# NetSSE

*Release latest*

**REGMO**

**Sep 04, 2024**

# CONTENTS

CHAPTER

ONE

# NETSSE USER GUIDE

## 1.1 Contents

### 1.1.1 Introduction

#### Background

> *— When an Adequate Sampling of the Oceans Would Benefit Marine Industries, Coastal Communities, and Science.*

Many technologies have been developed to estimate sea states, traditionally applied on an individual basis; see Fig. 1.1. Schematically, the four main sources of wave data could be categorised as:

1. **In-situ measurements** from fixed or floating systems, which comprise moored and drifting wave-buoys, as well as capacitance wave probes and pressure gauges;
2. **Remote sensing data** from satellites and aircraft, using altimeters or synthetic aperture radars (SAR);
3. Spatiotemporal wave inference from X-band (or microwave) **marine radars**[1], which equip more and more ships nowadays, along with the new LIDAR technology[2][3];
4. Data generated by third-generation **ocean wind-wave spectral models**.

In spite of the many available means for **sea state estimation** (*SSE*), the scarcity of in-situ data from vast areas of the world's oceans is an ongoing problem[10]. It appears sensible to view the many different SSE methods as complementary and combinable, rather than competing with each other. Overall, there is a growing interest from the ocean engineering and science communities in leveraging the complementary abilities of individual platforms in multidisciplinary research analysing multiple types of wave observations in an integrated manner[11].

**Ship-as-a-wave-buoy** (*SAWB*), which could be considered as being part of the first category in the above list of SSE means, is a concept and class of SSE methods which have received a lot of interest in maritime research. The principle

[1] Borge, J. C. N., Reichert, K., and Dittmer, J. Use of nautical radar as a wave monitoring instrument. *Coastal Engineering 37*, 3-4 (1999), 331–342.

[2] Fu, T. C., Fullerton, A. M., Hackett, E. E., and Merrill, C. Shipboard measurement of ocean waves. In *Proceedings of the International Conference on Offshore Mechanics and Arctic Engineering* (2011), vol. 6, American Society of Mechanical Engineers, pp. 699–706.

[3] Kabel, T., Georgakis, C. T., and Zeeberg, A. R. Mapping ocean waves using new LIDAR equipment. In *Proceedings of the International Offshore and Polar Engineering Conference* (2019), vol. 3, International Society of Offshore and Polar Engineers, pp. 2558–2564.

[5] Mounet, R. E. G. *PhD Defense - Sea state estimation based on measurements from multiple observation platforms*. Presentation at the Technical University of Denmark, January 2024.

[10] Smith, S. R., Alory, G., Andersson, A., Asher, W., Baker, A., Berry, D. I., Drushka, K., Figurskey, D., Freeman, E., Holthus, P., Jickells, T., Kleta, H., Kent, E. C., Kolodziejczyk, N., Kramp, M., Loh, Z., Poli, P., Schuster, U., Steventon, E., Swart, S., Tarasova, O., De La Villéon, L. P., and Shiffer, N. V. Ship-based contributions to global ocean, weather, and climate observing systems. *Frontiers in Marine Science 6* (2019), 434.

[11] Villas Bôas, A. B., Ardhuin, F., Ayet, A., Bourassa, M. A., Brandt, P., Chapron, B., Cornuelle, B. D., Farrar, J. T., Fewings, M. R., Fox-Kemper, B., Gille, S. T., Gommenginger, C., Heimbach, P., Hell, M. C., Li, Q., Mazloff, M. R., Merrifield, S. T., Mouche, A., Rio, M. H., Rodriguez, E., Shutler, J. D., Subramanian, A. C., Terrill, E. J., Tsamados, M., Ubelmann, C., and van Sebille, E. Integrated observations of global surface winds, currents, and waves: Requirements and challenges for the next decade. *Frontiers in Marine Science 6* (2019), 425.

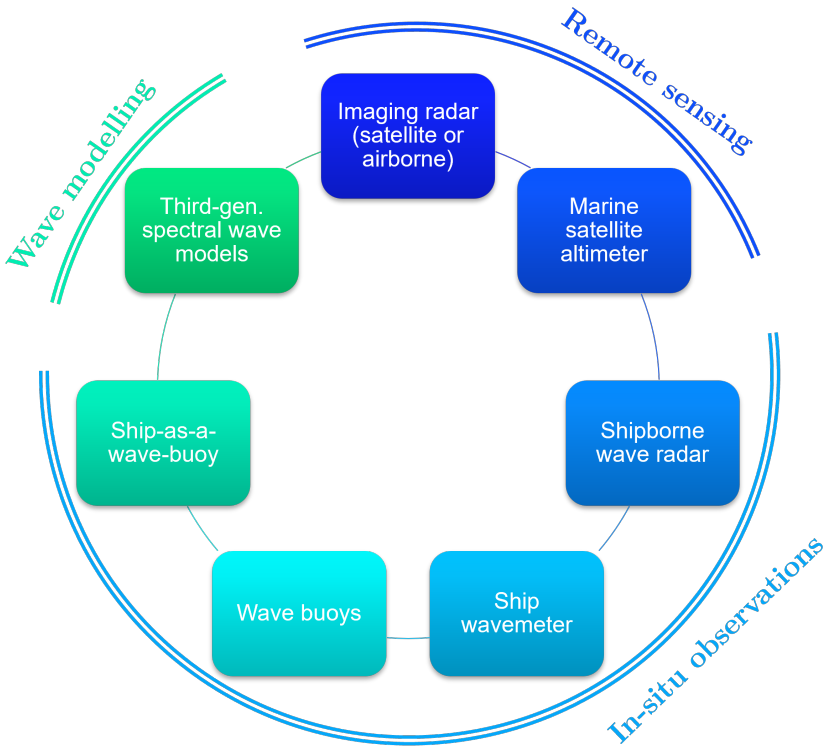


Fig. 1.1: **Overview of the most widely used technologies for sea state estimation.** Adapted from Mounet[Page 1, 5].

is rather simple at first sight: a vessel is considered as a wave-riding buoy, experiencing waves in the seaway, thus inducing motions and other responses of the platform. The so-called **wave buoy analogy** (*WBA*) suggests the idea that the vessel responses can reveal some valuable information about the waves that have induced those responses, just like the motions of a wave-riding buoy that can be processed to eventually provide a complete description of the sea state. Most vessels usually have various sensors installed on board to measure their responses. Those sensors help monitor the vessel"s performance, stability, and structural strength. Many studies have demonstrated that such measurements can also be used to derive a cost-efficient and accurate estimate of the encountered wave conditions. Nielsen[6] made a comprehensive review of available WBA methods.

## Vision and aims

> — *When Quantity Is a Quality of Its Own.*

The idea of a **network of wave sensors** integrating in-situ observations from multiple vessels has emerged in the scientific literature, introduced, for instance, by Nielsen[7][8] and Nielsen *et al.*[9], as a concept in which individual means for *SSE* are considered as complementary and used collaboratively. Fusing wave data from multiple, heterogeneous observation platforms has the potential to increase the accuracy of sea state estimates, by mitigating the uncertainties inherent to measurements from the individual platforms.

[6] Nielsen, U. D. A concise account of techniques available for shipboard sea state estimation. *Ocean Engineering 129* (2017), 352–362.

[7] Nielsen, U. D. *Sea state estimation based on measurements of wave-induced ship responses*. Technical University of Denmark, Department of Mechanical Engineering, Kongens Lyngby, Denmark, December 2018. Doctoral dissertation for the degree of Doctor Technices.

[8] Nielsen, U. D. Sea state estimation using simultaneous data from multiple observation platforms. In *Proceedings of JASNAOE's Autumn Conference 2018* (2018), The Japan Society of Naval Architects and Ocean Engineers, pp. 37–40.

[9] Nielsen, U. D., Brodtkorb, A. H., and Sørensen, A. J. Sea state estimation using multiple ships simultaneously as sailing wave buoys. *Applied Ocean Research 83* (2019), 65–76.

Considering the sheer number of ships in transit on the open sea, there is a true interest and prospect of using wave information derived from station-kept – as well as advancing – vessels in a network-based approach. The maritime industry evolves nowadays in an era of digitalisation, aimed at increasing safety and energy efficiency. A large portion of the world fleet of merchant ships has installed sensors to permanently monitor the vessel responses. A tremendous amount of data is being collected in ship operations and remains to be exploited for many different purposes. In this scenario, the *WBA* method shows high relevance. Combined with data from other observation platforms, accurate and cost-efficient ship-based observations are a vital input to a networked procedure to continuously map the wave systems around the globe with increased confidence[4].

A network-based approach facilitates collaboration between adjacent vessels, to bring more timely information to decision support systems and (possibly remote) operators. The technology for communication of data at sea is improving, enabling the sharing of information about the sea state within fleets of ships. Beyond conventional ships, this is also relevant for the operation and control of small and autonomous surface vehicles (ASVs), for which situational awareness with regard to the wave environment is crucial[4].

**NetSSE** was created by researchers to test and distribute engineering tools towards the development of network-based solutions for better-quality sea state estimates on both a local on-site position and at a regional scale. Here, "better-quality" refers to estimates with overall decreased uncertainty with respect to state-of-the-art methods, for instance, higher accuracy, precision, robustness, reliability, and availability. It was decided to have the implementations based on the Python language, which is an open-source, data science programming language widely used in the world nowadays.

#### References

### 1.1.2 Fundamental theoretical concepts

This section provides the basics and fundamental theory involved in NetSSE, including a brief review of the relevant literature. The text and illustrations are copied and adapted from Mounet[42].

#### Modelling the ocean wave environment

> — *Somewhere in Between Pure Randomness and Physical Determinism.*

Nonlinear dynamical processes govern the generation of real waves through ocean surface stress and their evolution from wind waves to swell. In a wave system generated by wind, like in the oceans, the heights of successive waves vary; the waves are said to be **irregular**. Moreover, the elevation along the crests also varies, hence the **short-crestedness** of real waves.

When it is sampled over time at a fixed-point location, the surface elevation behaves as a stochastic (or random) process. The exact shape of the ocean surface cannot be forecast with full certainty. Such **aleatory uncertainty** is intrinsic (because it is natural and physical) and cannot be reduced or eliminated[34]. Nonetheless, it is generally assumed that, over a limited period of time, typically thirty minutes to one hour, and over a restricted portion of space, the surface waves can be described by a stationary and ergodic process called the **sea state**, e.g., Lewis[26] (Chapter 8, Section 2). This assumption – sometimes referred to as the decoupling approach – means that, at a local scale, it is possible to estimate the short-term statistical properties of the wave process from a single recording of the wave elevation at a point location.

[4] Mounet, R. E. G. *Sea state estimation based on measurements from multiple observation platforms*. Ph.D. dissertation, Technical University of Denmark, 2023.

[42] Mounet, R. E. G. *Sea state estimation based on measurements from multiple observation platforms*. Ph.D. dissertation, Technical University of Denmark, 2023.

[3] Bitner-Gregersen, E. M., Bhattacharya, S. K., Chatjigeorgiou, I. K., Eames, I., Ellermann, K., Ewans, K., Hermanski, G., Johnson, M. C., Ma, N., Maisondieu, C., Nilva, A., Rychlik, I., and Waseda, T. Recent developments of ocean environmental description with focus on uncertainties. *Ocean Engineering 86* (2014), 26–46.

[4] Bitner-Gregersen, E. M., Ewans, K. C., and Johnson, M. C. Some uncertainties associated with wind and wave description and their importance for engineering applications. *Ocean Engineering 86* (2014), 11–25.

[26] Lewis, E. V., Ed. *Principles of Naval Architecture: Motions in Waves and Controllability. Volume III. Second Revision.* The Society of Naval Architects and Marine Engineers, Jersey City, NJ, USA, 1989.

The linear wave theory assumes that the sea surface elevation $\zeta(x_0, y_0, t)$ about the mean sea level (*MSL*) is a random linear superposition of a large number of non-interacting regular wave components, all travelling independently of one another in different directions at different frequencies[44]:

$$\zeta(x_0, y_0, t) = \sum_n a_n \sin\left(k_n(x_0 \cos\mu_n + y_0 \sin\mu_n) - \omega_{0,n} t + \phi_n\right), \tag{1.1}$$

where $t$ is the time and $(x_0, y_0)$ are the spatial coordinates in an Earth-fixed reference system. The wave components are densely distributed in frequency and direction. For the $n$-th component, $a_n$ is the amplitude, $\mu_n$ is the direction of travel, $\phi_n \in [-\pi, \pi)$ is a random phase angle, $\omega_{0,n}$ is the angular frequency ($\omega_{0,n} = 2\pi f_{0,n}$) satisfying the dispersion relationship,

$$\omega_{0,n}^2 = g k_n \tanh k_n h, \tag{1.2}$$

with $g$ the gravitational acceleration, $k_n = 2\pi/\lambda_n$ the wave number, $\lambda_n$ the corresponding wavelength, and $h$ the water depth, as illustrated in Fig. 1.2. In deep water conditions ($h \gtrsim \lambda_n/2$), the dispersion relation can be simplified to $\omega_{0,n}^2 = g k_n$.

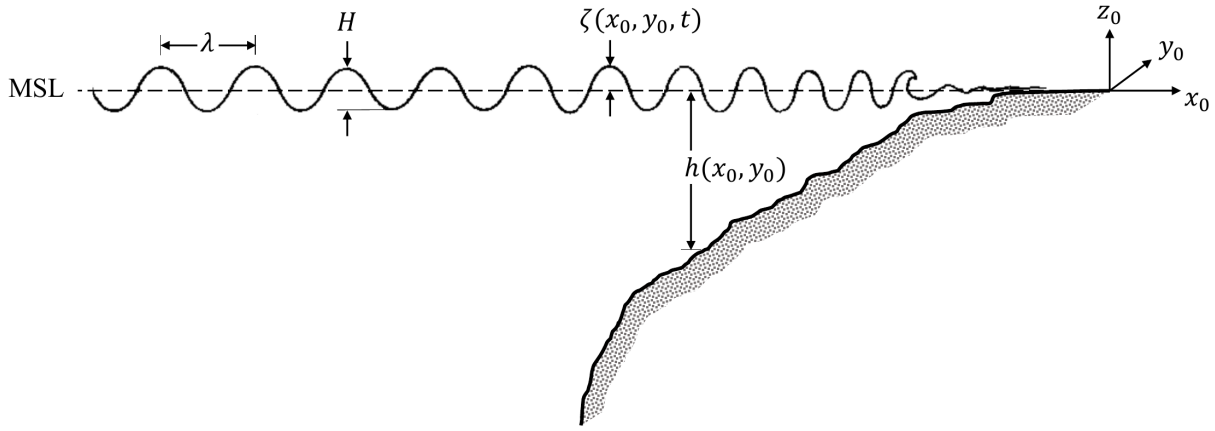


Fig. 1.2: **Illustration of the basic wave parameters to describe the sea surface elevation in the time domain.**

Since the component sinusoids in Eq. (1.1) have random phases, the total variance $\sigma_\zeta$ of the sea surface elevation equals the sum of the variances of its component wave trains[48]:

$$\sigma_\zeta \equiv \mathbb{E}[\zeta^2] = \sum_n \frac{a_n^2}{2}, \tag{1.3}$$

where the dependency on spatial coordinates $(x_0, y_0)$ has been dropped to consider the measurement at an arbitrary point and the dependency on time is avoided due to stationarity.

If the output of a wave recorder is filtered to select only those components on the right-hand side of Eq. (1.1) with frequencies in the range $\omega_0 - \delta\omega_0/2$ to $\omega_0 + \delta\omega_0/2$, giving a variance $\delta\sigma$, then a function $E(\omega_0)$ can be defined by[48]:

$$E(\omega_0) \equiv \lim_{\delta\omega_0 \to 0} \frac{\delta\sigma}{\delta\omega_0}. \tag{1.4}$$

$E(\omega_0)$ is a spectral density function and is sometimes called the **wave variance spectrum**, or simply, the wave spectrum.

[44] St Denis, M., and Pierson, W. J. *On the motions of ships in confused seas*, vol. 61. The Society of Naval Architects and Marine Engineers, New York, NY, USA, November 1953.

[48] Tucker, M. J., and Pitt, E. G. *Waves in ocean engineering*. Volume 5 in Elsevier Ocean Engineering Book Series. Elsevier, 2001.

If one can now filter not only in frequency, but also select those wavetrains whose direction of travel is between $\mu - \delta\mu/2$ and $\mu + \delta\mu/2$, giving a variance $\delta\sigma$, then by analogy with Eq. (1.4):

$$E(\omega_0, \mu) \equiv \lim_{\delta\omega_0, \delta\mu \to 0} \frac{\delta\sigma}{\delta\omega_0 \, \delta\mu} = \frac{\mathrm{d}^2\sigma}{\mathrm{d}\omega_0 \, \mathrm{d}\mu}. \tag{1.5}$$

The **directional wave spectrum** (*DWS*) $E(\omega_0, \mu)$ characterises how the wave power spectral density (*PSD*) is distributed across frequencies $\omega_0$ – or, equivalently, wave numbers $k$ – and directions of travel $\mu$. With an additional assumption about a Gaussian surface elevation, the DWS fully describes the stochastic properties of linear waves[24].

The DWS is the fundamental quantity of wave modelling and the quantity that allows calculating the consequences of interactions between waves and other matter[16]. Phase-averaged spectral wave models compute the development of the DWS, modelling the growth, transformation, and decay of ocean waves due to nonlinear wave-wave interactions and interactions with ocean surface winds and bathymetry. Rapid phase oscillations are not modelled, but instead, the slow evolution of the energy in each wave component is computed[22]. The **energy balance equation** describes the evolution of the DWS $E(\omega_0, \mu)$:

$$\frac{\partial E}{\partial t} + \frac{\partial}{\partial x_0}(c_{x_0} E) + \frac{\partial}{\partial y_0}(c_{y_0} E) + \frac{\partial}{\partial \omega_0}(c_{\omega_0} E) + \frac{\partial}{\partial \mu}(c_\mu E) = S_{\text{tot}}, \tag{1.6}$$

where $t$ is time, $c_{x_0}$ and $c_{y_0}$ are the propagation velocities in the geographical space $(x_0, y_0)$, while $c_{\omega_0}$ and $c_\mu$ are the propagation velocities in the spectral space[19]. Currents are neglected here. The left-hand side describes the propagation through a nonhomogeneous medium (i.e., in variable water depth), which conserves the total wave energy as an adiabatic invariant. The first term represents the local rate of change of energy density in time. The second and third terms describe the geographic propagation of PSD in the $(x_0, y_0)$ plane. The fourth term represents the shifting of the frequency due to depth variations, and the fifth term is related to depth-induced refraction with propagation velocity $c_\mu$ in the directional $\mu$-space. The right-hand side consists of source terms expressed as:

$$S_{\text{tot}} = S_{\text{in}} + S_{\text{ds}} + S_{\text{bot}} + S_{\text{surf}} + S_{\text{nl}}. \tag{1.7}$$

The source terms describe the wave generation from wind input $S_{\text{in}}$, the energy sinks – such as whitecapping dissipation $S_{\text{ds}}$, bottom friction $S_{\text{bot}}$, and depth-induced wave breaking $S_{\text{surf}}$ –, and a nonlinear wave-wave interaction term $S_{\text{nl}}$.

The **omnidirectional wave spectrum**, or, **point wave spectrum**, to distinguish it from the DWS, can be obtained by integration of the DWS over wave directions,

$$E(\omega_0) = \int_{-\pi}^{\pi} E(\omega_0, \mu) \, \mathrm{d}\mu. \tag{1.8}$$

Reciprocally, the DWS can be decomposed as the product of the point spectrum and a so-called **directional spreading function** (*DSF*) $D(\omega_0, \mu)$,

$$E(\omega_0, \mu) = E(\omega_0) \cdot D(\omega_0, \mu), \tag{1.9}$$

such that $D(\omega_0, \mu)$ is positive and the integral over directions $\int_{-\pi}^{\pi} D(\omega_0, \mu) \, \mathrm{d}\mu$ equals 1 for any frequency $\omega_0$.

