SIGWfast

A python package for the computation of scalar-induced gravitational wave spectra

Lukas T. Witkowski

https://github.com/Lukas-T-W/SIGWfast
lukas.witkowski@iap.fr

Institut d’Astrophysique de Paris, GReCO, UMR 7095 du CNRS et de Sorbonne Université, 98bis boulevard Arago, 75014 Paris, France

Abstract

SIGWfast is a python code to compute the Scalar-Induced Gravitational Wave 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 python 3. The version employing the C++ module is only available for Linux and MacOS systems.
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 (≳ 1 Mpc) 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 (≪ 1 Mpc), 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 [5–7] 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 Scalar fluctuations also induce gravitational waves during inflation, see e.g. [4], which can be promising targets for future detection efforts. Here, SIGW refers exclusively to gravitational waves induced during the post-inflationary era, which is what SIGWfast computes.

2 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. Its properties can be directly related to inflationary-era quantities.
As a result, the computation of the spectrum $\Omega_{gw}(k)$ of the SIGW is in the process of becoming a routine task for inflationary model builders with an interest in mechanisms beyond single-field 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 $P_\zeta(k)$ and a transfer function $T$, i.e. $\Omega_{gw} \sim \int \! \int \! T P_\zeta P_\zeta$. Even though the transfer functions are known explicitly for various post-inflationary histories [12–15], the integration typically has to be performed numerically, and analytic results can only be derived for idealised choices of $P_\zeta$ (like $\delta$-distributions [13,16] or lognormal peaks [17]), 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 are a couple of subtleties to be considered in this case. Firstly, the transfer function $T$ 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 $\Omega_{gw}(k)$ 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 $P_\zeta(k)$ is provided. It can also be run on all systems that have a functioning installation of python 3. For robustness, SIGWfast sets the integration cutoff automatically. It also makes very little demands on how the scalar power spectrum is to be provided. 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 further reduces the computation time by up to 25%. This option is however restricted to systems running on Linux or MacOS. Overall, on a modern laptop or desktop machine a computation of $\Omega_{gw}(k)$ with a quality suitable for publication typically takes $O(1)$ seconds for the python-only version as well as the version using the compiled C++ module.

SIGWfast can be downloaded from here. It 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. If you use SIGWfast, we kindly ask you to cite this arXiv article in any resulting publications.

The rest of this document contains a user guide that is structured as follows. In sec. 2 we explain in detail what SIGWfast computes. Sec. 3 is dedicated to instructions 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 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 $\Omega_{gw}(k)$ in gravitational waves today that was induced by scalar fluctuations with primordial power spectrum $P_\zeta(k)$. 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:

3For example, if $P_\zeta$ is to be given by an analytic formula, this does not have to be a vectorizable function.
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 universe with equation of state parameter \( w \), 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 \([10,11]\):

\[
\Omega_{\text{rad}}^{\text{gw}}(k) = N \int_0^1 \int_1^\infty d s \, d d \, \mathcal{T}_{\text{rad}}(d, s) \, \mathcal{P}_\zeta \left( \frac{k}{2} (s + d) \right) \mathcal{P}_\zeta \left( \frac{k}{2} (s - d) \right) ,
\]

with \([12,13]\)

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

To get the spectrum of gravitational waves today, the prefactor \( N \) needs to be set to \( N = c_g \Omega_{r,0} \), where \( \Omega_{r,0} = 8 \cdot 10^{-5} \) 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

\[
c_g = \frac{g_{*, \text{rad}}}{g_{*,0}} \left( \frac{g_{*,0}}{g_{*,\text{rad}}} \right)^{4/3} .
\]

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 \( N \) is implemented as the variable \( \text{norm} \) that can be set by the user.

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.

\[
\Omega_{\text{out}}^{\text{gw}}(k) = \Omega_{\text{rad}}^{\text{gw}}(k).
\]

The result is saved in a .npz file in the ‘data’ subdirectory containing the user-specified array of \( k \)-values and the corresponding values of \( \Omega_{\text{out}}^{\text{gw}} \). In addition, both \( \mathcal{P}_\zeta(k) \) and \( \Omega_{\text{gw}}(k) \) are plotted.

