips.

```
cd ~/SIGWfast/libraries
```

We have to work in this directory so that the file `SIGWfast.cpp` with the C++ code can be found. To compile the module by hand then enter:[19]

```
python3 setup.py install --home=~/SIGWfast/libraries
```

We used the command `python3` to make sure that python3 is used, as the command `python` can sometimes refer to the version of python2 that is shipped together with MacOS. The flag `--home=...` ensures that the module is deposited within the 'libraries' subdirectory, rather than added to the other modules of the python distribution. This makes it easier to remove it later if desired.

6 Licensing

SIGWfast is distributed under the MIT license. You should have received a copy of the MIT License along with SIGWfast. If not, see https://spdx.org/licenses/MIT.html.

Acknowledgements

During the development of SIGWfast Lukas T. Witkowski was supported by the European Research Council under the European Union's Horizon 2020 research and innovation programme (grant agreement No 758792, project GEODESI). We are indebted to Dr. Jacopo Fumagalli, without whom SIGWfast would have never been developed in this form and whose inputs vastly improved the code. We are also grateful to Prof. Sébastien Renaux-Petel: by expanding his group's research agenda to include scalar-induced gravitational waves, he made the development of SIGWfast possible. We also thank Dr. John W. Ronayne, whose immense knowledge of python helped get this project off the ground.

A Legendre functions on the cut and associated Legendre functions

For reference, below we collect the definitions of $\mathsf{P}^\mu_\nu(x)$, $\mathsf{Q}^\mu_\nu(x)$ and $\mathcal{Q}^\mu_\nu(x)$ as can be found in [21]:

$$\mathsf{P}^\mu_\nu(x) = \left(\frac{1+x}{1-x}\right)^{\mu/2} \frac{1}{\Gamma[1-\mu]} F\big(\nu+1,-\nu;1-\mu;\tfrac{1}{2}(1-x)\big)\,, \tag{12}$$

$$\begin{aligned}\mathsf{Q}^\mu_\nu(x) = \frac{\pi}{2\sin(\pi\mu)}\Bigg[&\cos(\pi\mu)\left(\frac{1+x}{1-x}\right)^{\mu/2} \frac{1}{\Gamma[1-\mu]} F\big(\nu+1,-\nu;1-\mu;\tfrac{1}{2}(1-x)\big)\,, \\ &- \left(\frac{1-x}{1+x}\right)^{\mu/2} \frac{\Gamma[\nu+\mu+1]}{\Gamma[\nu-\mu+1]\Gamma[1+\mu]} F\big(\nu+1,-\nu;1+\mu;\tfrac{1}{2}(1-x)\big)\Bigg]\,,\end{aligned} \tag{13}$$

$$\mathcal{Q}^\mu_\nu(x) = \frac{\pi^{1/2}(x^2-1)^{\mu/2}}{2^{\nu+1}x^{\nu+\mu+1}} \frac{1}{\Gamma[\nu+3/2]} F\big(\tfrac{1}{2}\nu+\tfrac{1}{2}\mu+1,\tfrac{1}{2}\nu+\tfrac{1}{2}\mu+\tfrac{1}{2};\nu+\tfrac{3}{2};\tfrac{1}{x^2}\big)\,, \tag{14}$$

where $F(a,b;c;x)$ is the Gauss hypergeometric function.

References

[1] A. Achúcarro et al., *Inflation: Theory and Observations*, 2203.08128.

[2] LISA Cosmology Working Group collaboration, *Cosmology with the Laser Interferometer Space Antenna*, 2204.05434.

[19] If using python environments, ensure the correct environment if activated

[3] G. Domènech, *Scalar Induced Gravitational Waves Review*, *Universe* **7** (2021) 398 [2109.01398].

[4] M. Sasaki, T. Suyama, T. Tanaka and S. Yokoyama, *Primordial black holes—perspectives in gravitational wave astronomy*, *Class. Quant. Grav.* **35** (2018) 063001 [1801.05235].

[5] J. García-Bellido, *Primordial Black Holes*, *PoS* **EDSU2018** (2018) 042.

[6] B. Carr, K. Kohri, Y. Sendouda and J. Yokoyama, *Constraints on primordial black holes*, *Rept. Prog. Phys.* **84** (2021) 116902 [2002.12778].

[7] K.N. Ananda, C. Clarkson and D. Wands, *The Cosmological gravitational wave background from primordial density perturbations*, *Phys. Rev. D* **75** (2007) 123518 [gr-qc/0612013].

[8] D. Baumann, P.J. Steinhardt, K. Takahashi and K. Ichiki, *Gravitational Wave Spectrum Induced by Primordial Scalar Perturbations*, *Phys. Rev. D* **76** (2007) 084019 [hep-th/0703290].

[9] J.R. Espinosa, D. Racco and A. Riotto, *A Cosmological Signature of the SM Higgs Instability: Gravitational Waves*, *JCAP* **09** (2018) 012 [1804.07732].

[10] K. Kohri and T. Terada, *Semianalytic calculation of gravitational wave spectrum nonlinearly induced from primordial curvature perturbations*, *Phys. Rev. D* **97** (2018) 123532 [1804.08577].

[11] G. Domènech, *Induced gravitational waves in a general cosmological background*, *Int. J. Mod. Phys. D* **29** (2020) 2050028 [1912.05583].

[12] G. Domènech, S. Pi and M. Sasaki, *Induced gravitational waves as a probe of thermal history of the universe*, *JCAP* **08** (2020) 017 [2005.12314].

[13] R.-G. Cai, S. Pi, S.-J. Wang and X.-Y. Yang, *Resonant multiple peaks in the induced gravitational waves*, *JCAP* **05** (2019) 013 [1901.10152].

[14] S. Pi and M. Sasaki, *Gravitational Waves Induced by Scalar Perturbations with a Lognormal Peak*, *JCAP* **09** (2020) 037 [2005.12306].

[15] C. Unal, *Imprints of Primordial Non-Gaussianity on Gravitational Wave Spectrum*, *Phys. Rev. D* **99** (2019) 041301 [1811.09151].

[16] S. Garcia-Saenz, L. Pinol, S. Renaux-Petel and D. Werth, *No-go Theorem for Scalar-Trispectrum-Induced Gravitational Waves*, 2207.14267.

[17] J. Fumagalli, S. Renaux-Petel and L.T. Witkowski, *Oscillations in the stochastic gravitational wave background from sharp features and particle production during inflation*, *JCAP* **08** (2021) 030 [2012.02761].

[18] G.A. Palma, S. Sypsas and C. Zenteno, *Seeding primordial black holes in multifield inflation*, *Phys. Rev. Lett.* **125** (2020) 121301 [2004.06106].

[19] J. Fumagalli, M. Pieroni, S. Renaux-Petel and L.T. Witkowski, *Detecting primordial features with LISA*, *JCAP* **07** (2022) 020 [2112.06903].

[20] L.T. Witkowski, G. Domènech, J. Fumagalli and S. Renaux-Petel, *Expansion history-dependent oscillations in the scalar-induced gravitational wave background*, *JCAP* **05** (2022) 028 [2110.09480].

[21] "*NIST Digital Library of Mathematical Functions.*" http://dlmf.nist.gov/, Release 1.1.6 of 2022-06-30.