A standard functional form for the DSF was found by Mitsuyasu *et al.*[29]:

$$D(\mu|\omega_0) = D_0 \cos^{2s}\left(\frac{\mu - \mu_0}{2}\right), \tag{1.10}$$

[24] Krogstad, H. E., and Trulsen, K. Interpretations and observations of ocean wave spectra. *Ocean Dynamics 60*, 4 (2010), 973–991.

[16] Hauser, D., Kahma, K., Krogstad, H. E., Lehner, S., Monbaliu, J. A. J., and Wyatt, L. R. Measuring and analysing the directional spectra of ocean waves. EU COST Action 714, 2005.

[22] Komen, G. J., Cavaleri, L., Donelan, M., Hasselmann, K., Hasselmann, S., and Janssen, P. A. E. M. *Dynamics and modelling of ocean waves*. Cambridge Univ. Press, Cambridge, UK, 1996.

[19] Joensen, B., Niclasen, B. A., and Bingham, H. B. Wave power assessment in Faroese waters using an oceanic to nearshore scale spectral wave model. *Energy 235* (2021), 121404.

[29] Mitsuyasu, H., Tasai, F., Suhara, T., Mizuno, S., Ohkusu, M., Honda, T., and Rikiishi, K. Observations of the directional spectrum of ocean waves using a cloverleaf buoy. *Journal of Physical Oceanography 5*, 4 (1975), 750–760.

where $\mu_0$ denotes the principal wave direction at the specific frequency $\omega_0$, $s$ is a directional spreading parameter, and $D_0$ is a normalising constant introduced to satisfy the requirement on the integral of $D$; i.e.,

$$D_0 \equiv \left[ \int_{\mu_{\min}}^{\mu_{\max}} \cos^{2s}\left( \frac{\mu - \mu_0}{2} \right) \right]^{-1}. \tag{1.11}$$

If the widest possible range of directions is considered for the DSF, such that $\mu_{\min} = -\pi$ and $\mu_{\max} = \pi$, then the constant $D_0$ becomes

$$D_0 = \frac{2^{2s-1}}{\pi} \frac{\Gamma^2(s+1)}{\Gamma(2s+1)}, \tag{1.12}$$

where $\Gamma(\cdot)$ denotes the Gamma function.

Commonly used sea state parameters are computed from the DWS[17] and listed in Table 1.1. Some of them depend on the $n$-th order spectral moments, which are defined as:

$$m_n \equiv \int_0^\infty \omega_0^n \, E(\omega) \, \mathrm{d}\omega_0, \quad n \in \{-1\} \cup \mathbb{N}. \tag{1.13}$$

Table 1.1: Common definitions of the sea state parameters derived from the wave spectrum[17].

| Symbol | Description | Mathematical definition | Unit |
|---|---|---|---|
| $H_s$ | Significant wave height | $4\sqrt{m_0}$ | m |
| $T_E$ | Mean energy period | $2\pi(m_{-1}/m_0)$ | s |
| $T_{m01}$ | Mean wave period | $2\pi(m_0/m_1)$ | s |
| $T_p$ | Peak wave period | $2\pi/(\mathrm{argmax}_{\omega_0}[E(\omega_0)])$ | s |
| $T_z$ | Mean zero up-crossing period | $2\pi\sqrt{m_0/m_2}$ | s |
| $\mu_m$ | Overall mean wave direction | $\arctan(d/c)$ with $d = \int_{-\pi}^{\pi}\int_0^\infty E(\omega_0,\mu)\sin(\mu)\,\mathrm{d}\omega_0\,\mathrm{d}\mu$ and $c = \int_{-\pi}^{\pi}\int_0^\infty E(\omega_0,\mu)\cos(\mu)\,\mathrm{d}\omega_0\,\mathrm{d}\mu$ | rad |
| $\mu_p$ | Peak wave direction | $\mathrm{argmax}_{\mu}[E(\omega_0,\mu)]$ | rad |

For the purpose of providing a better description of the sea state, the reduced set of sea state parameters from Table 1.1 can be extended with similar parameters characterising each of the few wave systems that compose the DWS. Spectral partitioning methods[41] are used in dividing the PSD of the DWS into several clusters centred around common spectral peaks. This approach is particularly useful to analyse multimodal wave spectra involving one or several swell systems, as well as a possible wind sea system. The sea state description results in a list of parameters such as $H_{s,\mathrm{wind}}$, $H_{s,\mathrm{swell1}}$, $H_{s,\mathrm{swell2}}$, …, $T_{p,\mathrm{wind}}$, $T_{p,\mathrm{swell1}}$, etc.

In many engineering applications, such as the response analysis of marine structures exposed to waves, use of the wave spectrum is required in order to carry on the computations. In those cases, the sea state parameters can be exploited to reconstruct the wave spectrum by applying an adequately chosen parameterised spectrum. There exist several standard forms of wave spectra. The choice of one form is application-specific and made based on the geographical area of study. Among other effects, the water depth, the predominance of wind sea over swell, or the opposite, and the **fetch** – defined as the length of the body of water that the wind blows over – should be considered.

[17] IAHR. List of sea-state parameters. *Journal of Waterway, Port, Coastal, and Ocean Engineering 115*, 6 (1989), 793–808. IAHR Working Group on Wave Generation and Analysis.

[41] Portilla, J., Ocampo-Torres, F. J., and Monbaliu, J. Spectral partitioning and identification of wind sea and swell. *Journal of Atmospheric and Oceanic Technology 26*, 1 (2009), 107–122.

For example, the spectra of fully-developed wind waves in the ocean can be approximated by the Pierson and Moskowitz (PM) standard spectrum[40], expressed in terms of the wave height and period as:

$$E_{\text{PM}}(\omega_0) = \frac{5}{16}\, H_s^2\, \omega_p^4\, \omega_0^{-5} \exp\left(-\frac{5}{4}\left(\frac{\omega_0}{\omega_p}\right)^{-4}\right), \tag{1.14}$$

where $\omega_p = 2\pi/T_p$ is the angular spectral peak frequency.

Wind waves rapidly developed by a strong wind in a relatively restricted water body (i.e., a fetch-limited situation) usually feature a much sharper spectral peak than that given by Eq. (1.14)[12]. The Joint North Sea Wave Project (*JONSWAP*)[15] addressed this scenario of a developing sea state in a dedicated observation program, which resulted in the proposal of another spectral form. In its generalised version, the JONSWAP spectrum is formulated as a modification of the PM spectrum:

$$E_{\text{J}}(\omega_0) = A_\gamma\, E_{\text{PM}}(\omega_0)\, \gamma^{\exp\left(-0.5\left(\frac{\omega_0-\omega_p}{\sigma(\omega_0)\,\omega_p}\right)^2\right)}, \tag{1.15}$$

where $A_\gamma = 1 - 0.287 \ln(\gamma)$ is a normalising factor, $\gamma$ is called the **peak enhancement factor** – which controls the sharpness of the spectral peak – with a standard value of 3.3, and $\sigma$ is a spectral width parameter, taken as

$$\sigma(\omega_0) = \begin{cases} \sigma_a \simeq 0.07 & \text{for } \omega_0 \leq \omega_p, \\ \sigma_b \simeq 0.09 & \text{for } \omega_0 > \omega_p. \end{cases}$$

For $\gamma = 1$, the JONSWAP spectrum reduces to the PM spectrum.

When the respective heights and periods of the wind sea and swell systems are available, the DWS of the resultant sea state can be estimated by linearly superimposing the parameterised (point) wave spectra computed for each system in combination with a DSF[12]. Yet, such an approach is problematic in certain conditions because the observed DWS may differ significantly from the reconstructed spectrum. Instead, a much better description of the directional spreading in the sea state is accomplished by the Fourier coefficients of the DSF[2], defined as:

$$a_n(\omega_0) \equiv \int_{-\pi}^{\pi} D(\omega_0, \mu) \cos(n\mu)\, \mathrm{d}\mu,$$
$$b_n(\omega_0) \equiv \int_{-\pi}^{\pi} D(\omega_0, \mu) \sin(n\mu)\, \mathrm{d}\mu.$$

In the general short-crested case, the DSF is a continuous function of $\mu$ over $[0, 2\pi]$, satisfying $D(\omega_0, 0) = D(\omega_0, 2\pi)$. It is therefore possible to write its Fourier series decomposition as:

$$D(\omega_0, \mu) = \frac{1}{2\pi} + \frac{1}{\pi} \sum_{n=1}^{\infty} \{a_n(\omega_0) \cos(n\mu) + b_n(\omega_0) \sin(n\mu)\}. \tag{1.16}$$

The so-called "First 5" spectral wave parameters consist of the combination of $E(\omega_0)$, $a_1(\omega_0)$, $b_1(\omega_0)$, $a_2(\omega_0)$, and $b_2(\omega_0)$. This set of parameters contains sufficient information to estimate the DWS with good accuracy.

[40] Pierson, W. J., and Moskowitz, L. Proposed spectral form for fully developed wind seas based on similarity theory of S. A. Kitaigorodskii. *Journal of Geophysical Research 69*, 24 (1964), 5181–5190.

[12] Goda, Y. *Random seas and design of maritime structures*, 3 ed., vol. 33 of *Advanced Series on Ocean Engineering*. World Scientific Publishing Co. Pte. Ltd., 2010.

[15] Hasselmann, K., Barnett, T., Bouws, E., Carlson, H., Cartwright, D., Enke, K., Ewing, J., Gienapp, A., Hasselmann, D., Kruseman, P., Meerburg, A., Müller, P., Olbers, D., Richter, K., Sell, W., and Walden, H. Measurements of wind-wave growth and swell decay during the Joint North Sea Wave Project (JONSWAP). *Ergänzungsheft Zur Deutschen Hydrographischen Zeitschrift, Reihe A*, 12 (1973).

[2] Benoit, M., Frigaard, P., and Schäffer, H. A. Analysing multidirectional wave spectra: A tentative classification of available methods. In *Proceedings of the 27th IAHR Congress, San Francisco, CA, USA* (1997), E. Mansard, Ed., Canadian Government Publishing, pp. 131–158.

### Wave-induced response of surface-floating marine vessels

*— How Waves Set Vessels in Motion.*

In this section, the term "surface-floating marine vessel" is used in a general meaning, including all kinds of man-made floating structures able to be moved forward over the sea surface by means of an onboard propulsion and steering system. This definition therefore encompasses cargo ships, tankers, passenger vessels, research vessels, fishing boats, pleasure crafts, etc., but also even smaller surface vehicles like wave gliders and unmanned surface vehicles (*USV*).

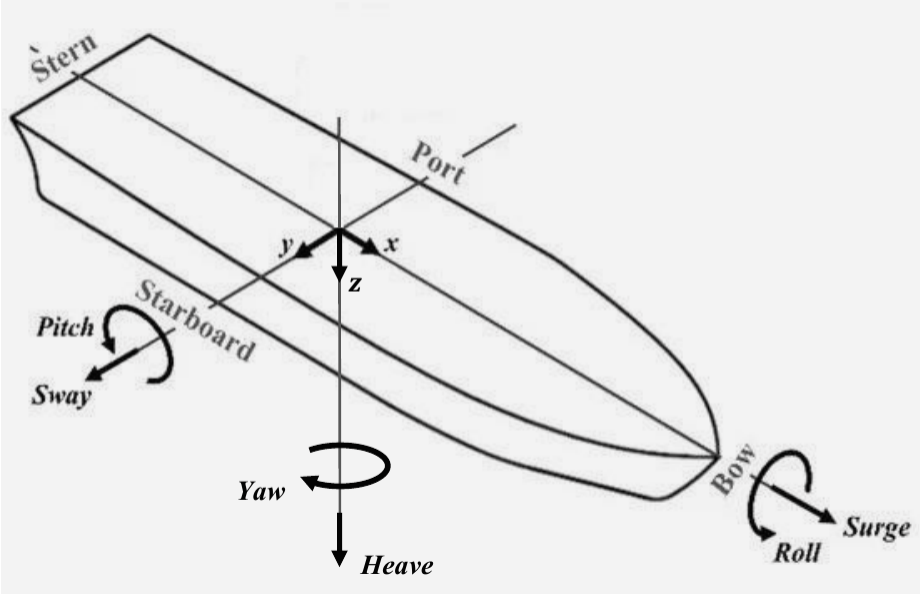


Fig. 1.3: **The six-DoF rigid-body motions of a vessel**. A body-fixed coordinate system is used, with the $z$-axis pointing towards the bottom of the vessel.

When sea waves encounter a floating vessel, they induce mechanical responses of the structure, such as rigid-body motions in all six degrees of freedom (*DoF*) as depicted in Fig. 1.3, vertical and horizontal bending moments, etc. Sensors placed onboard the vessel facilitate time records of these oscillating responses. The digital signals can be processed in the frequency domain by means of a Fast Fourier Transform (*FFT*) (or similar method), which yields the so-called **measured response spectra** $\tilde{S}_{RR}$ — in which notation the tilde emphasises that the quantity is derived from measurements. Fluctuations of the ship heading in an Earth-fixed reference system should be small enough over the duration of the time window that is used to compute the response spectra (to ensure stationary conditions). Apart from a dynamic positioning (*DP*) context, this assumption is most of the time valid for a 20-minute time window during transoceanic voyages of large in-service ships (e.g., container ships). In such scenarios, the vessel speed should also vary as little as possible during the time window.

For a vessel sailing at constant forward speed $U$ and compass heading (or yaw angle) $\psi$, the wave-induced vessel responses oscillate at a frequency that differs from the (intrinsic) wave frequency due to the **Doppler shift**. The **encounter frequency** $\omega_e = 2\pi f_e$ is related to the intrinsic wave frequency as expressed in Eq. (1.17):

$$\omega_e = \left|\omega_0 - \omega_0^2 \tau\right|, \tag{1.17}$$

where $\tau = \frac{U}{g} \cos\beta$ can be interpreted as the Doppler shift's intensity and $\beta$ is the direction of wave encounter relative to the vessel's centreline in a North-East-Down (*NED*) reference frame. If $\mu$ is defined as the direction where the waves are **travelling from** (nautical convention), e.g., relative to the geographical North, then $\beta$ is related to $\mu$ by the expression:

$$\beta = \pi + \mu - \psi. \tag{1.18}$$

### Visualise the Doppler shift

This online tool was created to visualise the effect of the Doppler shift in wave frequencies for a user-defined vessel speed and wave encounter angle. The right-hand side illustrates the mapping between intrinsic wave frequencies ($x$-axis) and encounter frequencies ($y$-axis) when waves are observed from a ship with forward speed $U$ at an encounter angle $\beta$. The parameters are user-defined on the left-hand side of the panel.

Considering a given encounter frequency, the mapping of Eq. (1.17) can result in **up to three** distinct intrinsic frequencies denoted $\omega_{0,1}$, $\omega_{0,2}$, and $\omega_{0,3}$, i.e., three different wave frequencies that will excite the vessel responses at the same excitation frequency $\omega_e$. The triple-value feature of the Doppler shift results in complications when transforming spectra between encounter and intrinsic frequency domains. The energy is conserved in the wave spectra when transforming from intrinsic frequency to encounter frequency domain, and vice-versa. The **encounter-domain wave spectrum** $E_e$ is therefore defined as:

$$E_e(\omega_e, \beta) = \left\langle E(\omega_0, \beta) \frac{\mathrm{d}\omega_0}{\mathrm{d}\omega_e} \right\rangle_{\omega_e}, \tag{1.19}$$

where the notation $\langle \cdot \rangle_{\omega_e}$ indicates that the calculations are performed for a given encounter frequency $\omega_e$. In particular, this entails that, for the specified heading $\beta$ – and implicitly, for the considered vessel speed $U$ – the right-hand side of Eq. (1.19) is actually the sum of up to three terms, involving each of the intrinsic frequencies $\omega_0$ associated to the particular $\omega_e$.

In NetSSE, all methods rely on the assumption that the ship acts as a linear time-invariant (LTI) wave filter, which is most often valid for analysing rigid-body motions in mild to moderate sea states. This means that the mathematical relationship between the ship response and the encountered waves is linear, which in the frequency domain involves first-order wave-to-response **transfer functions**. For a specific response component $R$, the transfer function denoted $\Phi_R = \Phi_R(\omega_e, \beta)$ is complex-valued and is a function of the wave frequency and encounter angle. The transfer function depends on the geometry of the submerged part of the hull, the loading condition (draught, inertia distribution, etc.), and the vessel speed. In general, potential flow theory is used to calculate the transfer functions, relying on 3D panel codes, strip theory[43], or semi-empirical closed-form expressions (*CFE*). When viscous effects are important (e.g., in the presence of boundary layer separation on sharp edges), then potential flow theory becomes insufficient and Computational Fluid Dynamics (*CFD*) may need to be used.

Prior knowledge of the DWS allows the theoretical computation of an estimate of the cross-spectral density function $\tilde{S}_{RR'}(\omega_e)$ between a pair of responses $R$ and $R'$. In the most general case of an advancing ship in propagating short-crested waves, the **response cross-spectrum** is estimated by:

$$\hat{S}_{RR'}(\omega_e) = \int_{-\pi}^{\pi} \Phi_R(\omega_e, \beta)\, \overline{\Phi_{R'}(\omega_e, \beta)}\, E_e(\omega_e, \beta)\, \mathrm{d}\beta, \tag{1.20}$$

where the overline ($\bar{\cdot}$) denotes the complex conjugate. For a station-kept vessel (i.e., with zero forward speed), $\omega_e$ can be substituted with the intrinsic frequency $\omega_0$ in Eq. (1.20), since the two quantities become equivalent in Eq. (1.17) when $U = 0$.

The auto-spectral density function $S_{RR}(\omega_e)$ for a single response $R$ can be estimated through the same Eq. (1.20). The product of transfer functions may be rewritten as the squared modulus (or, amplitude) $|\Phi_R(\omega_e, \beta)|^2$, which, in maritime engineering, is commonly called **Response Amplitude Operator** (*RAO*).

In the case of long-crested waves encountering the ship with an angle $\beta_0$, the integration over wave directions is no longer relevant and the dependency on wave direction becomes implicit in Eq. (1.20), thus becoming:

$$\hat{S}_{RR'}(\omega_e) = \Phi_R(\omega_e; \beta_0)\, \overline{\Phi_{R'}(\omega_e; \beta_0)}\, E_e(\omega_e). \tag{1.21}$$

[43] Salvesen, N., Tuck, E. O., and Faltinsen, O. M. *Ship motions and sea loads*, vol. 75. The Society of Naval Architects and Marine Engineers, New York, NY, USA, November 1970.

### Inverse methods for sea state estimation from vessel measurements

*— When the Devil Hides in the Details.*

This section is a summary of existing inverse methods to derive sea state estimates from information on the wave-induced responses of in-situ floating platforms.

In the special case of buoy measurements, the translational and angular motions, resulting from waves, are the basis for an estimate of the DWS describing the onsite sea state. Due to the simple geometry of the buoy, the motion cross-spectra are related by simple algebraic expressions to the first five spectral wave parameters $E(\omega_0)$, $a_1(\omega_0)$, $b_1(\omega_0)$, $a_2(\omega_0)$, and $b_2(\omega_0)$, all defined earlier in Section 1.1.2. These formulas are commonly found in the literature for various types of single-point measuring devices (heave-roll-pitch buoys, particle-following buoys, GPS buoys, clover-leaf buoys, etc.); see, e.g., Benoit *et al.*[Page 7, 2] and Tucker and Pitt[Page 4, 48]. A number of methods have been developed to estimate the full DWS from the information of the first five spectral wave parameters or from equivalent parameters. The most widely used are Fourier series decomposition methods, parametric methods[27], maximum likelihood methods[Page 7, 2], maximum entropy methods[21][28], and Bayesian directional methods[14].