### 2.2 SIGWfastEOS.py: gravitational waves induced during a phase with equation of state \( w \)

If inflation is not immediately followed by the hot big bang phase, 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

\[\begin{align*}
\text{(4)}
\end{align*}\]
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 \(^{14,15}\)

\[
\Omega_{\text{GW}}^w(k) = \mathcal{N} \left( \frac{k}{k_{\text{rh}}} \right)^{-2b} \int_0^1 \int_1^\infty \frac{d\zeta}{\zeta} T_w(d, s) \mathcal{P}_\zeta \left( \frac{k}{2} (s + d) \right) \mathcal{P}_\zeta \left( \frac{k}{2} (s - d) \right),
\]

with

\[
b = \frac{1}{3} - 3w.
\]

The integration kernel \( T_w(d, s) \) takes the form \(^{3,14,15}\):

\[
T_w(d, s) = \mathcal{F}(b) \left( \frac{(d^2 - 1)(s^2 - 1)}{d^2 - s^2} \right)^2 \left| 1 - \frac{2}{(s^2 - d^2)^2} \right| \\
\times \left\{ \mathcal{P}_b^{-b}(-y) + \frac{2 + b}{1 + b} \mathcal{P}_{b+2}(-y) \right\}^2 \Theta(s - c_s^{-1}) \\
+ \frac{4}{\pi^2} \left[ \mathcal{Q}_b^{-b}(-y) + \frac{2 + b}{1 + b} \mathcal{Q}_{b+2}(-y) \right] \Theta(s - c_s^{-1}) \\
+ \frac{4}{\pi^2} \left[ \mathcal{Q}_b^{-b}(-y) + \frac{2 + b}{1 + b} \mathcal{Q}_{b+2}(-y) \right] \Theta(c_s^{-1} - s) \right\},
\]

with

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

where \( c_s \) is the propagation speed of scalar fluctuations, \( \mathcal{P}_b^p(x) \) and \( \mathcal{Q}_b^p(x) \) are the Ferrers function of the first and second kind, respectively, and \( \mathcal{Q}_b^p(x) \) is the associated Legendre function of the second kind. We recorded their definitions in appendix \(^A\).

The factor \((k/k_{\text{rh}})^{-2b}\) accounts for the fact that before the transition to radiation domination, gravitational waves redshift differently from the bulk. The ‘reheating scale’ \( k_{\text{rh}} \) corresponds to the comoving wavenumber that entered the horizon at the transition to radiation domination, i.e. \( k_{\text{rh}} = a_{\text{rh}}H_{\text{rh}} \), and is in general different for different cosmological models. What is important is that \((5)\) is valid for \( k \gg k_{\text{rh}} \), i.e. for modes that entered the horizon well before reheating, see e.g. \(^3\) for details. That is, we should not use \((5)\) and hence \texttt{SIGWfastEOS.py} to compute the SIGW with \( k \lesssim k_{\text{rh}} \). As in \texttt{SIGWfast.py}, the normalisation factor \( \mathcal{N} \) is implemented as the variable \texttt{norm} whose value can be specified by the user. To obtain the energy density fraction in gravitational waves today, this should be set to \( \mathcal{N} = c_s\Omega_{r,0} \), see the discussion in sec. \(^2.1\) for more details. For \( \mathcal{N} = 1 \) the expression in \((5)\) corresponds to the gravitational wave spectrum right after the transition to radiation domination.

The principal inputs for \texttt{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. The code \texttt{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 canonical scalar field \((c_s^2 = 1)\). The user can choose between these two options by setting the flag \texttt{cs_equal_one} to \texttt{False} or \texttt{True}, respectively. In the adiabatic perfect fluid case the value of \( w \) needs to be chosen in the interval \( 0 < w < 1 \) for the kernel in \((7)\) to be valid, and \texttt{SIGWfastEOS.py} only permits values in this range. To reproduce the results of \texttt{SIGWfast.py} for radiation domination, in \texttt{SIGWfastEOS.py} one would need to set \( c_s^2 = w = 1/3 \) (i.e. \( w = 1/3 \) and \texttt{cs_equal_one = False})\(^7\). Instead of providing \( k_{\text{rh}} \) as a further input, the script

\(^6\)Again, we ignore the contribution from the primordial trispectrum for reasons as explained in footnote\(^4\).

\(^7\)For \( c_s^2 = w = 1/3 \) the kernel \( T_w \) in \((7)\) exhibits divergent pieces, which one can show to cancel analytically, so that the kernel reduces to \( T_{\text{rad}} \) in \((2)\) as expected. To deal with these unphysical divergences numerically, when inputting a value \( w \) with \( |w - 1/3| < \epsilon \) in \texttt{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.
SIGWfastEOS.py computes \( \Omega_{\text{GW}} \) output of SIGWfastEOS.py: \( \Omega_{\text{GW}}^{\text{out}}(k) = \left( \frac{k_{\text{ref}}}{k_{\text{rh}}} \right)^{2b} \times \Omega_{\text{GW}}^{\text{in}}(k) \).

Here \( k_{\text{ref}} \) is the reference unit for wavenumbers \( k \) as chosen by the user.\(^8\) As the factor \( \left( \frac{k_{\text{ref}}}{k_{\text{rh}}} \right)^{2b} \) is a constant, its only effect is an overall rescaling which does not affect the spectral shape of \( \Omega_{\text{GW}}^{\text{in}} \) 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 corresponding values \( \Omega_{\text{GW}}^{\text{out}} \). In addition, both \( P_{\zeta}(k) \) and \( \Omega_{\text{GW}}(k) \) are plotted.

3 Prerequisites, installation and configuration

3.1 Prerequisites

SIGWfast can be used on any system that supports python 3. We recommend using python environments and a package manager such as ‘conda’. The following python modules are required:\(^9\)

\begin{align*}
\text{math, matplotlib, numpy, os, scipy, sys, time, tqdm.}
\end{align*}

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

\begin{align*}
\text{distutils, platform, shutil, subprocess,}
\end{align*}

and a working C++ compiler.

3.2 Installation

To use SIGWfast, download the latest release as a .zip or .tar.gz archive from [this page.\( \text{\textsuperscript{\textcolor{blue}{\small{\textsuperscript{\textendash}\text{\textsuperscript{\textsuperscript{\textendash}\textsuperscript{\textsuperscript{\textsuperscript{\textendash}}}\textendash}}}}}}\] After decompression the necessary files and directory structure are already in place in the parent directory.

3.3 File content

The parent directory 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 \( \Omega_{\text{GW}} \).
- `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 \( \Omega_{\text{GW}}(k) \) 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 \( P_{\zeta}(k) \) for the inflation model (11) is included.

\(^8\)The scale \( k_{\text{ref}} \) is defined in the sense that writing \( k = 0.1 \) in the python script corresponds to \( k = 0.1k_{\text{ref}} \).

\(^9\)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.
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`.

### 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 $P_{\Omega k}(k)$ 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!

### 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. 10. This is where you can adjust the script for your purposes. The necessary steps are as follows:

1. Set `filenameGW`. Choose a name for the .npz file that will contain the results for $\Omega_{gw}(k)$ and that will be deposited in the ‘data’ subdirectory. The $k$-values and corresponding $\Omega_{gw}$-values in this file will be accessible via the keywords ‘karray’ and ‘OmegaGW’. 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, $\Omega_{gw}$ 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, $\Omega_{gw}$ will be computed from a scalar power spectrum that needs to declared explicitly as a function $Pofk(k)$. See sec. 3.5 for detailed instructions on how this is to be done.