Marine vessels have a more complicated geometric shape compared to wave buoys, which introduces further complexities and uncertainties in the mathematical relationship between the waves and the induced vessel responses. As such, the same standard methods to analyse measurements from wave buoys cannot directly be applied for sea state estimation (*SSE*) from records of vessel responses. Physics-based frequency-domain approaches of the wave buoy analogy (*WBA*) involve the use of transfer functions to provide estimates of the complete directional wave spectrum based on ship response measurements. The main mathematical task of the wave estimation problem is to equate the theoretically estimated response spectrum $\hat{S}_{RR'}(\omega_e)$ of Eq. (1.20) with the measured one $\tilde{S}_{RR'}(\omega_e)$. Different strategies are applied to solve the highly under-determined equation system describing the physical relationship between the PSD of waves and corresponding responses. In general, frequency-domain methods can be classified into two types: on one hand, the **parametric methods** which assume the wave spectrum to be composed of a linear combination of parameterised wave spectra, e.g., Montazeri *et al.*[31][32] and Zago *et al.*[49], and, on the other hand, the **non-parametric methods** where the values of the wave spectrum are recovered in a completely discretised frequency-directional domain without prior assumption of a spectrum model, e.g., Iseki and Ohtsu[18], Tannuri *et al.*[47], Nielsen[33], Nielsen and Dietz[35], Brodtkorb and Nielsen[5], Kubo *et al.*[25]. These methods produce reasonable results in fair agreement with those of real ("classic") wave buoys. Some methods are not limited to an analysis of the rigid-body motions only; for instance, Chen *et al.*[6][7] used

[27] Longuet-Higgins, M. S., Cartwright, D. E., and Smith, N. D. Observation of the directional spectrum of sea waves using the motions of a floating buoy. *Ocean Wave Spectra* (1963), 111–136.

[21] Kobune, K., and Hashimoto, N. Estimation of directional spectra from the maximum entropy principle. In *Proceedings of the 5th International Offshore Mechanics and Arctic Engineering Symposium, Tokyo, Japan* (1986), vol. 1, pp. 80–85.

[28] Lygre, A., and Krogstad, H. E. Maximum entropy estimation of the directional distribution in ocean wave spectra. *Journal of Physical Oceanography 16*, 12 (1986), 2052–2060.

[14] Hashimoto, N., Kobune, K., and Kameyama, Y. Estimation of directional spectrum using the Bayesian approach and its application to field data analysis. *Rep. Port Harbor Res. Inst., Ministry of Transport, Japan 26* (1987), 57–100.

[31] Montazeri, N. *Estimation of waves and ship responses using onboard measurements*. Ph.D. dissertation, Technical University of Denmark, Department of Mechanical Engineering, Kongens Lyngby, Denmark, 2016.

[32] Montazeri, N., Nielsen, U. D., and Jensen, J. J. Estimation of wind sea and swell using shipboard measurements – A refined parametric modelling approach. *Applied Ocean Research 54* (2016), 73–86.

[49] Zago, L., Simos, A. N., Kawano, A., and Kogishi, A. M. A new vessel motion based method for parametric estimation of the waves encountered by the ship in a seaway. *Applied Ocean Research 134* (2023), 103499.

[18] Iseki, T., and Ohtsu, K. Bayesian estimation of directional wave spectra based on ship motions. *Control Engineering Practice 8*, 2 (2000), 215–219.

[47] Tannuri, E. A., Sparano, J. V., Simos, A. N., and Da Cruz, J. J. Estimating directional wave spectrum based on stationary ship motion measurements. *Applied Ocean Research 25*, 5 (2003), 243–261.

[33] Nielsen, U. D. Estimations of on-site directional wave spectra from measured ship responses. *Marine Structures 19*, 1 (2006), 33–69.

[35] Nielsen, U. D., and Dietz, J. Estimation of sea state parameters by the wave buoy analogy with comparisons to third generation spectral wave models. *Ocean Engineering 216* (2020), 107781.

[5] Brodtkorb, A. H., and Nielsen, U. D. Automatic sea state estimation with online trust measure based on ship response measurements. *Control Engineering Practice 130* (2023), 105375.

[25] Kubo, H., Okada, T., Chen, X., Kawamura, Y., Mitsuyuki, T., and Hayakawa, G. Bayesian updating of estimated parameters representing multimodal directional wave spectrum using measured ship hull stresses. In *Proceedings of the International Conference on Offshore Mechanics and Arctic Engineering* (2023), vol. 2: Structures, Safety, and Reliability, p. V002T02A032.

[6] Chen, X., Okada, T., Kawamura, Y., and Mitsuyuki, T. Estimation of on-site directional wave spectra using measured hull stresses on 14,000 TEU large container ships. *Journal of Marine Science and Technology (Japan) 25*, 3 (2020), 690–706.

[7] Chen, X., Okada, T., Kawamura, Y., and Mitsuyuki, T. Estimation of directional wave spectra and hull structural responses based on measured

several sensors measuring the hull stresses on container ships with advance speed to estimate the on-site directional wave spectra, with good accuracy results.

The procedure for these inverse methods is however far from straightforward and a number of technical complications are not entirely solved yet. In particular, the larger size (length, breadth, draught) of a ship compared to a wave buoy results in that the ship does not respond much to high encountered wave frequencies, commonly formulated as the **low-pass filter** characteristic. This is an issue for the estimation of the higher frequency part of the wave spectrum. Moreover, for an advancing vessel with non-zero forward speed, the Doppler shift distorts the wave spectrum in a body-fixed reference system. The transformation of the encounter wave spectrum back to the intrinsic frequency domain in an Earth-fixed reference frame does not have a unique solution in following seas; this is because of the triple-value problem, as was mentioned in Section 6, which eventually represents a loss of (wave) information and introduces uncertainties in the estimated spectrum.

The dependency on accurate transfer functions for the studied vessel has several disadvantages in model-based methods; one disadvantage is that transfer functions obtained through physical models have inherent uncertainties when evaluated in actual conditions. In contrast, purely data-driven approaches exist, relying on artificial intelligence (*AI*), i.e., machine learning (*ML*) and deep learning principles in SSE computations. The non-explicit relationship between sea state parameters and several features of the wave-induced responses of a particular ship is learned by comprehensive training with a large dataset of measured responses against available sea state information. The main advantages of those model-free methods are that the relationship between waves and ship motions is not assumed to be linear and the effects of uncertainties in the transfer functions are completely avoided in the estimates. On the one hand, **artificial neural networks** (*ANN*) have shown auspicious results in estimating the sea state based on records of the ship responses, e.g., Duz *et al.*[11], Kawai *et al.*[20], Mittendorf *et al.*[30], Cheng *et al.*[8], Nielsen *et al.*[36]. On the other hand, the need for collecting a large number of high-quality sensor measurements is emphasised in order to constitute the training set for a single ship. And so is the importance of having accurate ("external") data used during the training phase as a proxy for the ground true wave parameters at the exact spatiotemporal position of the ship. Moreover, there is no guarantee that the model is able to capture the physical properties of sea states that were not sufficiently represented in the training dataset, or in new operational conditions (e.g., considering the speed and loading setup).

Another noteworthy point is that ANN models are usually described as black-box models, since it is not possible to interpret the thousands of computations that are made in hidden layers, especially as they are devoid of any physical foundation. An ANN model built to solve a specific task cannot always be directly reused or adapted to solve a similar task. As an example, imagine a scenario where two ships are deployed in an offshore location to carry out an operation at sea. Both ships are equipped with a wave sensor that derives an estimate of the wave spectrum by resorting to a deep learning model. The training has been performed independently for each ship. Consider that an estimate of the DWS is derived by each of the two sensors. An issue would arise if the estimates end up being substantially different. Of course, if the transfer functions of the vessels are known, one could try to compare estimates of the response spectra via Eq. (1.20) using one or the other DWS, which would enable checking the quality of the wave estimate from both ships. If the transfer functions are completely unknown, then this "easy" check (allowed by physical principles) is not possible, and one would instead need to train a different (essentially reversed) ANN model in order to estimate responses from the given information of a DWS. Hybrid methods, combining physics-based methods of the WBA and ML-based methods,

hull data on 14,000 TEU large container ships. *Marine Structures 80* (2021), 103087.

[11] Duz, B., Mak, B., Hageman, R., and Grasso, N. Real time estimation of local wave characteristics from ship motions using artificial neural networks. In *Practical Design of Ships and Other Floating Structures*, T. Okada, K. Suzuki, and Y. Kawamura, Eds., vol. 65 of *Lecture Notes in Civil Engineering*. Springer Singapore, 2021, pp. 657–678.

[20] Kawai, T., Kawamura, Y., Okada, T., Mitsuyuki, T., and Chen, X. Sea state estimation using monitoring data by convolutional neural network (CNN). *Journal of Marine Science and Technology (Japan) 26*, 3 (2021), 947–962.

[30] Mittendorf, M., Nielsen, U. D., Bingham, H. B., and Storhaug, G. Sea state identification using machine learning: A comparative study based on in-service data from a container vessel. *Marine Structures 85* (2022), 103274.

[8] Cheng, X., Li, G., Han, P., Skulstad, R., Chen, S., and Zhang, H. Data-driven modeling for transferable sea state estimation between marine systems. *IEEE Transactions on Intelligent Transportation Systems 23*, 3 (2022), 2561–2571.

[36] Nielsen, U. D., Iwase, K., and Mounet, R. E. G. Comparing machine learning-based sea state estimates by the wave buoy analogy. *Applied Ocean Research 149* (2024), 104042.

have emerged, leveraging the advantages and minimising the cons of both classes of methods[13,34,37]. The ML-informed physics-based method proposed by Nielsen *et al.*[34,37] showed particularly promising results.

A specific disadvantage of the frequency-domain approaches is the requirement for stationary conditions, as well as the difficulty of producing real-time estimates, because of the need for backdated data over a certain time window to compute response spectra. Newer studies have worked on time-domain formulations. Some methods rely on Kalman filtering to estimate the directional wave spectrum, e.g., Pascoal and Guedes Soares[38], Pascoal *et al.*[39]. Dallolio *et al.*[9] employed a nonlinear second-order observer to estimate the wave encounter frequency from the heave motion of a wave-propelled unmanned surface vehicle. Dirdal *et al.*[10] proposed a phase-time-path difference model using a shipboard array of *IMU* sensors for online estimation of the wave direction and wavenumber. So far, the literature contains only a few methods developed to reconstruct wave profiles from measured response signals in real time. Akinturk *et al.*[1] have studied the identification of wave profiles using ANN. The machine learning method can train models that estimate the neighbouring wave field and predict the relative wave heading without knowledge of ship response characteristics. However, training data is needed to build the estimation model, and the difficulty lies in collecting a sufficient amount of training data. Takami *et al.*[45,46] presented a new approach to estimate the encountered sea surface elevation sequences based on short-time sequences of wave-induced vessel response measurements, by assuming that the response can be represented as a superposition of Prolate Spheroidal Wave Functions. Komoriyama *et al.*[23] developed a Kalman filter for identifying wave profiles encountered by a ship with no forward speed.

#### References

### 1.1.3 Usage

#### Summary of present functionalities

**NetSSE consists of a Python implementation that**:

- Proposes a structure for the establishment of a functional, heterogeneous network of wave sensor systems, including vessels as "sailing wave buoys";
- Facilitates the data-driven estimation of vessel hydrodynamic models to mathematically describe the response dynamics of floating observation platforms in waves;
- Enables the fusion of on-site spectral wave data from various kinds of in-situ observation platforms located in the same sea state;

[13] Han, P., Li, G., Cheng, X., Skjong, S., and Zhang, H. An uncertainty-aware hybrid approach for sea state estimation using ship motion responses. *IEEE Transactions on Industrial Informatics 18*, 2 (2022), 891–900.

[34] Nielsen, U. D., Bingham, H. B., Brodtkorb, A. H., Iseki, T., Jensen, J. J., Mittendorf, M., Mounet, R. E. G., Shao, Y., Storhaug, G., Sørensen, A. J., and Takami, T. Estimating waves via measured ship responses. *Scientific Reports 13*, 1 (2023), 17342.

[37] Nielsen, U. D., Mittendorf, M., Shao, Y., and Storhaug, G. Wave spectrum estimation conditioned on machine learning-based output using the wave buoy analogy. *Marine Structures 91* (2023), 103470.

[38] Pascoal, R., and Guedes Soares, C. Kalman filtering of vessel motions for ocean wave directional spectrum estimation. *Ocean Engineering 36*, 6-7 (2009), 477–488.

[39] Pascoal, R., Perera, L. P., and Guedes Soares, C. Estimation of directional sea spectra from ship motions in sea trials. *Ocean Engineering 132* (2017), 126–137.

[9] Dallolio, A., Alfredsen, J. A., Fossen, T. I., and Johansen, T. A. Experimental validation of a nonlinear wave encounter frequency estimator onboard a wave-propelled USV. *IFAC-PapersOnLine 54*, 16 (2021), 188–194.

[10] Dirdal, J. A., Skjetne, R., Rohá, J., and Fossen, T. I. Online wave direction and wave number estimation from surface vessel motions using distributed inertial measurement arrays and phase-time-path-differences. *Ocean Engineering 249* (2022), 110760.

[1] Akinturk, A., Zaman, H., Seo, D. C., He, M., and Mak, L. Estimates of wave field in the vicinity of a floating body using its motions and machine learning techniques. In *Oceans Conference Record (IEEE)* (2021), vol. 2021, Institute of Electrical and Electronics Engineers Inc., pp. 1–11.

[45] Takami, T., Nielsen, U. D., Jensen, J. J., and Chen, X. Estimation of encountered wave elevation sequences based on response measurements in multi-directional seas. *Applied Ocean Research 135* (2023), 103570.

[46] Takami, T., Nielsen, U. D., Xi, C., Jensen, J. J., and Oka, M. Reconstruction of incident wave profiles based on short-time ship response measurements. *Applied Ocean Research 123* (2022), 103183.

[23] Komoriyama, Y., Iijima, K., Tatsumi, A., and Fujikubo, M. Identification of wave profiles encountered by a ship with no forward speed using Kalman filter technique and validation by tank tests - long-crested irregular wave case -. *Ocean Engineering 271* (2023), 113627.

- Allow the spatial estimation of wave conditions within a large geographical domain, integrating individual sea state estimates derived from the records of ship responses.

#### Outlook

The developer team hopes that the current and future versions of NetSSE be relevant, accessible, and beneficial to a large range of stakeholders, not only from the various industries with interests in the sea but, equally important, from the international research community in ocean engineering and science, as well as international institutions and local governmental agencies.

### 1.1.4 Installation

#### Python version support

NetSSE requires a Python version of 3.12 (or any later version) to be installed in the environment.

#### Installing NetSSE

To use NetSSE, install it in a virtual environment using `pip`:

```
(.venv) $ python3 -m pip install netsse
```

This will also install or update the Python packages that NetSSE is dependent on, when needed.

#### Dependencies

NetSSE depends on a number of other open-source Python packages, of which all of the necessary are automatically installed when using the `pip install` manager. The required package versions are as follows:

```
numpy>=2.0.0
scipy>=1.14.0
matplotlib>=3.8.4
geopy>=2.4.1
```

### 1.1.5 Glossary

**AI**
: Artifical Intelligence.

**ANN**
: Artificial Neural Network.

**CFD**
: Computational Fluid Dynamics.

**CFE**
: Closed-Form Expression.

**DoF**
: Degree of Freedom.

**DP**
Dynamic Positioning.

**DSF**
Directional Spreading Function.

**DTU**
Danmarks Tekniske Universitet / Technical University of Denmark.

**DWS**
Directional Wave Spectrum.

**EMEP**
Extended Maximum Entropy Principle.

**FFT**
Fast Fourier Transform.

**IMU**
Inertial Measurement Unit.

**JONSWAP**
Joint North Sea Wave Project

**FIFO**
First-In/First-Out.

**ML**
Machine Learning.

**MSL**
Mean Sea Level.

**NED**
North-East-Down.

**PSD**
Power Spectral Density.

**RAO**
Response Amplitude Operator.

**SAWB**
Ship-As-a-Wave-Buoy

**SSE**
Sea State Estimation.

An ensemble of methods and techniques used to characterise the ocean wave system at a given time and position.

**std.**
Standard deviation.

**USV**
Unmanned Surface Vehicle.

**WBA**
Wave Buoy Analogy

### 1.1.6 Future work

**Todo**

This section is still under development.

CHAPTER

TWO

# EXAMPLES

Below, you can find examples of how NetSSE functions are used for sea state estimation from various observation platforms.

Examples in the form of Python scripts can also be found in the examples directory on GitLab.

**Todo**

This section is still under development.

## 2.1 Contents

### 2.1.1 Standard wave spectra

In this example application, it is shown how wave spectra can be generated by means of standardized expressions. The following modules and packages are imported:

```
import numpy as np
import matplotlib.pyplot as plt
from netsse.base import WaveSpectrum
```

#### 1. Introduction

In many engineering applications, such as the response analysis of marine structures exposed to waves, use of the wave spectrum is required in order to carry on the computations. In those cases, the sea state parameters can be exploited to reconstruct the wave spectrum by applying an adequately chosen **parameterised spectrum**.

There exist several standard forms of wave spectra. The choice of one form is application-specific and made based on the geographical area of study. Among other effects, the water depth, the predominance of wind sea over swell, or the opposite, and the fetch – defined as the length of the body of water that the wind blows over – should be considered (Mounet, 2023).