4. Optional: declare filenamePz. In case $\Omega_{gw}(k)$ 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 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\[11\]. 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 $\mathcal{N} = c_g \Omega_{r,0}$, see sec. 2 above.

7. Declare the range of wavenumbers $k$ for which $\Omega_{gw}(k)$ 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 $k_{min}$, an upper limit $k_{max}$ 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\[12\]. 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:

\[11\]The reason is that ‘sigwfast’ contains methods to perform in the integrations over $d$ and $s$ in (1) and (4), but is ignorant of the integrands.

\[12\]This is also good practice from the physics point-of-view, as one expects $\Omega_{gw}(k)$ and $\mathcal{P}_\zeta(k)$ to exhibit structure on roughly the same scales.
8. Set the value of $w$. In SIGWfastEOS.py we also need to specify 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 $w$.

9. Set the flag cs_equal_one. In SIGWfastEOS.py the computation of $\Omega_{gw}(k)$ 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 canonical scalar field ($c_s^2 = 1$), see sec. 2. By setting $\text{cs\_equal\_one} = \text{True}$ the computation is performed for the canonical scalar field case ($c_s^2 = 1$), while for the choice $\text{cs\_equal\_one} = \text{False}$ it is the adiabatic perfect fluid ($c_s^2 = w$) result that is computed. In the latter case the value of $w$ in the previous step has to be chosen in the range $0 < w < 1$.

### 3.6 How to input the primordial scalar power spectrum $P_\zeta(k)$

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 $P_\zeta(k)$. This can be provided as an analytical formula or in terms of numerical data:

- **Analytical formula** ($\text{Num\_Pofk} = \text{False}$): 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 $P_\zeta$. 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 [8] 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.\(^{13}\) 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.\(^{15}\) 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 $P_\zeta(k)$.

- **Numerical data** ($\text{Num\_Pofk} = \text{True}$): 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 $P_\zeta$-values, which should be accessible via the keywords ‘karray’ and ‘Pzeta’, respectively. That is, after loading the data in this file via the command $\text{Pdata} =$

\(^{13}\)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.

\(^{14}\)By default, $kpzeta$ ranges from $kmin/2$ to $2*kmax$ and contains $4x$ as many entries as $komega$.

\(^{15}\)Consider the following situation where the default definition of $kpzeta$ would lead to wrong results. Take the case where $P_\zeta(k)$ 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 $\Omega_{gw}(k)$, hence choosing $komega$ to only contain values $k \ll k_\star$. In this case $kpzeta$ should be defined independently from $komega$. 

Figure 2: Section of code where the analytical expression for $P_\zeta(k)$ can be defined as the function $P_\text{ofk}(k)$, 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 $\exp(-2\eta_\perp^2\kappa)$ compared to (10) to avoid excessively large values. In the lower part the default definition of the array $kpzeta$ is shown, over which $P_\zeta(k)$ is discretized in the computation.

The script is now 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 $P_\zeta(k)$ that exhibits further substructure in the form of oscillations. For a constant turn rate, the contribution to $P_\zeta(k)$ can be computed analytically and takes the form $[8,20]$:}

$$P_\zeta(k) = P_\text{env}(\kappa) \left[ 1 + (\kappa - 1) \cos \left( 2e^{-\frac{\kappa}{2}\eta_\perp^2} \right) + \sqrt{(2 - \kappa)\kappa} \sin \left( 2e^{-\frac{\kappa}{2}\eta_\perp^2} \right) \right] \times \Theta(2 - \kappa),$$  

with $P_\text{env}(\kappa) = P_0 \frac{2^{2(2-\kappa)\kappa} \eta_\perp^2}{4(2 - \kappa)\kappa} \delta^2$, and $\kappa = k/k_\star$, where $\Theta(x)$ denotes the Heaviside theta function and $k_\star$ corresponds to the locus where the ‘envelope’ $P_\text{env}(k)$ of the scalar power spectrum peaks. The other parameters are $(\delta)$, the duration of the turn.

---

16To make the arrays of $k$-values and $P_\zeta$-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.
for precisely this purpose. However, one can also use the corresponding scalar-induced gravitational wave spectrum $\Omega_{gw}(k)$ (right panel) as computed by $\text{SIGWfast.py}$ (or $\text{SIGWfastEOS.py}$ with $w = 1/3$ and $\text{cs\_equal\_one} = \text{False}$). The unit $k_{\text{ref}}$ corresponds to $k_\star$, the maximum of the envelope of $\mathcal{P}_\zeta(k)$. Using the python-only version ($\text{Use\_Cpp} = \text{False}$) the computation takes 1.2s with $\text{SIGWfast.py}$ and 2.8s with $\text{SIGWfastEOS.py}$ on the development machine (Macbook Pro with M1 CPU). Using the compiled C++ module ($\text{Use\_Cpp} = \text{True}$) the same computation takes 0.9s with $\text{SIGWfast.py}$ and 2.5s with $\text{SIGWfastEOS.py}$.

in $\eta$-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 $\text{SIGWfast.py}$ and $\text{SIGWfastEOS.py}$, 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 $k = 1$. This is done by multiplying (10) by an additional factor of $\exp(-2\eta_\perp \delta)$ [17]. The argument of $Pofk(k)$ corresponds to the variable $\kappa$ in (10), as we choose to measure $k$ in units of $k_\star$. The parameters ($\delta, \eta_\perp, \mathcal{P}_0$) are implemented as the global variables ($\text{delta, etap, P0}$).

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

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

4.1 Gravitational waves induced during radiation domination

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

We choose configuration settings as in fig. 1. We have set $\text{Num\_Pofk} = \text{False}$ to use the analytical formula for the scalar power spectrum. We wish to compute $\Omega_{gw}(k)$ for wavenumbers in the range $k \in [0.01 k_\star, 2.5 k_\star]$, i.e. setting $k_{\text{min}} = 0.01$ and $k_{\text{max}} = 2.5$, which encloses the maximum of $\mathcal{P}_{\text{env}}$. We then define the array $\text{omega}$ by covering this interval with $nk = 200$ points that are linearly spaced ($\text{numpy.linspace}$), which we expect to provide a sufficient resolution of the result. The ‘normalization’ factor $N$ in (1) or (3) has been set to unity ($\text{norm} = 1$) for simplicity.

When executing $\text{SIGWfast.py}$ (or $\text{SIGWfastEOS.py}$), the script computes $\Omega_{gw}(k)$ 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 $\Omega_{gw}(k)$, which we show in fig. 3. The following points are worth highlighting:

- The plot of $\mathcal{P}_\zeta(k)$ in the left panel is an interpolation of the scalar power spectrum evaluated on $\text{kpzeta}$. Here we use the standard definition of $\text{kpzeta}$, see footnote [14]. As this interpolation

---

17 This can be ‘undone’ in the end by multiplying the computed result for $\Omega_{gw}(k)$ by a factor $\exp(4\eta_\perp \delta)$. 

---
We observe that \texttt{SIGWfastEOS.py} is slower than \texttt{SIGWfast.py} by a constant offset of \(\sim 1.6\)s, 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 up to \(\sim 25\%\).

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|c|c|}
\hline
Number of points \(n_k\) computed & 200 & 400 & 1000 & 2000 \\
\hline
\(t_{\text{comp}}\) of \texttt{SIGWfast.py} (python-only version) & 1.2s & 2.5s & 6.5s & 13.6s \\
\hline
\(t_{\text{comp}}\) of \texttt{SIGWfast.py} (with C++ module) & 0.9s & 1.9s & 5.0s & 10.7s \\
\hline
\(t_{\text{comp}}\) of \texttt{SIGWfastEOS.py} (python-only version) & 2.8s & 4.1s & 8.1s & 15.3s \\
\hline
\(t_{\text{comp}}\) of \texttt{SIGWfastEOS.py} (with C++ module) & 2.5s & 3.5s & 6.7s & 12.4s \\
\hline
\end{tabular}
\caption{Comparison of computation times \(t_{\text{comp}}\) of \texttt{SIGWfast.py} and \texttt{SIGWfastEOS.py} (for the python-only version and the version using the compiled C++ module) for different values of \(n_k\), with the other configuration parameters as in fig. 1. In \texttt{SIGWfastEOS.py} we have further set \(w=1/3\) and \texttt{cs_equal_one=False} to replicate the results of \texttt{SIGWfastEOS.py}. Computations were done on the development machine (Macbook Pro with M1 CPU) for the standard example (11) as input for \(P_\zeta(k)\).}
\end{table}