### 2. Sea state parameters

Assume a sea state characterised by the following two parameters:

- Significant wave height: $H_s = 2$ m.
- Peak wave period: $T_p = 10$ s.

```
Hs = 2
Tp = 10
```

```
fmax = 0.5
Nf = 100
f = np.linspace(0,fmax,Nf)
spec1d = WaveSpectrum(Hs,Tp,f,unit_freq='Hz')
spec1d.set_from_standard('JONSWAP',gamma=3.3)
spec1d.__dict__
```

```
JONSWAP spectrum with gamma = 3.3
```

```
{'Hs': 2,
 'Tp': 10,
 'timestamp': datetime.datetime(2024, 9, 4, 9, 53, 25, 621303),
 'lat': 0,
 'lon': 0,
 'depth': 0,
 'duration': 1800,
 'f': array([0.        , 0.00505051, 0.01010101, 0.01515152, 0.02020202,
        0.02525253, 0.03030303, 0.03535354, 0.04040404, 0.04545455,
        0.05050505, 0.05555556, 0.06060606, 0.06565657, 0.07070707,
        0.07575758, 0.08080808, 0.08585859, 0.09090909, 0.0959596 ,
        0.1010101 , 0.10606061, 0.11111111, 0.11616162, 0.12121212,
        0.12626263, 0.13131313, 0.13636364, 0.14141414, 0.14646465,
        0.15151515, 0.15656566, 0.16161616, 0.16666667, 0.17171717,
        0.17676768, 0.18181818, 0.18686869, 0.19191919, 0.1969697 ,
        0.2020202 , 0.20707071, 0.21212121, 0.21717172, 0.22222222,
        0.22727273, 0.23232323, 0.23737374, 0.24242424, 0.24747475,
        0.25252525, 0.25757576, 0.26262626, 0.26767677, 0.27272727,
        0.27777778, 0.28282828, 0.28787879, 0.29292929, 0.2979798 ,
        0.3030303 , 0.30808081, 0.31313131, 0.31818182, 0.32323232,
        0.32828283, 0.33333333, 0.33838384, 0.34343434, 0.34848485,
        0.35353535, 0.35858586, 0.36363636, 0.36868687, 0.37373737,
        0.37878788, 0.38383838, 0.38888889, 0.39393939, 0.3989899 ,
        0.4040404 , 0.40909091, 0.41414141, 0.41919192, 0.42424242,
        0.42929293, 0.43434343, 0.43939394, 0.44444444, 0.44949495,
        0.45454545, 0.45959596, 0.46464646, 0.46969697, 0.47474747,
        0.47979798, 0.48484848, 0.48989899, 0.49494949, 0.5       ]),
 'Sf': array([0.00000000e+000, 0.00000000e+000, 0.00000000e+000, 0.00000000e+000,
        0.00000000e+000, 2.54146386e-130, 1.34092441e-061, 2.64173468e-032,
        3.25376512e-018, 8.12466134e-011, 1.13343835e-006, 3.10637024e-004,
        9.51438902e-003, 8.07194202e-002, 3.13007504e-001, 7.42569629e-001,
        1.30732433e+000, 2.06147914e+000, 3.54781338e+000, 6.35293573e+000,
        7.70300951e+000, 5.90522384e+000, 3.73016141e+000, 2.48105612e+000,
        1.89555896e+000, 1.59247382e+000, 1.38611097e+000, 1.21438876e+000,
```

(continues on next page)

(continued from previous page)

```
       1.06293581e+000, 9.29120397e-001, 8.11736007e-001, 7.09372739e-001,
       6.20451554e-001, 5.43379723e-001, 4.76647834e-001, 4.18878021e-001,
       3.68842117e-001, 3.25462479e-001, 2.87803483e-001, 2.55058563e-001,
       2.26535656e-001, 2.01642711e-001, 1.79874090e-001, 1.60798287e-001,
       1.44047082e-001, 1.29306110e-001, 1.16306749e-001, 1.04819195e-001,
       9.46465756e-002, 8.56199612e-002, 7.75941540e-002, 7.04441272e-002,
       6.40620220e-002, 5.83546121e-002, 5.32411621e-002, 4.86516178e-002,
       4.45250744e-002, 4.08084779e-002, 3.74555247e-002, 3.44257248e-002,
       3.16836053e-002, 2.91980311e-002, 2.69416239e-002, 2.48902660e-002,
       2.30226744e-002, 2.13200349e-002, 1.97656875e-002, 1.83448553e-002,
       1.70444103e-002, 1.58526708e-002, 1.47592255e-002, 1.37547818e-002,
       1.28310327e-002, 1.19805416e-002, 1.11966417e-002, 1.04733476e-002,
       9.80527881e-003, 9.18759181e-003, 8.61592088e-003, 8.08632594e-003,
       7.59524661e-003, 7.13946167e-003, 6.71605328e-003, 6.32237534e-003,
       5.95602542e-003, 5.61481986e-003, 5.29677172e-003, 5.00007109e-003,
       4.72306760e-003, 4.46425479e-003, 4.22225623e-003, 3.99581300e-003,
       3.78377257e-003, 3.58507883e-003, 3.39876311e-003, 3.22393612e-003,
       3.05978076e-003, 2.90554557e-003, 2.76053887e-003, 2.62412351e-003]),
'om': array([0.        , 0.03173326, 0.06346652, 0.09519978, 0.12693304,
       0.1586663 , 0.19039955, 0.22213281, 0.25386607, 0.28559933,
       0.31733259, 0.34906585, 0.38079911, 0.41253237, 0.44426563,
       0.47599889, 0.50773215, 0.53946541, 0.57119866, 0.60293192,
       0.63466518, 0.66639844, 0.6981317 , 0.72986496, 0.76159822,
       0.79333148, 0.82506474, 0.856798  , 0.88853126, 0.92026451,
       0.95199777, 0.98373103, 1.01546429, 1.04719755, 1.07893081,
       1.11066407, 1.14239733, 1.17413059, 1.20586385, 1.23759711,
       1.26933037, 1.30106362, 1.33279688, 1.36453014, 1.3962634 ,
       1.42799666, 1.45972992, 1.49146318, 1.52319644, 1.5549297 ,
       1.58666296, 1.61839622, 1.65012947, 1.68186273, 1.71359599,
       1.74532925, 1.77706251, 1.80879577, 1.84052903, 1.87226229,
       1.90399555, 1.93572881, 1.96746207, 1.99919533, 2.03092858,
       2.06266184, 2.0943951 , 2.12612836, 2.15786162, 2.18959488,
       2.22132814, 2.2530614 , 2.28479466, 2.31652792, 2.34826118,
       2.37999443, 2.41172769, 2.44346095, 2.47519421, 2.50692747,
       2.53866073, 2.57039399, 2.60212725, 2.63386051, 2.66559377,
       2.69732703, 2.72906028, 2.76079354, 2.7925268 , 2.82426006,
       2.85599332, 2.88772658, 2.91945984, 2.9511931 , 2.98292636,
       3.01465962, 3.04639288, 3.07812614, 3.10985939, 3.14159265]),
'Som': array([0.00000000e+000, 0.00000000e+000, 0.00000000e+000, 0.00000000e+000,
       0.00000000e+000, 4.04486536e-131, 2.13414749e-062, 4.20445133e-033,
       5.17852803e-019, 1.29308001e-011, 1.80392317e-007, 4.94394179e-005,
       1.51426204e-003, 1.28468947e-002, 4.98166914e-002, 1.18183627e-001,
       2.08067129e-001, 3.28094595e-001, 5.64652037e-001, 1.01110112e+000,
       1.22597204e+000, 9.39845564e-001, 5.93673627e-001, 3.94872345e-001,
       3.01687579e-001, 2.53450080e-001, 2.20606413e-001, 1.93275975e-001,
       1.69171488e-001, 1.47874104e-001, 1.29191798e-001, 1.12900178e-001,
       9.87479318e-002, 8.64815689e-002, 7.58608589e-002, 6.66665076e-002,
       5.87030461e-002, 5.17989623e-002, 4.58053470e-002, 4.05938311e-002,
       3.60542695e-002, 3.20924342e-002, 2.86278505e-002, 2.55918422e-002,
       2.29258052e-002, 2.05797066e-002, 1.85107940e-002, 1.66824931e-002,
       1.50634704e-002, 1.36268401e-002, 1.23494932e-002, 1.12115311e-002,
       1.01957875e-002, 9.28742496e-003, 8.47359412e-003, 7.74314547e-003,
```

(continues on next page)

(continued from previous page)

```
       7.08638568e-003, 6.49487098e-003, 5.96123190e-003, 5.47902427e-003,
       5.04260240e-003, 4.64701098e-003, 4.28789262e-003, 3.96140888e-003,
       3.66417244e-003, 3.39318894e-003, 3.14580687e-003, 2.91967440e-003,
       2.71270216e-003, 2.52303091e-003, 2.34900370e-003, 2.18914152e-003,
       2.04212228e-003, 1.90676242e-003, 1.78200087e-003, 1.66688504e-003,
       1.56055859e-003, 1.46225065e-003, 1.37126640e-003, 1.28697875e-003,
       1.20882104e-003, 1.13628062e-003, 1.06889308e-003, 1.00623729e-003,
       9.47930887e-004, 8.93626336e-004, 8.43007402e-004, 7.95786030e-004,
       7.51699555e-004, 7.10508218e-004, 6.71992950e-004, 6.35953390e-004,
       6.02206108e-004, 5.70583018e-004, 5.40929949e-004, 5.13105370e-004,
       4.86979233e-004, 4.62431940e-004, 4.39353407e-004, 4.17642227e-004]),
 'nature': 'standard',
 'method': 'JONSWAP',
 'gamma': 3.3}
```

### References

1. Mounet, R.E.G. (2023). *Sea State Estimation Based on Measurements from Multiple Observation Platforms*. PhD Thesis. Technical University of Denmark. https://orbit.dtu.dk/en/publications/sea-state-estimation-based-on-measurements-from-multiple-observat

## 2.1.2 Building an observation network

```
import netsse
```

```
vessel1 = netsse.base.Platform('Containership1')
vessel2 = netsse.base.Platform('RoRo1')
vessel3 = netsse.base.Platform()
print(vessel3.name)
```

```
Platform9XE0N
```

```
network = netsse.base.Network(name='TestNetwork',nature='test',platforms=[vessel1,
↪vessel2])
print([(platform.name,platform.owner) for platform in network.platforms])
```

```
[('Containership1', 'undefined'), ('RoRo1', 'undefined')]
```

```
network.add_platform(vessel3)
print([(platform.name,platform.owner) for platform in network.platforms])
```

```
Platform 'Platform9XE0N' was added to the network.
[('Containership1', 'undefined'), ('RoRo1', 'undefined'), ('Platform9XE0N', 'undefined')]
```

```
vessel2.owner = 'SomeCompany'
network.add_platform(vessel2)
print([(platform.name,platform.owner) for platform in network.platforms])
```

```
Platform 'RoRo1' in TestNetwork was updated with new information.
[('Containership1', 'undefined'), ('RoRo1', 'SomeCompany'), ('Platform9XE0N', 'undefined
↪')]
```

```
network.remove_platform(vessel3,'Containership1','NonExistentShip')
print([(platform.name,platform.owner) for platform in network.platforms])
```

```
Platform 'Platform9XE0N' was removed from TestNetwork.
Platform 'Containership1' was removed from TestNetwork.
Platform 'NonExistentShip' not found.
[('RoRo1', 'SomeCompany')]
```

CHAPTER

THREE

# API REFERENCE

This page contains auto-generated API reference documentation[1].

## 3.1 netsse

An open-source Python package for **network-based sea state estimation** from ships, buoys, and other observation platforms.



To credit the author, users are encouraged to use below reference:

```
Mounet, R. E. G., & Nielsen, U. D. NetSSE: An open-source Python package
for network-based sea state estimation from ships, buoys, and other
observation platforms (version 2.0). Technical University of Denmark,
GitLab. July 2024. https://doi.org/10.11583/DTU.26379811.
```

*Last updated on 11-07-2024 by R.E.G. Mounet*

[1] Created with sphinx-autoapi

### 3.1.1 Subpackages

#### netsse.analys

A set of functions for **analysing** ocean wave measurements.

#### Copyright (C) 2024 Technical University of Denmark, R.E.G. Mounet

*This code is part of the NetSSE software.*

NetSSE is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

NetSSE is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see https://www.gnu.org/licenses/.

To credit the author, users are encouraged to use below reference:

```
Mounet, R. E. G., & Nielsen, U. D. NetSSE: An open-source Python package
for network-based sea state estimation from ships, buoys, and other
observation platforms (version 2.0). Technical University of Denmark,
GitLab. July 2024. https://doi.org/10.11583/DTU.26379811.
```

*Last updated on 11-07-2024 by R.E.G. Mounet*

#### Submodules

#### netsse.analys.buoy

Functions to **process wave buoy** motion signals into directional wave spectra.

#### Copyright (C) 2024 Technical University of Denmark, R.E.G. Mounet

*This code is part of the NetSSE software.*

NetSSE is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

NetSSE is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see https://www.gnu.org/licenses/.

To credit the author, users are encouraged to use below reference:

```
Mounet, R. E. G., & Nielsen, U. D. NetSSE: An open-source Python package
for network-based sea state estimation from ships, buoys, and other
observation platforms (version 2.0). Technical University of Denmark,
GitLab. July 2024. https://doi.org/10.11583/DTU.26379811.
```

*Last updated on 11-07-2024 by R.E.G. Mounet*

## Functions

### netsse.analys.buoy.cross_spec2Fourier_coef

netsse.analys.buoy.**cross_spec2Fourier_coef**(*Gf*)

Computes the Fourier coefficients from the cross-spectra of a heave-East-North wave buoy, as per Benoit et al. (1997).

**Parameters**

**Gf** (*array_like of shape (3,3,Nf)*) – Cross-spectra (Heave-East-North), as a function of frequency in Hertz.

**Returns**

- **Sf** (*array_like of shape (Nf,)*) – One-sided variance spectrum of the waves [m^2.s], as a function of frequency in Hertz.
- **a1, a2, b1, b2** (*array_like of shape (Nf,)*) – Frequency-dependent Fourier coefficients of the directional wave spectrum.

**See also**

*Shannon_MEMII_Newton*
Reconstructs the directional spreading function based on the first four Fourier coefficients of a directional wave spectrum.

#### References

Benoit, M., Frigaard, P., & Schäffer, H. A. (1997). *Analysing Multidirectional Wave Spectra: A tentative classification of available methods*. Proceedings of the 27th IAHR Congress, San Francisco, CA, USA (pp. 131–158). Canadian Government Publishing.

#### Example

```
>>> Sf, a1, a2, b1, b2 = cross_spec2Fourier_coef(Gf)
```

### netsse.analys.buoy.Shannon_MEMII_Newton

netsse.analys.buoy.**Shannon_MEMII_Newton**(*a1*, *a2*, *b1*, *b2*, *freq*, *theta*, *maxiter*, *tol_error*, *miniter=50*, *approx=False*)

Reconstructs the directional spreading function based on the first four Fourier coefficients of a directional wave spectrum.

The function implements the Maximum Entropy Principle (Hashimoto, 1997), either via running the Newton local linearisation method or following the approximation method by Kim et al. (1994).

**Parameters**

- **a1** (*array_like of shape (Nf,)*) – First four Fourier coefficients of the directional wave spectrum.
- **a2** (*array_like of shape (Nf,)*) – First four Fourier coefficients of the directional wave spectrum.
- **b1** (*array_like of shape (Nf,)*) – First four Fourier coefficients of the directional wave spectrum.
- **b2** (*array_like of shape (Nf,)*) – First four Fourier coefficients of the directional wave spectrum.
- **freq** (*array_like of shape (Nf,)*) – Vector of (discretized) wave frequencies [Hz].
- **theta** (*array_like of shape (Ntheta,)*) – Vector of wave headings [deg].
- **maxiter** (*int*) – Maximum number of iterations.
- **tol_error** (*float*) – Tolerance in the relative error.
- **miniter** (*int, default 50*) – Minimum number of iterations.
- **approx** (*bool, default False*) – Boolean which indicates whether the approximation method should be used for finding the Lagrangian multipliers.

**Returns**

- **D** (*array_like of shape (Nf,Ntheta)*) – Directional spreading function.
- **flag** (*array_like of shape (Nf,)*) – Boolean flag indicating unconverged frequencies (`flag = 0` for unconverged)
- **L1, L2, L3, L4** (*array_like of shape (Nf,)*) – Optimized Lagrange multipliers.

**See also**

*cross_spec2Fourier_coef*
: Computes the Fourier coefficients from the cross-spectra of a heave-East-North wave buoy.

*netsse.analys.emep.emep*
: Extended Maximum Entropy Principle (EMEP) method for reconstructing the directional spreading function based on the cross-power spectra of measured wave-induced responses.

## References


1. Benoit, M., Frigaard, P., & Schäffer, H. A. (1997). *Analysing Multidirectional Wave Spectra: A tentative classification of available methods*. Proceedings of the 27th IAHR Congress, San Francisco, CA, USA (pp. 131–158). Canadian Government Publishing.
2. Hashimoto, N. (1997). *Analysis of the Directional Wave Spectrum from Field Data*. Advances in coastal and ocean engineering. 3:103-44.
3. Kim, T., Lin, L.-H., & Wang, H. (1994). *Application of Maximum Entropy Method to the Real Sea Data*. In Coastal Engineering (pp. 340–355).

### Example

```
>>> D, flag, L1, L2, L3, L4 =
...     Shannon_MEMII_Newton(a1,a2,b1,b2,freq,theta,maxiter,tol_error,miniter=50,
↪False)
```

## netsse.analys.buoy.Fourier2spread_dist_params

netsse.analys.buoy.**Fourier2spread_dist_params**(*a1*, *b1*, *a2*, *b2*)

Infers the parameters of the directional spreading distribution function from the Fourier coefficients, as per Kuik et al. (1988).

**Parameters**

- **a1** (*array_like of shape (Nf,)*) – First four Fourier coefficients of the directional wave spectrum.
- **b1** (*array_like of shape (Nf,)*) – First four Fourier coefficients of the directional wave spectrum.
- **a2** (*array_like of shape (Nf,)*) – First four Fourier coefficients of the directional wave spectrum.
- **b2** (*array_like of shape (Nf,)*) – First four Fourier coefficients of the directional wave spectrum.

**Returns**

- **alpha** (*array_like of shape (Nf,)*) – Wave direction [rad].
- **sigma** (*array_like of shape (Nf,)*) – Directional spread [rad].
- **gamma** (*array_like of shape (Nf,)*) – Skewness of the directional distribution [-].
- **delta** (*array_like of shape (Nf,)*) – Kurtosis of the directional distribution [-].

**See also**

*spread_dist_params2Fourier*
Infers the Fourier coefficients from the parameters of the directional spreading distribution function.

### References

Kuik, A. J., van Vledder, G. P., & Holthuijsen, L. H. (1988). *A Method for the Routine Analysis of Pitch-and-Roll Buoy Wave Data*. Journal of Physical Oceanography, 18(7), 1020–1034.

### Example

```
>>> [alpha, sigma, gamma, delta] = Fourier2spread_dist_params(a1,b1,a2,b2)
```

## netsse.analys.buoy.spread_dist_params2Fourier

netsse.analys.buoy.**spread_dist_params2Fourier**(*alpha*, *sigma*, *gamma*, *delta*, *unit='rad'*)

Infers the Fourier coefficients from the parameters of the directional spreading distribution function, as per Kuik et al. (1988).

**Parameters**

- **alpha** (*array_like of shape (Nf,)*) – Wave direction [rad, or deg].
- **sigma** (*array_like of shape (Nf,)*) – Directional spread [rad, or deg].
- **gamma** (*array_like of shape (Nf,)*) – Skewness of the directional distribution [-].
- **delta** (*array_like of shape (Nf,)*) – Kurtosis of the directional distribution [-].
- **unit** (*{'rad','deg'}, optional*) – Unit of the wave direction and directional spread: `'deg'` or `'rad'` (default).

**Returns**

**a1, b1, a2, b2** – First four Fourier coefficients of the directional wave spectrum.

**Return type**

array_like of shape (Nf,)

**See also**

*Fourier2spread_dist_params*
Infers the parameters of the directional spreading distribution function from the Fourier coefficients.

*Shannon_MEMII_Newton*
Reconstructs the directional spreading function based on the first four Fourier coefficients of a directional wave spectrum.

### References

Kuik, A. J., van Vledder, G. P., & Holthuijsen, L. H. (1988). *A Method for the Routine Analysis of Pitch-and-Roll Buoy Wave Data*. Journal of Physical Oceanography, 18(7), 1020–1034.

### Example

```
>>> [a1,b1,a2,b2] = spread_dist_params2Fourier(alpha,sigma,gamma,delta,unit)
```

| | |
|---|---|
| *cross_spec2Fourier_coef*(Gf) | Computes the Fourier coefficients from the cross-spectra of a |
| *Shannon_MEMII_Newton*(a1, a2, b1, b2, freq, theta, ...) | Reconstructs the directional spreading function based on the first four Fourier |
| *Fourier2spread_dist_params*(a1, b1, a2, b2) | Infers the parameters of the directional spreading distribution function |
| *spread_dist_params2Fourier*(alpha, sigma, gamma, delta) | Infers the Fourier coefficients from the parameters of the directional |

### netsse.analys.emep

Python implementation of the **EMEP** algorithm, used for analyzing directional wave spectra from field data.

### Copyright (C) 2024 Technical University of Denmark, R.E.G. Mounet

*This code is part of the NetSSE software.*

NetSSE is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

NetSSE is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see https://www.gnu.org/licenses/.

To credit the author, users are encouraged to use below reference:

```
Mounet, R. E. G., & Nielsen, U. D. NetSSE: An open-source Python package
for network-based sea state estimation from ships, buoys, and other
observation platforms (version 2.0). Technical University of Denmark,
GitLab. July 2024. https://doi.org/10.11583/DTU.26379811.
```

*Last updated on 11-07-2024 by R.E.G. Mounet*

### Functions

### netsse.analys.emep.norm_resp

netsse.analys.emep.**norm_resp**(*Rxy_ReIm*, *TRF*, *S1D*, *f*, *mn=None*, *ReIm=None*, *Na=None*, *axy=None*)

Normalizes the response spectra and transfer functions for application of the EMEP method.

> **Note**
>
> The normalization factors are found in Bendat & Piersol (2010).

**Parameters**

- **Rxy_ReIm** (*array_like of shape (Nfreq,Nxy)*) – Response spectra as a function of frequency `f`, considering *Nxy* pairs of responses.