\begin{itemize}
\item The plot of \(\Omega_{\text{GW}}(k)\) in the right panel of fig. 3 is the main output of SIGWfast, and the corresponding data has been saved in the file ‘\texttt{OmegaGW_of_k.npz}’ in the ‘data’ subdirectory. The spectrum \(\Omega_{\text{GW}}(k)\) exhibits the characteristic morphology of gravitational waves induced by scalar fluctuations with a peak in \(P_\zeta(k)\) at \(k = k_\ast\), i.e. a broad bump around \(k \sim 0.4k_\ast\) and principal peak around \(k \approx 2/\sqrt{3}k_\ast\). The most conspicuous property are however the oscillatory modulations of the principal peak, which are a direct consequence of the oscillations in \(P_\zeta(k)\). See [8] for a detailed analysis of the origin and the properties of these modulations in \(\Omega_{\text{GW}}(k)\). The detection prospects of such an oscillatory gravitational wave spectrum with the upcoming observatory LISA has been studied in [21].

\item Using the python-only version (\texttt{Use_Cpp = False}) the computation takes 1.2s with \texttt{SIGWfast.py} and 2.8s with \texttt{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 \texttt{Use_Cpp = True}, which is available for machines running on MacOS or Linux. Executing the python script \texttt{SIGWfast.py} or \texttt{SIGWfastEOS.py}, a module named ‘sigwfast’ is compiled, which takes \(\mathcal{O}(1)\) seconds. The computation of \(\Omega_{\text{GW}}(k)\) itself is then shortened to 0.9s with \texttt{SIGWfast.py} and 2.5s with \texttt{SIGWfastEOS.py}. In any further runs with flag setting \texttt{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 \(n_k\) computed is changed, see table 1.
\end{itemize}

We can also reproduce the results in fig. 3 using a numerical scalar power spectrum as input, by setting the flag \texttt{Num_Pofk = True}, in which case the data for \(P_\zeta(k)\) contained in the file ‘\texttt{data/P_of_k.npz}’ is used. In the released version of SIGWfast this file contains data for the standard example [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 $w$. 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 $P_\zeta(k)$ as given in (11). In fig. 4 we then show plots of $\Omega_{gw}(k)$ produced by `SIGWfastEOS.py` for $w=0.8$ and `cs_equal_one = False` (left panel) and `cs_equal_one = True` (right panel). On the development machine the computation times are $\sim3s$ (left panel plot) and $\sim1s$ (right panel plot), respectively. Recall that for $w \neq 1/3$ (i.e. $b \neq 0$) the output of `SIGWfastEOS.py` is not $\Omega_{gw}(k)$ today, but $\Omega_{gw}(k) \times (k_{ref}/k_{rh})^2b$, 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 $\Omega_{gw}(k)$ 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 $\Omega_{gw}(k)$ does not exhibit any visible oscillations. For further reading on this topic we refer readers to [22], where the effect of the expansion history of the universe on oscillatory signatures in $\Omega_{gw}(k)$ has been analysed in detail.