- **TRF** (*Multidimensional array of shape (Nresp,Ntheta,Nfreq)*) – Transfer functions for *Nresp* different responses, as functions of wave direction and frequency `f`.
- **S1D** (*array_like of shape (Nfreq,)*) – 1-D wave spectrum, as a function of frequency `f`.
- **f** (*array_like of shape (Nfreq,)*) – Vector of wave frequencies.
- **mn** (*array_like of shape (Nxy,2), optional*) – Definition of the considered responses in `Rxy_ReIm`. Elements in `mn` correspond to the indices of the pairs of responses along the first dimension of `TRF`, i.e., from 0 to *Nresp*-1. Default is `None`; in that case, the response pairs are ordered in the following order (illustrated for *Nresp = 3*):

  ```
  Rxy_ReIm = [Re(R00),Re(R11),Re(R22),Re(R01),Re(R02),Re(R12),Im(R01),
  Im(R02),Im(R12)]
  ```

  ```
  mn = [[0, 0], [1, 1], [2, 2], [0, 1], [0, 2], [1, 2], [0, 1], [0, 2],
  [1, 2]]
  ```

- **ReIm** (*array_like of shape (Nxy,) or (Nxy,1), optional*) – Vector indicating whether the real part (`'R'`) or imaginary part (`'I'`) of the response cross-spectrum is considered in `Rxy_ReIm`. Default is `None`; in that case, the real and imaginary parts of the response pairs are ordered in the order shown above, thus:

  ```
  ReIm = ['R', 'R', 'R', 'R', 'R', 'R', 'I', 'I', 'I']
  ```

- **Na** (*str, or array_like of shape (Nxy,), optional*) – Number of the ensembled average for computation of the response spectra variance. Default is `None`, for which case `Na = 1`.
- **axy** (*array_like of shape (Nxy,), optional*) – Additional weights (must be positive). Any weight greater than zero increases the importance given to a corresponding pair of responses in the EMEP method. Default is `None`, for which case the weights are set to zero.

**Returns**

- **Rxy_ReIm_n** (*array_like of shape (Nfreq,Nxy)*) – Normalized version of the response spectra.
- **Hn** (*array_like of shape (Ntheta,Nfreq,Nxy)*) – Normalized version of the real and imaginary parts of the product of the transfer functions for pairs of responses. The real and imaginary parts for the pairs of responses are output in the same order as `Rxy_ReIm`.

**See also**

*emep*
Extended Maximum Entropy Principle (EMEP) method for reconstructing the directional spreading function based on the cross-power spectra of measured wave-induced responses.

### References

Bendat, J. S., & Piersol, A. G. (2010). Random Data: Analysis and Measurement Procedures. In Measurement Science and Technology (Vol. 11, Issue 12). Wiley. https://doi.org/10.1002/9781118032428

### Example

```
>>> Rxy_ReIm_n, Hn = norm_resp(Rxy_ReIm,TRF,S1D,f,mn=None,ReIm=None,Na=None,
↪axy=None)
```

## netsse.analys.emep.norm_DSF

netsse.analys.emep.**norm_DSF**(*D*, *theta*)

Normalizes the directional spreading function.

The function ensures that the output has a unit integral over wave directions.

**Parameters**

- **D** (*array_like of shape (nt,nf)*) – Directional spreading function.
- **theta** (*array_like of shape (nt,)*) – Vector of wave directions.

**Returns**

**Dn** – Normalized spreading function.

**Return type**

array_like of shape (nt,nf)

### Example

```
>>> Dn = norm_DSF(D, theta)
```

## netsse.analys.emep.emep

netsse.analys.emep.**emep**(*Sxyn*, *Hn*, *theta*, *fi*, *k*, *opt=None*)

Applies the Extended Maximum Entropy Principle (EMEP) method to reconstruct the directional spreading function based on the cross-power spectra of measured wave-induced responses.

**Note**

This implementation is based on the functions EMEP in the DIWASP toolbox and EMEM in the WAFO toolbox.

**Parameters**

- **Sxyn** (*ndarray of shape (nf,m*m), where nf is the number of frequencies and m is the number of sensors*) – Real and imaginary parts of the normalised cross-power spectral density:

  `Sxyn(f,i) = Re(Smn(f)/(Szeta(f)*Wmn(f)))` or the imaginary (`Im(.)`) counterpart.

- **Hn** (*ndarray of shape (nt,nf,m*m), where nt is the number of theta values*) – Matrix of the real and imaginary parts of the normalised RAO products, in the same order of response pairs as for `Sxyn`:

  `Hn(theta,f,i) = Re(Phi_m(theta,f)*conj(Phi_n(theta,f))/Wmn(f))` or the imaginary (`Im(.)`) counterpart.

- **theta** (*ndarray of shape (nt,)*) – Vector of wave headings.

  > **Warning**
  >
  > The wave heading *must be* in a wrapped format, corresponding to [0,360] deg. For ship responses, those headings must be the *relative* wave headings.

- **fi** (*ndarray of shape (nf,)*) – Frequency vector.
- **k** (*array_like*) – List of indices corresponding to frequencies where the wave power spectral density is substantially greater than zero.
- **opt** (*dict, optional*) – Optional parameters controlling the EMEM calculation. Available options are:
  - **'errortol'**
    [float, default 0.0005] Error tolerance for convergence.
  - **'maxiter'**
    [int, default 25] Maximum number of iterations for the Newton-Raphson method.
  - **'relax'**
    [float, default 1] Relaxation parameter for controlling step shape in optimization.
  - **'maxcoef'**
    [float, default 10000] Maximum value for coefficients to prevent divergence.
  - **'coefabstol'**
    [float, default 0.01] Coefficient absolute tolerance for convergence.
  - **'minmodelorder'**
    [int, default 1] Minimum model order for AIC evaluation.
  - **'maxmodelorder'**
    [int, default M/2 + 1] Maximum model order for AIC evaluation.
  - **'diradjust'**
    [float, default 0] Deviation term to adjust the wave directions to other direction conventions.

    > **Warning**
    >
    > For ship responses, a value of $-\pi/2$ was necessary to use: `opt = {'diradjust': numpy.pi/2}`.

  - **'message'**
    [{0,1}] Display messages during the calculation.

**Returns**

**D** – Estimated directional spreading distribution matrix of shape (nt, nf).

**Return type**
ndarray

**See also**

*norm_resp*
Normalizes the response spectra and transfer functions.

*norm_DSF*
Normalizes the directional spreading function.

*netsse.analys.buoy.Shannon_MEMII_Newton*
Reconstructs the directional spreading function based on the first four Fourier coefficients of a directional wave spectrum.

### References


1. Hashimoto, N. (1997). "Analysis of the directional wave spectra from field data." Advances in Coastal and Ocean Engineering, Vol.3., pp.103-143.
2. DIWASP, a directional wave spectra toolbox for MATLAB: User Manual. Research Report WP-1601-DJ (V1.1), Centre for Water Research, University of Western Australia.
3. Brodtkorb, P.A., Johannesson, P., Lindgren, G., Rychlik, I., Rydén, J. and Sö, E. (2000). "WAFO - a Matlab toolbox for analysis of random waves and loads", Proc. 10th Int. Offshore and Polar Eng. Conf., Seattle, USA, Vol III, pp. 343-350.


### Example

```
>>> D = emep(Sxyn, Hn, theta, fi, k, opt=None)
```

| | |
|---|---|
| *norm_resp*(Rxy_ReIm, TRF, S1D, f[, mn, ReIm, Na, axy]) | Normalizes the response spectra and transfer functions for application of the EMEP method. |
| *norm_DSF*(D, theta) | Normalizes the directional spreading function. |
| *emep*(Sxyn, Hn, theta, fi, k[, opt]) | Applies the Extended Maximum Entropy Principle (EMEP) method to reconstruct the directional |

## netsse.analys.enc_spec

Functions to compute spectra in a **body-fixed** reference system.

To credit the author, users are encouraged to use below reference:

```
Mounet, R. E. G., & Nielsen, U. D. NetSSE: An open-source Python package
for network-based sea state estimation from ships, buoys, and other
observation platforms (version 2.0). Technical University of Denmark,
GitLab. July 2024. https://doi.org/10.11583/DTU.26379811.
```

*Last updated on 11-07-2024 by R.E.G. Mounet*

## Functions

### netsse.analys.enc_spec.resp_spec_1d

netsse.analys.enc_spec.**resp_spec_1d**(*f0*, *ws*, *TRF1*, *TRF2*, *fe*, *U*, *beta*)

Computes a response cross-spectrum in both the absolute and encountered frequency domains, in a case of long-crested seas.

**Parameters**

- **f0** (*array_like of shape (Nf0,1)*) – Vector of absolute frequencies [Hz].
- **ws** (*array_like of shape (Nf0,1) in 1-D*) – 1-D wave spectrum, as a function of (absolute) wave frequency `f0` [Hz].
- **TRF1** (*array_like of shape (Nf0,1)*) – Transfer functions of two response components, as a function of frequency `f0` [Hz].
- **TRF2** (*array_like of shape (Nf0,1)*) – Transfer functions of two response components, as a function of frequency `f0` [Hz].
- **fe** (*array_like of shape (Nfe,1)*) – Vector of encounter frequencies [Hz].
- **U** (*float*) – Ship speed [m/s].
- **beta** (*float*) – Ship heading, relative to the wave direction [deg].

**Returns**

- **RspecAbs** (*array_like of shape (Nf0,1)*) – Response spectrum, as a function of absolute wave frequency `f0` [Hz].
- **RspecEnc** (*array_like of shape (Nfe,1)*) – Response spectrum, as a function of encountered frequency `fe` [Hz].

**See also**

*resp_spec_2d*
Computes a response cross-spectrum in the encounter-frequency domain, in case of short-crested sea.

### Example

```
>>> RspecAbs, RspecEnc = resp_spec_1d(f0,ws,TRF1,TRF2,fe,U,beta)
```

## netsse.analys.enc_spec.resp_spec_2d

netsse.analys.enc_spec.**resp_spec_2d**(*freq0*, *mu*, *ws0*, *beta_TRF*, *TRF1*, *TRF2*, *freqE*, *U*, *psi*, *conv='from'*, *unit_freq='rad/s'*)

Computes a response cross-spectrum in the encounter-frequency domain, in a case of short-crested seas.

**Note**

This is a Python implementation of the algorithm given in Nielsen et al. (2021).

**Parameters**

- **freq0** (*array_like of shape (Nf0,)*) – Vector of absolute frequencies, in which the wave spectrum and the transfer functions are expressed.
- **mu** (*array_like of shape (Nmu,)*) – Vector of wave directions [deg], relative to North (in a NED-reference frame).
- **ws0** (*array_like of shape (Nf0,Nmu) in 2-D*) – 2-D wave spectrum, as a function of (absolute) wave frequency `freq0` and wave direction `mu` [rad].
- **beta_TRF** (*array_like of shape (Nbeta,)*) – Vector of relative wave headings [deg], in which the transfer functions are expressed.
- **TRF1** (*array_like of shape (Nf0,Nbeta)*) – Transfer function of the first and second response component, respectively, as a function of frequency `freq0` and relative wave heading.
- **TRF2** (*array_like of shape (Nf0,Nbeta)*) – Transfer function of the first and second response component, respectively, as a function of frequency `freq0` and relative wave heading.
- **freqE** (*array_like of shape (Nfe,)*) – Vector of encounter frequencies for the output response spectrum.
- **U** (*float*) – Ship speed [m/s]
- **psi** (*float*) – Ship heading [deg], relative to North (in a NED-reference frame).
- **conv** (*{'from','to'}, optional*) – The convention that is used to express the direction `mu` of wave spectrum:
  - **'from' :**
    The naval architecture convention, indicating where the waves are COMING FROM.

- **'to' :**
  The oceanographic convention, indicating where the waves are GOING TO. The default is "from" direction.

- **unit_freq** (*{'rad/s','Hz'}, optional*) – Unit of the frequencies:
  - **'rad/s' :**
    The variables `freq0` and `freqE` denote the angular frequencies in radians per second.
  - **'Hz' :**
    The 2-D wave spectrum `ws0` is expressed as a function of frequency in Hertz. The default is 'rad/s'.

**Returns**
**RspecEnc** – Response spectrum, as a function of encounter frequency `freqE`.

**Return type**
array_like of shape (Nfe,)

**See also**

*resp_spec_1d*
Computes a response cross-spectrum in both the absolute and encountered frequency domains, in a case of long-crested seas.

### References

Nielsen, U. D., Mounet, R. E. G., & Brodtkorb, A. H. (2021). Tuning of transfer functions for analysis of wave-ship interactions. Marine Structures, 79, 103029. https://doi.org/10.1016/j.marstruc.2021.103029

### Example

```
>>> RspecEnc =
...     resp_spec_2d(freq0,mu,ws0,beta_TRF,TRF1,TRF2,freqE,U,psi,'from','rad/s')
```

| | |
|---|---|
| *resp_spec_1d*(f0, ws, TRF1, TRF2, fe, U, beta) | Computes a response cross-spectrum in both the absolute and encountered |
| *resp_spec_2d*(freq0, mu, ws0, beta_TRF, TRF1, TRF2, ...) | Computes a response cross-spectrum in the encounter-frequency domain, in a case of |

## netsse.analys.sawb

Python implementation of various algorithms for sea state estimation using the **wave-buoy analogy** (WBA), i.e. using the wave-induced responses of ships considered as sailing wave buoys (SAWB).

To credit the author, users are encouraged to use below reference:

```
Mounet, R. E. G., & Nielsen, U. D. NetSSE: An open-source Python package
for network-based sea state estimation from ships, buoys, and other
observation platforms (version 2.0). Technical University of Denmark,
GitLab. July 2024. https://doi.org/10.11583/DTU.26379811.
```

*Last updated on 11-07-2024 by R.E.G. Mounet*

## Functions

### netsse.analys.sawb.specres

`netsse.analys.sawb.`**`specres`**(*Rxy*, *TRF*, *freq*, *beta*, *opt=None*)

Computes an estimate of the sea state using the spectral-residual method from Brodtkorb et al. (2017) and Nielsen et al. (2018).

The heave, roll, and pitch response cross-spectra are used as input. Using the motion transfer functions of the vessel, the wave spectrum is estimated through iteration.

> **Warning**
>
> Long-crested wave conditions are assumed in this sea state estimation method.

> **Note**
>
> This Python code is based on a MATLAB/Simulink implementation by Astrid H. Brodtkorb.

**Parameters**

- **Rxy** (*array_like of shape (Nfreq,6)*) – The complex-valued response spectra, given as in: `Rxy = [heaveheave, rollroll, pitchpitch, heaveroll, heavepitch, rollpitch]`.
- **TRF** (*array_like of shape (Nfreq,3*Nbeta)*) – The transfer functions of the ship, in heave, roll, and pitch, concatenated along the second axis as in: `TRF = [heave_TF, roll_TF,pitch_TF]`

> **Note**
>
> The phase of the complex transfer functions is not important in this implementation, and the amplitude alone can be provided without this affecting the estimation results.

- **freq** (*array_like of shape (Nfreq,)*) – The frequencies of the response spectra and the transfer functions (should match).
- **beta** (*array_like of shape (Nbeta,)*) – The discretized heading angles [deg] at which the transfer functions are known.

> **Tip**
>
> For a port-starboard symmetric ship, directions from 0 deg to 180 deg only can be considered to lower the computational cost.

- **hij** (*array_like of shape (6,)*) – The iteration gains of the algorithm.
- **opt** (*dict, optional*) – Optional parameters controlling the SSE calculation. Available options are:
  - **'maxiter'**
    [int, or array_like of shape (6,)] Maximum number of iterations (default: 50).
  - **'tolCoef'**
    [float, or array_like of shape (6,)] Tolerance coefficient (default: 0.1).
  - **'gainFact'**
    [float, or array_like of shape (6,)] Gain factor, as a fraction of the maximum gain value (default: 0.5).
  - **'weights'**
    [float, or array_like of shape (6,)] Weights given to each response in the calculation of the final spectrum estimate (default: equal weight for each response, i.e. wij = 1/6).

> **Note**
>
> *array_like* entries of the dictionary can be input for a response-specific option. In such case, the array elements must be provided in the same order as for the response spectra.

**Returns**

- **S_wave** (*array_like of shape (Nfreq,)*) – The estimated 1-D wave spectrum.
- **beta_est** (*float*) – The estimated relative wave heading [degrees].
- **num_it** (*array_like of shape (6,)*) – The average number of iterations used per heading angle, for the individual response pairs organised as: `num_it = [it3, it4, it5, it34, it35, it45]`.

### References


1. Brodtkorb, A. H., Nielsen, U. D., & Sørensen, A. J. (2018). Online wave estimation using vessel motion measurements. IFAC-PapersOnLine, 51(29), 244–249. https://doi.org/10.1016/j.ifacol.2018.09.510
2. Brodtkorb, A. H., Nielsen, U. D., & Sørensen, A. J. (2018). Sea state estimation using vessel response in dynamic positioning. Applied Ocean Research, 70, 76–86. https://doi.org/10.1016/j.apor.2017.09.005
3. Nielsen, U. D., Brodtkorb, A. H., & Sørensen, A. J. (2018). A brute-force spectral approach for wave estimation using measured vessel motions. Marine Structures, 60, 101–121. https://doi.org/10.1016/j.marstruc.2018.03.011
4. Brodtkorb, A. H., & Nielsen, U. D. (2023). Automatic sea state estimation with online trust measure based on ship response measurements. Control Engineering Practice, 130. https://doi.org/10.1016/j.conengprac.2022.105375


### Example

```
>>> S_wave, beta_est, num_it = specres(Rxy, TRF, freq, beta, opt=None)
```

| | |
|---|---|
| *specres*(Rxy, TRF, freq, beta[, opt]) | Computes an estimate of the sea state using the spectral-residual method |

## netsse.analys.spec

Functions to compute sea state parameters from **wave spectra**.

### Copyright (C) 2024 Technical University of Denmark, R.E.G. Mounet



To credit the author, users are encouraged to use below reference:

```
Mounet, R. E. G., & Nielsen, U. D. NetSSE: An open-source Python package
for network-based sea state estimation from ships, buoys, and other
observation platforms (version 2.0). Technical University of Denmark,
GitLab. July 2024. https://doi.org/10.11583/DTU.26379811.
```

*Last updated on 11-07-2024 by R.E.G. Mounet*

## Functions

### netsse.analys.spec.spec1d_to_params

netsse.analys.spec.**spec1d_to_params**(*S*, *freq*, *unit_freq='rad/s'*, *smooth_Tp=False*)

Computes the main sea state parameters from a given 1-D wave spectrum.

**Parameters**

- **S** (*array_like*) – 1-D wave spectrum. If S has more than one dimension, then one axis must have a length Nfreq.
- **freq** (*array_like of shape (Nfreq,)*) – Set of discretized frequencies.
- **unit_freq** (*{'rad/s','Hz'}, optional*) – Unit of the frequencies:
  - **'rad/s' :**
    The variable freq denotes the circular frequencies in radians per second.
  - **'Hz' :**
    The 1-D wave spectrum is expressed as a function of frequency in Hertz.
- **smooth_Tp** (*bool, default False*) – Specify whether the output Tp value should be smoothened by fitting a polynomial of order 3 in the vicinity of the spectral peak and estimating the peak period through the fitted polynomial.

**Returns**

- **m** (*array_like of shape (6,...,)*) – Spectral moments $[m_{-1}, m_0, m_1, m_2, m_3, m_4]$.
- **Hm0** (*float, or array_like*) – Spectral significant wave height [m].
- **Tp** (*float, or array_like*) – Peak wave period [s].
- **Tm01** (*float, or array_like*) – Mean wave period [s].
- **Tm02** (*float, or array_like*) – Zero up-crossing period [s].
- **Tm24** (*float, or array_like*) – Mean crest period [s].
- **TE** (*float, or array_like*) – Mean energy period [s].
- **Sm02** (*float, or array_like*) – Significant wave steepness [-].
- **epsilon** (*float, or array_like*) – Spectral bandwidth [-].
- **Qp** (*float, or array_like*) – Goda's peakedness parameter [-].