5 Troubleshooting

SIGWfast has been written for python 3 and will not work with python 2. 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. SIGWfast has also been tested on Ubuntu 20.04.4 and Windows Server 2019 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_Gpp = True` in the block of code titled ‘Configuration’ and its use leads to a reduction in computation times of up to 25%. This option is only available for systems running on Linux and

---

\[\text{We slightly adjusted the limits of } k_{omega} \text{ compare to those in the configuration in fig. 1 for better aesthetics.}\]

\[\text{See the following blog post on running python on Apple Silicon chips.}\]
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-main’ is located in the home directory ‘∼’. Hence, on the command line enter:

```
cd ~/SIGWfast-main/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:

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

We used the command python3 to make sure that python 3 is used, as the command python can sometimes refer to the version of python 2 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, a copy of which is included in the release. In case you have not received the license statement, see [https://spdx.org/licenses/MIT.html](https://spdx.org/licenses/MIT.html).

Acknowledgements

We are indebted to 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 Sébastien Renaux-Petel, whose scientific insights and skilled leadership of the research group made the development of SIGWfast possible. Special thanks go to Guillem Domènech, whose research provided the scientific basis for SIGWfastEOS.py. We also thank John W. Ronayne, whose immense knowledge of python helped get this project off the ground. During the development of SIGWfast, L.T.W was supported by the European Research Council under the European Union’s Horizon 2020 research and innovation programme (grant agreement No 758792, project GEODESI).

A Ferrers functions and associated Legendre function of the second kind

Here we collect the definitions of $P_\mu^\nu(x)$, $Q_\mu^\nu(x)$ and $Q_\mu^\nu(x)$ that appear in [7], as can be found in [23].

\[
P_\mu^\nu(x) = \left( \frac{1 + x}{1 - x} \right)^{\mu/2} \frac{1}{\Gamma[1 - \mu]} F\left( \nu + 1, -\nu; 1 - \mu; \frac{1}{2}(1 - x) \right),
\]

(12)

\[
Q_\mu^\nu(x) = \frac{\pi}{2 \sin(\pi\mu)} \left[ \cos(\pi\mu) \left( \frac{1 + x}{1 - x} \right)^{\mu/2} \frac{1}{\Gamma[1 - \mu]} F\left( \nu + 1, -\nu; 1 - \mu; \frac{1}{2}(1 - x) \right),
\]

\[
-\left( \frac{1 - x}{1 + x} \right)^{\mu/2} \frac{\Gamma[\nu + \mu + 1]}{\Gamma[\nu - \mu + 1]\Gamma[1 + \mu]} F\left( \nu + 1, -\nu; 1 + \mu; \frac{1}{2}(1 - x) \right) \right],
\]

(13)

\[
Q_\mu^\nu(x) = \frac{\pi^{1/2}(2x^2 - 1)^{\mu/2}}{2^{\nu+1}x^{\nu+\mu+1}} \frac{1}{\Gamma[\nu + 3/2]} F\left( \frac{1}{2}\nu + 1, \frac{1}{2}\mu + 1, \frac{1}{2}\nu + \frac{1}{2}\mu + \frac{1}{2}; \frac{1}{2}; \frac{1}{2}x^2 \right),
\]

(14)

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

20If using python environments, ensure the correct environment is activated.
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.

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

[4] J. Fumagalli, G.A. Palma, S. Renaux-Petel, S. Sypsas, L.T. Witkowski and C. Zenteno, *Primordial gravitational waves from excited states*, JHEP 03 (2022) 196 [2111.14664].

[5] 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].

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

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

[8] 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].

[9] J. Fumagalli, S.e. Renaux-Petel and L.T. Witkowski, *Resonant features in the stochastic gravitational wave background*, JCAP 08 (2021) 059 [2105.06481].

[10] 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].

[11] 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].

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

[13] 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].

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

[15] 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].

[16] 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].

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

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

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

[20] G.A. Palma, S. Sypsas and C. Zenteno, *Seeding primordial black holes in multifield inflation*, Phys. Rev. Lett. 125 (2020) 121301 [2004.06106].
[21] J. Fumagalli, M. Pieroni, S. Renaux-Petel and L.T. Witkowski, *Detecting primordial features with LISA*, *JCAP* **07** (2022) 020 [2112.06903].

[22] 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].

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