**See also**

*spec2d_to_params*
Computes a set of overall parameters characterizing the sea state from the 2-D wave spectrum.

### Example

```
>>> m, Hm0, Tp, Tm01, Tm02, Tm24, TE, Sm02, epsilon, Qp =
...     spec1d_to_params(S,freq,unit_freq='rad/s',smooth_Tp=False)
```

## netsse.analys.spec.spec2d_to_params

netsse.analys.spec.**spec2d_to_params**(*S2D*, *freq*, *theta*, *unit_theta='deg'*, *smooth_peak=False*)

Computes a set of overall parameters characterizing the sea state from the 2-D wave spectrum.

**Parameters**

- **S2D** (*array_like of shape (...,Nf,Ntheta,...)*) – Directional wave spectrum [m^2.s/unit_theta], as a function of frequency in Hertz and wave direction in specified unit (unit_theta).
- **freq** (*array_like of shape (Nf,)*) – Vector of (discretized) wave frequencies [Hz] (must include $f = 0$ Hz).
- **theta** (*array_like of shape (Ntheta,)*) – Vector of wave headings [the unit is specified in unit_theta].

> **Attention**
>
> theta must be in a wrapped format, i.e., corresponding to [0,360] deg.

- **unit_theta** (*{'deg','rad'}, optional*) – Unit of the wave directions. This applies for the expression of the wave spectrum too. Can be 'deg' (for degrees), or 'rad' (for radians). 'deg' is the default.
- **smooth_peak** (*bool, optional*) – Specify whether the output Tp and theta_p values should be smoothened by fitting a polynomial of order 3 in the vicinity of the spectral peak and estimating the peak period and direction through the fitted polynomial. The default is False.

**Returns**

- **m** (*array_like of shape (6,)*) – Spectral moments $[m_{-1}, m_0, m_1, m_2, m_3, m_4]$.
- **Hm0** (*float, or array_like*) – Spectral significant wave height [m].
- **Tp** (*float, or array_like*) – Peak wave period [s].
- **Tm01** (*float, or array_like*) – Mean wave period [s].
- **Tm02** (*float, or array_like*) – Zero up-crossing period [s].
- **Tm24** (*float, or array_like*) – Mean crest period [s].
- **TE** (*float, or array_like*) – Mean energy period [s].
- **Sm02** (*float, or array_like*) – Significant wave steepness [-].
- **epsilon** (*float, or array_like*) – Spectral bandwidth [-].
- **Qp** (*float, or array_like*) – Goda's peakedness parameter [-].
- **theta_p** (*float, or array_like*) – Peak wave direction [deg].
- **theta_m** (*float, or array_like*) – Mean overall wave direction [deg].

- **sigma_m** (*float, or array_like*) – Mean directional spreading [deg].

**See also**

*spec1d_to_params*
Computes the main sea state parameters from a given 1-D wave spectrum.

### Example

```
>>> m, Hm0, Tp, Tm01, Tm02, Tm24, TE, Sm02, epsilon, Qp, theta_p, theta_m, sigma_m =
...     spec2d_to_params(S2D,freq,theta,unit_theta='deg',smooth_Tp=False)
```

## netsse.analys.spec.spread_dist_to_spec2d

netsse.analys.spec.**spread_dist_to_spec2d**(*Sf*, *D*, *theta*)

Converts the given 1-D wave spectrum and directional spreading function into a 2-D wave spectrum.

**Parameters**

- **Sf** (*array_like of shape (...,Nf,...)*) – One-sided variance spectrum of the waves, as a function of frequency.
- **D** (*array_like of shape (...,Nf,Ntheta,...)*) – Directional spreading function.
- **theta** (*array_like of shape (Ntheta,)*) – Vector of wave headings (with arbitrary unit).

**Returns**

**S2D** – Directional wave spectrum, as a function of frequency and wave direction (with arbitrary units).

**Return type**

array_like of shape (…,Nf,Ntheta,…)

**See also**

*spec2d_to_spread_dist*
Converts the given 2-D wave spectrum into a directional spreading function and a 1-D wave spectrum.

### Example

```
>>> S2D = spread_dist_to_spec2d(Sf,D,theta)
```

### netsse.analys.spec.spec2d_to_spread_dist

netsse.analys.spec.**spec2d_to_spread_dist**(*S2D*, *theta*)

Converts the given 2-D wave spectrum into a directional spreading function and a 1-D wave spectrum.

**Parameters**

- **S2D** (*array_like of shape (...,Nf,Ntheta,...)*) – Directional wave spectrum, as a function of frequency and wave direction (with arbitrary units).
- **theta** (*array_like of shape (Ntheta,)*) – Vector of wave headings (with arbitrary unit).

**Returns**

- **Sf** (*array_like of shape (...,Nf,...)*) – One-sided variance spectrum of the waves, as a function of frequency.
- **D** (*array_like of shape (...,Nf,Ntheta,...)*) – Directional spreading function.

**See also**

*spread_dist_to_spec2d*
Converts the given 1-D wave spectrum and directional spreading function into a 2-D wave spectrum.

#### Example

```
>>> Sf, D = spec2d_to_spread_dist(S2D,theta)
```

| | |
|---|---|
| *spec1d_to_params*(S, freq[, unit_freq, smooth_Tp]) | Computes the main sea state parameters from a given 1-D wave spectrum. |
| *spec2d_to_params*(S2D, freq, theta[, unit_theta, ...]) | Computes a set of overall parameters characterizing the sea state from the |
| *spread_dist_to_spec2d*(Sf, D, theta) | Converts the given 1-D wave spectrum and directional spreading function |
| *spec2d_to_spread_dist*(S2D, theta) | Converts the given 2-D wave spectrum into a directional spreading function |

### netsse.model

A set of functions for computing linear hydrodynamic **models** for simplified geometries of vessels.

### Submodules

### netsse.model.ship

Closed-form expressions of the linear wave-to-motion transfer functions for a box-shaped uniformly-loaded **monohull ship**.

### Functions

#### netsse.model.ship.heaveCF

netsse.model.ship.**heaveCF**(*om0*, *beta_deg*, *U*, *L*, *B0*, *T*, *C_B=1*)

Computes the heave transfer function amplitude, using the closed-form expressions presented in Jensen et al. (2004).

**Parameters**

- **om0** (*array_like of shape (Nom0,)*) – Vector of absolute wave frequencies [rad/s].
- **beta_deg** (*array_like of shape (Nbeta,)*) – Vector of heading angles [deg].
- **U** (*float*) – Vessel forward speed [m/s].
- **L** (*float*) – Length of the ship [m].
- **B0** (*float*) – Breadth of the ship [m].
- **T** (*float*) – Draft of the ship [m].
- **C_B** (*float, default 1*) – Block coefficient of the ship [-].

**Returns**

**heave** – Coefficients of the heave transfer function amplitude [m/m].

**Return type**

numpy.ndarray of shape (Nom0,Nbeta)

**See also**

*pitchCF*, *rollCF*

#### References

Jensen, J. J., Mansour, A. E., & Olsen, A. S. (2004). Estimation of ship motions using closed-form expressions. Ocean Engineering, 31(1), 61–85. https://doi.org/10.1016/S0029-8018(03)00108-2

#### Example

```
>>> heave = heaveCF(om0,beta_deg,U,L,B0,T,C_B=1)
```

#### netsse.model.ship.pitchCF

netsse.model.ship.**pitchCF**(*om0*, *beta_deg*, *U*, *L*, *B0*, *T*, *C_B=1*)

Computes the pitch transfer function amplitude, using the closed-form expressions presented in Jensen et al. (2004).

**Parameters**

- **om0** (*array_like of shape (Nom0,)*) – Vector of absolute wave frequency [rad/s].
- **beta_deg** (*array_like of shape (Nbeta,)*) – Vector of heading angle [deg].
- **U** (*float*) – Vessel forward speed [m/s].

- **L** (*float*) – Length of the ship [m].
- **B0** (*float*) – Breadth of the ship [m].
- **T** (*float*) – Draft of the ship [m].
- **C_B** (*float, default 1*) – Block coefficient of the ship [-].

**Returns**
**pitch** – Coefficients of the pitch transfer function amplitude [rad/m].

**Return type**
numpy.ndarray of shape (Nom0,Nbeta)

**See also**

*heaveCF*, *rollCF*

### References

Jensen, J. J., Mansour, A. E., & Olsen, A. S. (2004). Estimation of ship motions using closed-form expressions. Ocean Engineering, 31(1), 61–85. https://doi.org/10.1016/S0029-8018(03)00108-2

### Example

```
>>> pitch = pitchCF(om0,beta_deg,U,L,B0,T,C_B=1)
```

## netsse.model.ship.rollCF

netsse.model.ship.**rollCF**(*om0*, *beta_deg*, *U*, *L*, *B0*, *T*, *C_B*, *C_WP*, *GM_T*, *kappa*, *mu*, *T_N=0*)

Computes the roll transfer function amplitude, using the closed-form expressions presented in Jensen et al. (2004).

**Parameters**

- **om0** (*array_like of shape (Nom0,)*) – Vector of absolute wave frequencies [rad/s].
- **beta_deg** (*array_like of shape (Nbeta,)*) – Vector of heading angles [deg].
- **U** (*float*) – Vessel forward speed [m/s].
- **L** (*float*) – Length of the ship [m].
- **B0** (*float*) – Breadth of the ship [m].
- **T** (*float*) – Draft of the ship [m].
- **C_B** (*float*) – Block coefficient of the ship [-].
- **C_WP** (*float*) – Waterplane area coefficient of the ship [-].
- **GM_T** (*float*) – Transverse metacentric height [m].
- **kappa** (*float*) – Custom parameter (chosen between 0 and C_WP) representing the ratio between the length of the aft beam and the whole ship length in the simplified ship model for roll.
- **mu** (*float, default 0*) – Ratio between added viscous damping and critical damping.

- **T_N** (*float, optional*) – Roll natural period [s].

> **Note**
>
> If not specified as input, T_N is calculated using empirical formulas from ADA147598.

**Returns**
**roll** – Coefficients of the roll transfer function amplitude [rad/m].

**Return type**
numpy.ndarray of shape (Nom0,Nbeta)

> **See also**
>
> *heaveCF*, *pitchCF*

### References


1. Jensen, J. J., Mansour, A. E., & Olsen, A. S. (2004). Estimation of ship motions using closed-form expressions. Ocean Engineering, 31(1), 61–85. https://doi.org/10.1016/S0029-8018(03)00108-2
2. ADA147598, 1973. Code of Safety for Fishermen and Fishing Vessels. Part B. Safety and Health Requirements for the Construction and Equipment of Fishing Vessels. Inter-Governmental Maritime Consultative Organization, London, England.


### Example

```
>>> roll = rollCF(om0,beta_deg,U,L,B0,T,C_B,C_WP,GM_T,kappa,mu,T_N=0)
```

| | |
|---|---|
| *heaveCF*(om0, beta_deg, U, L, B0, T[, C_B]) | Computes the heave transfer function amplitude, using the closed-form |
| *pitchCF*(om0, beta_deg, U, L, B0, T[, C_B]) | Computes the pitch transfer function amplitude, using the closed-form |
| *rollCF*(om0, beta_deg, U, L, B0, T, C_B, C_WP, GM_T, ...) | Computes the roll transfer function amplitude, using the closed-form |

## netsse.simul

A set of functions for **simulating** linear irregular waves and first-order motion responses of ships and other structures.

### Submodules

### netsse.simul.ship_resp

Numerical simulation of **ship responses**.

## Classes

### netsse.simul.ship_resp.MultiThread

### netsse.simul.ship_resp.MultiThread.run

netsse.simul.ship_resp.MultiThread.**run**()

Method representing the thread's activity.

You may override this method in a subclass. The standard run() method invokes the callable object passed to the object's constructor as the target argument, if any, with sequential and keyword arguments taken from the args and kwargs arguments, respectively.

**class** netsse.simul.ship_resp.**MultiThread**(*name*)

Bases: threading.Thread

This constructor should always be called with keyword arguments. Arguments are:

*group* should be None; reserved for future extension when a ThreadGroup class is implemented.

*target* is the callable object to be invoked by the run() method. Defaults to None, meaning nothing is called.

*name* is the thread name. By default, a unique name is constructed of the form "Thread-N" where N is a small decimal number.

*args* is a list or tuple of arguments for the target invocation. Defaults to ().

*kwargs* is a dictionary of keyword arguments for the target invocation. Defaults to {}.

If a subclass overrides the constructor, it must make sure to invoke the base class constructor (Thread.__init__()) before doing anything else to the thread.

**name**

A string used for identification purposes only.

It has no semantics. Multiple threads may be given the same name. The initial name is set by the constructor.

## Methods

| | |
|---|---|
| *run*() | Method representing the thread's activity. |

| | |
|---|---|
| *MultiThread* | This constructor should always be called with keyword arguments. Arguments are: |

## Functions

### netsse.simul.ship_resp.simul_ship_resp

netsse.simul.ship_resp.**simul_ship_resp**(*S2D*, *freq_wave*, *betas_wave*, *U=0*, *RespStr='Waves'*, *max_threads=0*, *freq_TRF=[0]*, *betas_TRF=0*, *TRF_amps=0*, *TRF_phases=0*, *NoTs=10*, *fs=10*, *T=30 * 60*, *Nomega=500*, *AddNoise=False*, *snr=20*)

Generates several time histories of the waves encountered by a ship and, optionally, the wave-induced vessel motion responses (heave, roll, pitch).

A 2D wave spectrum must be given as input. If ship response is to be simulated too, then the wave-to-motion transfer functions of the vessel must also be provided.

**Parameters**

- **S2D** (*array_like of shape (Nf_wave,Nmu)*) – 2-D wave spectrum characterizing the sea state [m^2.s.s/rad].
- **freq_wave** (*array_like of shape (Nf_wave,)*) – Vector of wave frequencies [Hz] in which the wave spectrum is expressed.
- **betas_wave** (*array_like of shape (Nmu,)*) – Vector of the *relative* wave headings [deg] for the wave spectrum.

> **Note**
>
> The relative wave headings are defined as: 0 deg: following sea; 90 deg: beam waves from port side; 180 deg: head sea; 270 deg: beam waves from starboard side.

- **U** (*float, default 0*) – Vessel speed [m/s].
- **RespStr** (*{'Waves','Motions'}, optional*) – Use `'Motions'` if vessel motions in waves are also to be simulated (in addition to the waves). Otherwise, use `'Waves'` and only the wave surface elevation will be simulated. The default is `'Waves'`.
- **max_threads** (*int, default 0*) – Indicates to maximum number of threads to be used to parallelize the execution.

> **Attention**
>
> A large value for `max_threads` should only be used if the program is run on a high-performance computer. The code execution risks crashing otherwise.

- **freq_TRF** (*array_like of shape (Nf_TRF,), default [0,]*) – Vector of wave frequencies [Hz] in which the transfer functions are expressed.
- **betas_TRF** (*array_like of shape (Nbeta,), default 0*) – Vector of the relative wave headings [deg] indicating the direction of propagation of the energy (relative to vessel heading) for the transfer functions.
- **TRF_amps** (*array_like of shape (Nf_TRF, Nbeta, 3), optional*) – The amplitude of the transfer functions for heave, roll, and pitch, respectively. The default is 0.
- **TRF_phases** (*array_like of shape (Nf_TRF, Nbeta), optional*) – The phase of the transfer functions for heave, roll, and pitch, respectively. The default is 0.

- **NoTs** (*int, default 10*) – Number of time series to be generated.
- **fs** (*float, default 10*) – Sampling frequency [Hz] for the generated time histories.
- **T** (*float, optional*) – Duration [s] of the time histories. The default is 30*60 = 1800 s.
- **Nomega** (*int, default 500*) – Number of wave components used for each direction to generate the time histories.
- **AddNoise** (*bool, default False*) – A boolean that indicates whether random noise (Gaussian white noise) is added to be added the motion time histories.
- **snr** (*float, default 20*) – The signal-to-noise ratio to be used for addition of noise.

**Returns**

- **time** (*array_like of shape (T*fs,).*) – Vector of time [s].
- **wavet** (*array_like of shape (NoTs,T*fs).*) – Wave elevation [m] time series.
- **heavet, rollt, pitcht** (*array_like of shape (NoTs,T*fs).*) – Heave, roll and pitch time series.

> **Note**
>
> `heavet`, `rollt`, `pitcht` are returned if and only if RespStr = `'Motions'`

> **See also**
>
> ***sum_components***
> Performs the summation of the sine components to simulate in the time domain the encountered wave elevation or the wave-induced responses on a ship.

### Example

```
>>> # Wave simulations
>>> time, wavet =
...     simul_ship_resp(S2D,freq_wave,betas_wave,U,'Waves',0,NoTs=10,fs=10,T=30*60,
...                     Nomega=500,AddNoise=True,snr=20)
>>> # Motion simulations
>>> time, wavet, heavet, rollt, pitcht =
...     simul_ship_resp(S2D,freq_wave,betas_wave,U,'Motions',0,freq_TRF,betas_TRF,
...                     TRF_amps,TRF_phases,NoTs,fs,T,Nomega,AddNoise=True,snr=20)
```

## netsse.simul.ship_resp.sum_components

netsse.simul.ship_resp.**sum_components**(*aw*, *V_seed*, *W_seed*, *epsilon_seed*, *om_enc*, *time*, *seed*, *result*, *TRF_amps=0*, *TRF_phases=0*, *RespStr='Waves'*, *AddNoise=False*, *snr=20*)

Performs the summation of the sine components to simulate in the time domain the encountered wave elevation or the wave-induced responses on a ship.

**Parameters**

- **aw** (*array_like of shape (Nfreq,Nbeta,1)*) – Amplitude of the wave components.
- **V_seed** (*array_like of shape (Nfreq,Nbeta,1)*) – Standard normal distributed variables with mean 0 and std. 1.
- **W_seed** (*array_like of shape (Nfreq,Nbeta,1)*) – Standard normal distributed variables with mean 0 and std. 1.
- **epsilon_seed** (*array_like of shape (Nfreq,Nbeta,1)*) – Random phases [rad] of the wave components. Those are uniformly distributed variables between 0 and $2\pi$.
- **om_enc** (*array_like of shape (Nfreq,1,1)*) – Vector of encounter wave frequencies [rad/s].
- **time** (*array_like of shape (1,1,Nt)*) – Vector of time [s].
- **seed** (*int*) – Index of the seed number.
- **result** (*array_like of shape (Nseed,Nt)*) – Anterior value of the vector of wave/response sequence.

**Note**

Consult the documentation of the *netsse.simul.ship_resp.simul_ship_resp()* function for information on the other parameters.

**Returns**

**result** – Updated value of the vector of wave/response sequence.

**Return type**

array_like of shape (Nseed,Nt)

**See also**

*simul_ship_resp*, *process_queue*

### netsse.simul.ship_resp.process_queue

netsse.simul.ship_resp.**process_queue**()

A function to process the FIFO queue.

This basic function runs the function *netsse.simul.ship_resp.sum_components()* with the queued arguments, until the queue is empty, after which it signals that the task is completed.

**See also**

*MultiThread*, *sum_components*

| | |
|---|---|
| *simul_ship_resp*(S2D, freq_wave, betas_wave[, U, ...]) | Generates several time histories of the waves encountered by a ship and, |
| *sum_components*(aw, V_seed, W_seed, epsilon_seed, ...) | Performs the summation of the sine components to simulate in the time domain |
| *process_queue*() | A function to process the FIFO queue. |

### netsse.tools

A set of functions that are used in other parts of the NetSSE package for the modelling of environmental conditions and other (mathematical) **tools**.

### Copyright (C) 2024 Technical University of Denmark, R.E.G. Mounet

*This code is part of the NetSSE software.*

NetSSE is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

NetSSE is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see https://www.gnu.org/licenses/.

To credit the author, users are encouraged to use below reference:

```
Mounet, R. E. G., & Nielsen, U. D. NetSSE: An open-source Python package
for network-based sea state estimation from ships, buoys, and other
observation platforms (version 2.0). Technical University of Denmark,
GitLab. July 2024. https://doi.org/10.11583/DTU.26379811.
```

*Last updated on 11-07-2024 by R.E.G. Mounet*

### Submodules

### netsse.tools.envir_cond

Relevant functions for **environmental conditions**.

### Copyright (C) 2024 Technical University of Denmark, R.E.G. Mounet

*This code is part of the NetSSE software.*

NetSSE is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

NetSSE is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see https://www.gnu.org/licenses/.

To credit the author, users are encouraged to use below reference:

```
Mounet, R. E. G., & Nielsen, U. D. NetSSE: An open-source Python package
for network-based sea state estimation from ships, buoys, and other
observation platforms (version 2.0). Technical University of Denmark,
GitLab. July 2024. https://doi.org/10.11583/DTU.26379811.
```

*Last updated on 11-07-2024 by R.E.G. Mounet*

## Functions

### netsse.tools.envir_cond.JONSWAP_DNV

netsse.tools.envir_cond.**JONSWAP_DNV**(*Tp*, *Hs*, *omega*, *gamma='standard'*, *h=0*)

Computes the JONSWAP spectrum corresponding to the input sea state parameters.

The JONSWAP spectrum is formulated as a modification of a Pierson-Moskowitz spectrum for a developing sea state in a fetch limited situation.

**Parameters**

- **Tp** (*float*) – Peak period [s].
- **Hs** (*float*) – Significant wave height [m].
- **omega** (*array_like of shape (Nfreq,)*) – Vector of angular frequencies [rad/s].
- **gamma** (*{'standard','DNV',float}, optional*) – Peak shape parameter [-]. The value can be user-provided as a float. Alternatively, if `'standard'` is input, then `gamma` will take the standard value of 3.3., while a value `'DNV'` as input leads to following the procedure 3.5.5.5 described in DNV-RP-C205.

  **Tip**

  Use `gamma = 1` to output a standard Pierson-Moskowitz spectrum.

- **h** (*float, default 0*) – Water depth [m]. If `h` is specified as input argument, then the output JONSWAP spectrum is corrected to account for finite water depth, becoming a standard TMA spectrum as per Bouws et al. (1985).

**Returns**

**S_J** – Standard wave spectrum [m^2.s/rad].

**Return type**

array_like of shape (Nfreq,)

#### References


1. DNV-RP-C205, "Environmental Conditions and Environmental Loads, April 2007.
2. Bouws, E., Gunther, H., Rosenthal, W., & Vincent, C. L. (1985). *Similarity of the wind wave spectrum in finite depth water. 1. Spectral form.* Journal of Geophysical Research-Oceans, 90(NC1), 975–986. https://doi.org/10.1029/JC090iC01p00975


**See also**

*lin_disprel*
A fast and accurate approximation of the linear wave dispersion relationship in finite water depth.

#### Example

```
>>> S_J = JONSWAP_DNV(Tp,Hs,omega,gamma='standard',h=0)
```

### netsse.tools.envir_cond.lin_disprel

netsse.tools.envir_cond.**lin_disprel**(*omega*, *h*)

Computes the wavenumbers from wave angular frequencies in finite water depth.

This function implements a fast and accurate approximation of the linear wave dispersion relationship in finite water depth.

**Parameters**

- **omega** (*array_like or float*) – Wave angular frequencies [rad/s].
- **h** (*float*) – Mean water depth [m]

**Returns**

**k** – Wavenumbers [1/m]

**Return type**

array_like or float

#### Example

```
>>> k = lin_disprel(omega,h)
```

### netsse.tools.envir_cond.wavespec1dto2d

netsse.tools.envir_cond.**wavespec1dto2d**(*S1d*, *theta*, *theta0*, *s*)

Transforms a 1-D wave spectrum into a 2-D wave spectrum.

A cosine-2s function is used for the directional spreading distribution.

**Parameters**

- **S1d** (*array_like of shape (Nfreq,)*) – 1-D wave spectrum.
- **theta** (*array_like of shape (Ntheta,)*) – Vector of wave headings [deg] at which the 2-D spectrum must be computed.
- **theta0** (*float*) – Mean wave direction [deg].
- **s** (*float*) – Spreading parameter [-], related to the exponent of the cosine function.

**Returns**

**S2d** – 2-D wave spectrum, as a function of both the frequency and the wave heading [deg].

**Return type**

array_like of shape (Nfreq,Ntheta)

### References

Naess, Arvid, and Torgeir Moan. 2010. Stochastic Dynamics of Marine Structures. Stochastic Dynamics of Marine Structures. Vol. 9780521881555. Cambridge University Press. https://doi.org/10.1017/CBO9781139021364.

### Example

```
>>> S2d = wavespec1dto2d(S1d,theta,theta0,s)
```

| | |
|---|---|
| *JONSWAP_DNV*(Tp, Hs, omega[, gamma, h]) | Computes the JONSWAP spectrum corresponding to the input sea state |
| *lin_disprel*(omega, h) | Computes the wavenumbers from wave angular frequencies in finite water depth. |
| *wavespec1dto2d*(S1d, theta, theta0, s) | Transforms a 1-D wave spectrum into a 2-D wave spectrum. |

## netsse.tools.misc_func

**Miscellaneous functions** for NetSSE.

### Copyright (C) 2024 Technical University of Denmark, R.E.G. Mounet



To credit the author, users are encouraged to use below reference:

```
Mounet, R. E. G., & Nielsen, U. D. NetSSE: An open-source Python package
for network-based sea state estimation from ships, buoys, and other
observation platforms (version 2.0). Technical University of Denmark,
GitLab. July 2024. https://doi.org/10.11583/DTU.26379811.
```

*Last updated on 12-07-2024 by R.E.G. Mounet*

## Functions

### netsse.tools.misc_func.wrap

netsse.tools.misc_func.**wrap**(*U*, *axis=1*, *add=0*)

Wraps an array around one of its dimensions.

**Parameters**

- **U** (*array_like of shape (n,m) or (n,)*) – Input array to be wrapped around.
- **axis** (*{1,0}, optional*) – If U is a 2d-array, the axis of U along which to wrap. The default is 1.
- **add** (*float or array_like, default 0*) – Term that is added to the repeated part of U before concatenation.

**Returns**

**U_wrap** – Extended array, where the first column (respectively, row) of U has been repeated by concatenation at the end of U.

**Note**

- If U is a 2d-array, then the output array has a shape of (n+1*(axis==0), m+1*(axis==1)).
- If U is a 1d-array, then the output array has a shape of (n+1,). The first element of U has been repeated at the end.

**Return type**

array_like

**See also**

*pad*
Adds zeros to an array along one of its dimensions.

### Example

```
>>> U_wrap = wrap(U,axis=1,add=0)
```

### netsse.tools.misc_func.pad

netsse.tools.misc_func.**pad**(*U*, *axis=1*, *add=0*)

Adds zeros to an array along one of its dimensions.

**Parameters**

- **U** (*array_like of shape (n,m) or (n,)*) – Input array to be padded with zeros at the beginning.
- **axis** (*{1,0}, optional*) – If U is a 2d-array, the axis of U along which to pad. The default is 1.
- **add** (*float or array_like, default 0*) – Term that is added to the concatenated zeros.

**Returns**

**U_pad** – Extended array, where U has been padded with a first column (respectively, row) of zeros (plus an optional added term).

**Note**

- If U is a 2d-array, then the output array U_pad has a new shape of (n+1*(axis==0), m+1*(axis==1)).
- If U is a 1d-array, then the output array U_pad has a new shape of (n+1,). A zero element has been added at the beginning.

**Return type**
array_like

**See also**

*wrap*
Wraps an array around one of its dimensions.

#### Example

```
>>> U_wrap = pad(U,axis=1,add=0)
```

### netsse.tools.misc_func.re_range

netsse.tools.misc_func.**re_range**(*theta*, *start=0*, *unit='deg'*)

Converts angles to a new range (of extent 360 degrees).

**Parameters**

- **theta** (*array_like*) – Array of angles to be rearranged.
- **start** (*float, default 0*) – Lower boundary of the range that should contain the new values of theta. The upper boundary will be set automatically to start + 360 degrees.

- **unit** (*{'deg','rad'}, optional*) – Specifies the unit of `theta` and `start` as `'rad'` or `'deg'`. The default is `'deg'`.

**Returns**

**theta_rerange** – Array of angles where the elements of `theta` have been converted to the new range `[start, start+360)` degrees.

**Return type**

array_like

**See also**

*ang_diff*
Calculates the signed difference between two angles.

### Example

```
>>> theta_rerange = re_range(theta,start=0,unit='deg')
```

## netsse.tools.misc_func.ang_diff

netsse.tools.misc_func.**ang_diff**(*alpha*, *beta*, *unit='deg'*)

Calculates the signed difference between two angles.

**Parameters**

- **alpha** (*float or array_like*) – Input angle as a scalar value or as an array of angles.
- **beta** (*float or array_like*) – Input angle as a scalar value or as an array of angles.
- **unit** (*{'deg','rad'}, optional*) – Specifies the unit of the angles as `'rad'` or `'deg'`. The default is `'deg'`.

**Returns**

**delta** – The output difference between alpha and beta in the range [-180,180] degrees.

**Note**

This function subtracts `alpha` from `beta`, i.e., computes the signed difference `beta-alpha`, taking into account the circularity at 360 degrees.

**Return type**

float or array_like

**See also**

*re_range*
Converts angles to a new range of values.

*ang_mean*
Calculates the mean angle from an array of angular data.

*ang_std*
Calculates the circular standard deviation for an array of angular data.

### Example

```
>>> delta = ang_diff(alpha, beta, unit='deg')
```

## netsse.tools.misc_func.ang_mean

netsse.tools.misc_func.**ang_mean**(*angles*, *unit='deg'*)

Calculates the mean angle from an array of angular data.

This function properly handles the circularity of angles at 360 degrees.

**Parameters**

- **angles** (*array_like*) – Array of angles.
- **unit** (*{'deg','rad'}, optional*) – Specifies the unit of the angles as `'rad'` or `'deg'`. The default is `'deg'`.

**Returns**
**mean_a** – Mean angle in the range [-180,180) degrees.

**Return type**
float

**See also**

*ang_diff*
Calculates the signed difference between two angles.

*ang_std*
Calculates the circular standard deviation of an array of angular data.

### Example

```
>>> mean_angle = ang_mean(angles, unit='deg')
```

## netsse.tools.misc_func.ang_std

netsse.tools.misc_func.**ang_std**(*angles*, *unit='deg'*)

Calculates the circular standard deviation of an array of angular data.

This function properly handles the circularity of angles at 360 degrees. The standard "two-pass" computation is implemented.

**Parameters**

- **angles** (*array_like*) – Array of angles.

- **unit** (*{'deg','rad'}, optional*) – Specifies the unit of the angles as `'rad'` or `'deg'`. The default is `'deg'`.

**Returns**
**std_a** – Circular standard deviation.

**Return type**
float

**See also**

*ang_diff*
Calculates the signed difference between two angles.

*ang_mean*
Calculates the mean angle from an array of angular data.

### Example

```
>>> std_angle = ang_std(angles, unit='deg')
```

## netsse.tools.misc_func.areSame_vec

netsse.tools.misc_func.**areSame_vec**(*a*, *b*)

Checks whether two vectors are identical, element-wise.

**Parameters**

- **a** (*1d-array*) – Input vector.
- **b** (*1d-array*) – Input vector.

**Returns**
`False` for `a!=b`, `True` for `a=b`.

**Return type**
bool

**Note**

If `a` and `b` do *not* have the same shape, then `False` is returned.

**See also**

*areSame_mat*
Checks whether two matrices are identical, element-wise.

### Example

```
>>> areSame_vec(a,b)
```

## netsse.tools.misc_func.areSame_mat

netsse.tools.misc_func.**areSame_mat**(*A*, *B*)

Checks whether two matrices are identical, element-wise.

**Parameters**

- **A** (*2d-array*) – Input matrix.
- **B** (*2d-array*) – Input matrix.

**Returns**

`False` for `A!=B`, `True` for `A=B`.

**Return type**

bool

**Note**

If `A` and `B` do *not* have the same shape, then `False` is returned.

**See also**

*areSame_vec*
Checks whether two vectors are identical, element-wise.

### Example

```
>>> areSame_mat(A,B)
```

## netsse.tools.misc_func.weighted_std

netsse.tools.misc_func.**weighted_std**(*a*, *w*, *axis=None*, *ddof=1*)

Computes the weighted standard deviation of some data.

The weighted standard deviation is a measure of the spread of a distribution of the array elements from the mean, where some of the elements are more significant than others. The weighted standard deviation is calculated based on the weighted mean and it attaches more importance to data that have more weight than to data with less weight (National Institute of Standards and Technology, 1996).

**Parameters**

- **a** (*array_like*) – Variable for which the weighted std. must be computed.
- **w** (*array_like*) – Array of weights. All elements must be positive. The array `w` must broadcastable with the array `a`.

- **axis** (*None or int, optional*) – Axis or axes along which the standard deviation is computed. The default is to compute the weighted standard deviation of the flattened array.
- **ddof** (*int, default 0*) – Delta Degrees of Freedom. The divisor used in the calculations is `(N_nz - ddof)/N_nz`, where `N_nz` represents the number of non-zero weights.

**Returns**
**std** – Weighted standard deviation of the data.

**Return type**
array_like

**See also**

*weighted_quantile*
Computes the $q$-th quantiles of weighted data.

### References

National Institute of Standards and Technology, 1996. Formula for the weighted standard deviation. Available at: https://www.itl.nist.gov/div898/software/dataplot/refman2/ch2/weightsd.pdf (Consulted on 06-08-2023).

### Example

```
>>> std = weighted_std(a,w,axis=None,ddof=1)
```

## netsse.tools.misc_func.weighted_quantile

netsse.tools.misc_func.**weighted_quantile**(*x*, *q*, *w*, *a=0.5*, *b=0.5*, *axis=-1*)

Computes the $q$-th quantiles of weighted data.

The quantiles are computed along one of the axes of the input array `x`.

**Parameters**

- **x** (*array_like*) – Input data array for which the quantiles must be computed.
- **q** (*1d-array of shape (Nq,)*) – Sequence of quantiles to compute, which must be between 0 and 1.
- **w** (*array_like*) – Array of weights. The arrays `x` and `w` must be broadcastable.
- **a** (*floats, default 0.5*) – User-defined constant used in the computation of the quantiles. The default is 0.5 to minimize biases (Rogers, 2003).
- **b** (*floats, default 0.5*) – User-defined constant used in the computation of the quantiles. The default is 0.5 to minimize biases (Rogers, 2003).
- **axis** (*integer, default -1*) – Axis of `x` along which the `q`-quantiles are computed.

**Returns**
**v** – Values of the `q`-quantiles computed along the specified `axis` of the input array `x` with weights `w`.

**Return type**
array_like

**See also**

*weighted_std*
Computes the weighted standard deviation of some data.

### References

Rogers, J.W., 2003. Estimating the variance of percentiles using replicate weights, in: Proc. of the Joint Statistical Meetings, Survey Research Methods Section, American Statistical Association. pp. 3525–3532. URL: http://www.asasrms.org/Proceedings/y2003/Files/JSM2003-000742.pdf

### Example

```
>>> v = weighted_quantile(x, q, w, a=0.5, b=0.5, axis=-1)
```

## netsse.tools.misc_func.find_nearest_gridpoint

netsse.tools.misc_func.**find_nearest_gridpoint**(*lat_wps*, *lon_wps*, *lat_grid*, *lon_grid*, *interv=0.01*)

Finds the nearest neighbours to a series of waypoints within a grid of points.

**Note**

The grid points do *not* need to be uniformly spaced in any direction. The grid can also have been rotated of any angle about the vertical direction (e.g., to be aligned with some shoreline).

For a given waypoint at `(lat_wp,lon_wp)`, the nearest-neighbour candidates are first selected within the gridpoints that fall in the geographical area delimited by `lat_wp*[1-interv,1+interv]` in latitude and `lon_wp*[1-interv,1+interv]` in longitude. Then, the distance between the waypoint and the nearest-neighbour candidates is computed. The output neighbour is found as the candidate that minimises this distance.

**Parameters**

- **lat_wps** (*float, or 1d-array of shape (Nwp,)*) – Vector of waypoint latitudes [deg].
- **lon_wps** (*float, or 1d-array of shape (Nwp,)*) – Vector of waypoint longitudes [deg].
- **lat_grid** (*2d-array*) – Matrix defining the latitudes of the grid points [deg].
- **lon_grid** (*2d-array*) – Matrix defining the longitudes of the grid points [deg].
- **interv** (*float, default 0.01*) – Half-width [-] of the interval band fraction in latitude and longitude, used for screening the potential nearest-neighbour candidates.

**Returns**

- **index_nearest** (*numpy.array of shape (Nwp,2)*) – The coordinates in *(latitude,longitude)* of the nearest gridpoints to the waypoints [deg].

- **dist_nearest** (*numpy.array of shape (Nwp,)*) – The distance between the nearest gridpoints and the waypoints [km].

### Example

```
>>> index_nearest, dist_nearest =
...     find_nearest_gridpoint(lat_wps, lon_wps, lat_grid, lon_grid, interv=0.01)
```

## netsse.tools.misc_func.id_generator

netsse.tools.misc_func.**id_generator**(*size=6*, *chars=string.ascii_uppercase + string.digits*)

Randomly generates a cryptographically secure string of a given size.

**Parameters**

- **size** (*int, default 6*) – String size
- **chars** (*str, default 'ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789'*) – List of allowed characters

**Returns**

Output random string

**Return type**

str

### Example

```
>>> id_generator(size=6, chars=string.ascii_uppercase+string.digits)
```

| | |
|---|---|
| *wrap*(U[, axis, add]) | Wraps an array around one of its dimensions. |
| *pad*(U[, axis, add]) | Adds zeros to an array along one of its dimensions. |
| *re_range*(theta[, start, unit]) | Converts angles to a new range (of extent 360 degrees). |
| *ang_diff*(alpha, beta[, unit]) | Calculates the signed difference between two angles. |
| *ang_mean*(angles[, unit]) | Calculates the mean angle from an array of angular data. |
| *ang_std*(angles[, unit]) | Calculates the circular standard deviation of an array of |
| *areSame_vec*(a, b) | Checks whether two vectors are identical, element-wise. |
| *areSame_mat*(A, B) | Checks whether two matrices are identical, element-wise. |
| *weighted_std*(a, w[, axis, ddof]) | Computes the weighted standard deviation of some data. |
| *weighted_quantile*(x, q, w[, a, b, axis]) | Computes the $q$-th quantiles of weighted data. |
| *find_nearest_gridpoint*(lat_wps, lon_wps, lat_grid, ...) | Finds the nearest neighbours to a series of waypoints within a grid |
| *id_generator*([size, chars]) | Randomly generates a cryptographically secure string of a given size. |

### 3.1.2 Submodules

#### netsse.base

**Class definitions** for the NetSSE software.

#### Copyright (C) 2024 Technical University of Denmark, R.E.G. Mounet

*This code is part of the NetSSE software.*

NetSSE is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

NetSSE is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see https://www.gnu.org/licenses/.

To credit the author, users are encouraged to use below reference:

```
Mounet, R. E. G., & Nielsen, U. D. NetSSE: An open-source Python package
for network-based sea state estimation from ships, buoys, and other
observation platforms (version 2.0). Technical University of Denmark,
GitLab. July 2024. https://doi.org/10.11583/DTU.26379811.
```

*Last updated on 11-07-2024 by R.E.G. Mounet*

#### Classes

#### netsse.base.Network

#### netsse.base.Network.add_platform

netsse.base.Network.**add_platform**(**args*, *ignore_msg: bool = False*)

Add a *Platform* instance to the list of platforms in the network.

**Parameters**

- ***args** (*Platform object*) – Platform to be added to the network.
- **ignore_msg** (*bool, default False*) – If *False*, messages are printed to inform about any changes made to the network. If *True*, thoses messages are turned off.

#### Example

```
>>> plaform2 = netsse.base.Platform()
>>> network.add_platform(platform2,ignore_msg=True)
```

### netsse.base.Network.remove_platform

netsse.base.Network.**remove_platform**(**args*, *ignore_msg: bool = False*)

Remove a *Platform* instance from the list of platforms in the network.

**Parameters**

- ***args** (*Platform object or str*) – Platform to be removed from the network. Identified by the corresponding name or *Platform* instance.
- **ignore_msg** (*bool, default False*) – If *False*, messages are printed to inform about any changes made to the network. If *True*, thoses messages are turned off.

#### Example

```
>>> network.remove_platform(platform1,ignore_msg=True)
```

**class** netsse.base.**Network**(*name: str = None*, *nature: str = 'undefined'*, *platforms: list = []*)

Initialises the network.

**Parameters**

- **name** (*str, optional*) – Network name. By default, the name string is generated randomly in the format NetworkXXXX.
- **nature** (*str, default 'undefined'*) – Network nature.
- **platforms** (*list, default []*) – List of platforms (as *Platform* instances).

**See also**

Platform

#### Example

```
>>> plaform1 = netsse.base.Platform()
>>> network = netsse.base.Network(platforms=[plaform1])
```

### Methods

| | |
|---|---|
| *add_platform*(*args[, ignore_msg]) | Add a *Platform* instance to the list of platforms in the network. |
| *remove_platform*(*args[, ignore_msg]) | Remove a *Platform* instance from the list of platforms in the network. |

### netsse.base.Platform

### netsse.base.Platform.add_operation

netsse.base.Platform.**add_operation**(**args*, *ignore_msg: bool = False*)

Add an *Operation* instance to the list of operations of the platform.

**Parameters**

- ***args** (*Operation object*) – Operation to be added to the platform.
- **ignore_msg** (*bool, default False*) – If *False*, messages are printed to inform about any changes made to the platform. If *True*, thoses messages are turned off.

#### Example

```
>>> operation2 = netsse.base.Operation()
>>> platform1.add_operation(operation2,ignore_msg=True)
```

### netsse.base.Platform.remove_operation

netsse.base.Platform.**remove_operation**(**args*, *ignore_msg: bool = False*)

Remove an existing *Operation* instance from the list of operations of the platform.

**Parameters**

- ***args** (*Operation object or str*) – Operation to be removed from the network. Identified by the corresponding identifier or *Operation* instance.
- **ignore_msg** (*bool, default False*) – If *False*, messages are printed to inform about any changes made to the platform. If *True*, thoses messages are turned off.

#### Example

```
>>> platform1.remove_operation(platform1,ignore_msg=True)
```

### netsse.base.Platform.add_seastatecollec

netsse.base.Platform.**add_seastatecollec**(**args*, *ignore_msg: bool = False*)

Add a *SeaStateCollec* instance to the list of sea state collections of the platform.

**Parameters**

- ***args** (*SeaStateCollec object*) – Collections to be added to the platform.
- **ignore_msg** (*bool, default False*) – If *False*, messages are printed to inform about any changes made to the platform. If *True*, thoses messages are turned off.

#### Example

```
>>> seastatecollec2 = netsse.base.SeaStateCollec()
>>> platform1.add_seastatecollec(seastatecollec2,ignore_msg=True)
```

### netsse.base.Platform.remove_seastatecollec

netsse.base.Platform.**remove_seastatecollec**(**args*, *ignore_msg: bool = False*)

Remove an existing *SeaStateCollec* instance from the list of sea state collections of the platform.

**Parameters**

- ***args** (*SeaStateCollec object or str*) – Collections to be removed from the network. Identified by the corresponding identifier or *SeaStateCollec* instance.
- **ignore_msg** (*bool, default False*) – If *False*, messages are printed to inform about any changes made to the platform. If *True*, thoses messages are turned off.

#### Example

```
>>> platform1.remove_seastatecollec(seastatecollec1,ignore_msg=True)
```

**class** netsse.base.**Platform**(*name: str = None*, *owner: str = 'undefined'*, *operations: list = []*, *seastate_collections: list = []*)

Initialises the platform.

**Parameters**

- **name** (*str, optional*) – Name of the platform. By default, the name string is generated randomly in the format `PlatformXXXXX`.
- **owner** (*str, default 'undefined'*) – Platform owner.
- **operations** (*list, default []*) – List of operations.
- **seastate_collections** (*list, default []*) – List of sea state collections (as *SeaStateCollec* instances).

**See also**

`Network`, `Operation`, `SeaStateCollec`, `Vessel`, `Buoy`, `WaveRadar`

### Example

```
>>> operation1 = netsse.base.Operation()
>>> seastatecollec1 = netsse.base.SeaStateCollec()
>>> platform1 = netsse.base.Platform(name='MyPlatform',owner='SomeCompany',        .
↪..                                  operations=[operation1],        ...         ␣
↪                       seastate_collections=[seastatecollec1])
```

### Methods

| | |
|---|---|
| *add_operation*(*args[, ignore_msg]) | Add an *Operation* instance to the list of operations of the platform. |
| *remove_operation*(*args[, ignore_msg]) | Remove an existing *Operation* instance from the list of operations of the |
| *add_seastatecollec*(*args[, ignore_msg]) | Add a *SeaStateCollec* instance to the list of sea state collections |
| *remove_seastatecollec*(*args[, ignore_msg]) | Remove an existing *SeaStateCollec* instance from the list of sea state collections |

## netsse.base.Collection

**class** netsse.base.**Collection**(*area: str = 'undefined'*, *nature: str = 'undefined'*, *origin='undefined'*, *datetime_start=datetime.now()*, *datetime_end=datetime.now()*, *lat_min: float = 0*, *lat_max: float = 0*, *lon_min: float = 0*, *lon_max: float = 0*)

Initialises the data collection.

**Parameters**

- **area** (*str, default 'undefined'*) – Area of data collection.
- **nature** (*str, default 'undefined'*) – Nature of the data collection.
- **origin** (*str or object, default 'undefined'*) – Origin or source of the data collection.
- **datetime_start** (*datetime object, default datetime.now()*) – Time coverage of the data collection.
- **datetime_end** (*datetime object, default datetime.now()*) – Time coverage of the data collection.
- **lat_min** (*float, default 0*) – Latitude [deg North] at the southern/northern edge of the geographical domain covered by the data collection.
- **lat_max** (*float, default 0*) – Latitude [deg North] at the southern/northern edge of the geographical domain covered by the data collection.
- **lon_min** (*float, default 0*) – Longitude [deg East] at the western/eastern edge of the geographical domain covered by the data collection.
- **lon_max** (*float, default 0*) – Longitude [deg East] at the western/eastern edge of the geographical domain covered by the data collection.

**See also**

Operation, SeaStateCollec

### Example

```
>>> collec1 = netsse.base.Collection()
```

## netsse.base.Operation

**class** netsse.base.**Operation**(*id: str = None, area: str = 'undefined', nature: str = 'undefined', origin='undefined', datetime_start=datetime.now(), datetime_end=datetime.now(), lat_min: float = 0, lat_max: float = 0, lon_min: float = 0, lon_max: float = 0, segments: list = []*)

Initialises the operation.

**Parameters**

- **id** (*str, optional*) – Operation identifier. By default, the id string is generated randomly in the format `OperXXXXX`.
- **segments** (*list, default []*) – List of segments (as *Segment* instances) gathered during the operation.

**Note**

Consult `netsse.base.Collection.__init__()` for information on the other parameters.

**See also**

Collection, Platform, Segment

### Example

```
>>> operation1 = netsse.base.Operation()
```

## netsse.base.SeaStateCollec

**class** netsse.base.**SeaStateCollec**(*id: str = None, area: str = 'undefined', nature: str = 'undefined', origin='undefined', datetime_start=datetime.now(), datetime_end=datetime.now(), lat_min: float = 0, lat_max: float = 0, lon_min: float = 0, lon_max: float = 0, seastates: list = []*)

Initialises the sea state collection.

**Parameters**

- **id** (*str, optional*) – Data collection identifier. By default, the *id* string is generated randomly in the format `CollecXXXXX`.
- **seastates** (*list, default []*) – List of sea states (as *SeaState* instances) gathered in the data collection.

> **Note**
>
> Consult `netsse.base.Collection.__init__()` for information on the other parameters.

> **See also**
>
> `Collection`, `Platform`, `SeaState`

### Example

```
>>> seastatecollec1 = netsse.base.SeaStateCollec()
```

## netsse.base.SeaState

### netsse.base.SeaState.set_params

netsse.base.SeaState.**set_params**(*Hs*, *Tp*)

Sets the sea state parameters $H_s$ and $T_p$ to new user-defined values.

**Parameters**

- **Hs** (*float*) – Significant wave height [m].
- **Tp** (*float*) – Peak wave period [s].

### Examples

```
>>> seastate.set_params(Hs,Tp)
```

**class** netsse.base.**SeaState**(*Hs=0*, *Tp=0*, *timestamp=datetime.now()*, *lat=0*, *lon=0*, *depth=0*, *duration=30 * 60*)

Initialises the sea state at a given time and geographical position in the ocean.

**Parameters**

- **Hs** (*float, default 0*) – Significant wave height [m].
- **Tp** (*float, default 0*) – Peak wave period [s].
- **timestamp** (*datetime object, default datetime.now()*) – Timestamp at which the sea state applies.
- **lat** (*float, default 0*) – Latitude [degrees North].
- **lon** (*float, default 0*) – Longitude [degrees East].

- **depth** (*float, default 0*) – Water depth [m] at the studied location.
- **duration** (*float, optional*) – Time duration of the sea state [s], by default 30*60 = 1800 s.

### Examples

```
>>> seastate = SeaState(Hs,Tp,timestamp,lat,lon,depth,duration)
```

### Methods

| | |
|---|---|
| *set_params*(Hs, Tp) | Sets the sea state parameters $H_s$ and $T_p$ to new |

### netsse.base.WaveSequence

### netsse.base.WaveSequence.set_params

netsse.base.WaveSequence.**set_params**(*Hs*, *Tp*)

Sets the sea state parameters $H_s$ and $T_p$ to new user-defined values.

**Parameters**

- **Hs** (*float*) – Significant wave height [m].
- **Tp** (*float*) – Peak wave period [s].

### Examples

```
>>> seastate.set_params(Hs,Tp)
```

**class** netsse.base.**WaveSequence**(*Hs=0*, *Tp=0*, *timestamp=datetime.now()*, *lat=0*, *lon=0*, *depth=0*, *wave=None*, *time=None*, *fs=10*)

Bases: SeaState

Initialises the wave sequence.

**Parameters**

- **wave** (*array-like of shape (Nt,), optional*) – Sea surface elevation time-series [m]. By default None.
- **time** (*array-like of shape (Nt,), optional*) – Vector of time [s], by default None.
- **fs** (*float, default 10*) – Sampling frequency [Hz].

> **Note**
>
> Consult netsse.base.SeaState.__init__() for information on the other parameters.

**Raises**

`ValueError` – In case the input `time` and `wave` vectors do not have the same length.

### Examples

```
>>> waveseq =
...     WaveSequence(Hs, Tp, timestamp, lat, lon, depth, wave, time)
```

### Methods

| | |
|---|---|
| *set_params*(Hs, Tp) | Sets the sea state parameters $H_s$ and $T_p$ to new |

### netsse.base.WaveSpectrum

### netsse.base.WaveSpectrum.set_params

netsse.base.WaveSpectrum.**set_params**(*Hs*, *Tp*)

Sets the sea state parameters $H_s$ and $T_p$ to new user-defined values.

**Parameters**

- **Hs** (*float*) – Significant wave height [m].
- **Tp** (*float*) – Peak wave period [s].

### Examples

```
>>> seastate.set_params(Hs,Tp)
```

**class** netsse.base.**WaveSpectrum**(*Hs=np.nan*, *Tp=np.nan*, *freq=np.linspace(0, 1, 100)*, *ord=np.zeros((100,))*, *unit_freq='Hz'*, *timestamp=datetime.now()*, *lat=0*, *lon=0*, *depth=0*, *duration=30 * 60*)

Bases: `Spectrum`, `SeaState`

Initialises the sea state at a given time and geographical position in the ocean.

**Parameters**

- **Hs** (*float, default 0*) – Significant wave height [m].
- **Tp** (*float, default 0*) – Peak wave period [s].
- **timestamp** (*datetime object, default datetime.now()*) – Timestamp at which the sea state applies.
- **lat** (*float, default 0*) – Latitude [degrees North].
- **lon** (*float, default 0*) – Longitude [degrees East].
- **depth** (*float, default 0*) – Water depth [m] at the studied location.
- **duration** (*float, optional*) – Time duration of the sea state [s], by default 30*60 = 1800 s.

#### Examples

```
>>> seastate = SeaState(Hs,Tp,timestamp,lat,lon,depth,duration)
```

### Methods

| | |
|---|---|
| *set_params*(Hs, Tp) | Sets the sea state parameters $H_s$ and $T_p$ to new |

### netsse.base.Buoy

### netsse.base.Buoy.add_operation

netsse.base.Buoy.**add_operation**(**args*, *ignore_msg: bool = False*)

Add an *Operation* instance to the list of operations of the platform.

**Parameters**

- ***args** (*Operation object*) – Operation to be added to the platform.
- **ignore_msg** (*bool, default False*) – If *False*, messages are printed to inform about any changes made to the platform. If *True*, thoses messages are turned off.

#### Example

```
>>> operation2 = netsse.base.Operation()
>>> platform1.add_operation(operation2,ignore_msg=True)
```

### netsse.base.Buoy.remove_operation

netsse.base.Buoy.**remove_operation**(**args*, *ignore_msg: bool = False*)

Remove an existing *Operation* instance from the list of operations of the platform.

**Parameters**

- ***args** (*Operation object or str*) – Operation to be removed from the network. Identified by the corresponding identifier or *Operation* instance.
- **ignore_msg** (*bool, default False*) – If *False*, messages are printed to inform about any changes made to the platform. If *True*, thoses messages are turned off.

#### Example

```
>>> platform1.remove_operation(platform1,ignore_msg=True)
```

### netsse.base.Buoy.add_seastatecollec

netsse.base.Buoy.**add_seastatecollec**(**args*, *ignore_msg: bool = False*)

Add a *SeaStateCollec* instance to the list of sea state collections of the platform.

**Parameters**

- ***args** (*SeaStateCollec object*) – Collections to be added to the platform.
- **ignore_msg** (*bool, default False*) – If *False*, messages are printed to inform about any changes made to the platform. If *True*, thoses messages are turned off.

#### Example

```
>>> seastatecollec2 = netsse.base.SeaStateCollec()
>>> platform1.add_seastatecollec(seastatecollec2,ignore_msg=True)
```

### netsse.base.Buoy.remove_seastatecollec

netsse.base.Buoy.**remove_seastatecollec**(**args*, *ignore_msg: bool = False*)

Remove an existing *SeaStateCollec* instance from the list of sea state collections of the platform.

**Parameters**

- ***args** (*SeaStateCollec object or str*) – Collections to be removed from the network. Identified by the corresponding identifier or *SeaStateCollec* instance.
- **ignore_msg** (*bool, default False*) – If *False*, messages are printed to inform about any changes made to the platform. If *True*, thoses messages are turned off.

#### Example

```
>>> platform1.remove_seastatecollec(seastatecollec1,ignore_msg=True)
```

**class** netsse.base.**Buoy**(*name: str = None*, *owner: str = 'undefined'*, *operations: list = []*, *seastate_collections: list = []*, *type: str = 'undefined'*)

Bases: Platform

Initialises the buoy as a platform.

**Parameters**

**type** (*str, default 'undefined'*) – Buoy type.

> **Note**
>
> Consult netsse.base.Platform.__init__() for information on the other parameters.

#### Example

```
>>> buoy1 = netsse.base.Buoy()
```

### Methods

| | |
|---|---|
| *add_operation*(*args[, ignore_msg]) | Add an *Operation* instance to the list of operations of the platform. |
| *remove_operation*(*args[, ignore_msg]) | Remove an existing *Operation* instance from the list of operations of the |
| *add_seastatecollec*(*args[, ignore_msg]) | Add a *SeaStateCollec* instance to the list of sea state collections |
| *remove_seastatecollec*(*args[, ignore_msg]) | Remove an existing *SeaStateCollec* instance from the list of sea state collections |

## netsse.base.WaveRadar

## netsse.base.WaveRadar.add_operation

netsse.base.WaveRadar.**add_operation**(**args*, *ignore_msg: bool = False*)

Add an *Operation* instance to the list of operations of the platform.

**Parameters**

- ***args** (*Operation object*) – Operation to be added to the platform.
- **ignore_msg** (*bool, default False*) – If *False*, messages are printed to inform about any changes made to the platform. If *True*, thoses messages are turned off.

#### Example

```
>>> operation2 = netsse.base.Operation()
>>> platform1.add_operation(operation2,ignore_msg=True)
```

## netsse.base.WaveRadar.remove_operation

netsse.base.WaveRadar.**remove_operation**(**args*, *ignore_msg: bool = False*)

Remove an existing *Operation* instance from the list of operations of the platform.

**Parameters**

- ***args** (*Operation object or str*) – Operation to be removed from the network. Identified by the corresponding identifier or *Operation* instance.
- **ignore_msg** (*bool, default False*) – If *False*, messages are printed to inform about any changes made to the platform. If *True*, thoses messages are turned off.

#### Example

```
>>> platform1.remove_operation(platform1,ignore_msg=True)
```

### netsse.base.WaveRadar.add_seastatecollec

netsse.base.WaveRadar.**add_seastatecollec**(**args*, *ignore_msg: bool = False*)

Add a *SeaStateCollec* instance to the list of sea state collections of the platform.

**Parameters**

- ***args** (*SeaStateCollec object*) – Collections to be added to the platform.
- **ignore_msg** (*bool, default False*) – If *False*, messages are printed to inform about any changes made to the platform. If *True*, thoses messages are turned off.

#### Example

```
>>> seastatecollec2 = netsse.base.SeaStateCollec()
>>> platform1.add_seastatecollec(seastatecollec2,ignore_msg=True)
```

### netsse.base.WaveRadar.remove_seastatecollec

netsse.base.WaveRadar.**remove_seastatecollec**(**args*, *ignore_msg: bool = False*)

Remove an existing *SeaStateCollec* instance from the list of sea state collections of the platform.

**Parameters**

- ***args** (*SeaStateCollec object or str*) – Collections to be removed from the network. Identified by the corresponding identifier or *SeaStateCollec* instance.
- **ignore_msg** (*bool, default False*) – If *False*, messages are printed to inform about any changes made to the platform. If *True*, thoses messages are turned off.

#### Example

```
>>> platform1.remove_seastatecollec(seastatecollec1,ignore_msg=True)
```

**class** netsse.base.**WaveRadar**(*name: str = None*, *owner: str = 'undefined'*, *operations: list = []*, *seastate_collections: list = []*, *type: str = 'undefined'*)

Bases: Platform

Initialises the wave radar as a platform.

**Parameters**

**type** (*str, default 'undefined'*) – Wave radar type.

> **Note**
>
> Consult netsse.base.Platform.__init__() for information on the other parameters.

#### Example

```
>>> radar1 = netsse.base.WaveRadar()
```

### Methods

| | |
|---|---|
| *add_operation*(*args[, ignore_msg]) | Add an *Operation* instance to the list of operations of the platform. |
| *remove_operation*(*args[, ignore_msg]) | Remove an existing *Operation* instance from the list of operations of the |
| *add_seastatecollec*(*args[, ignore_msg]) | Add a *SeaStateCollec* instance to the list of sea state collections |
| *remove_seastatecollec*(*args[, ignore_msg]) | Remove an existing *SeaStateCollec* instance from the list of sea state collections |

## netsse.base.Vessel

## netsse.base.Vessel.add_operation

netsse.base.Vessel.**add_operation**(**args*, *ignore_msg: bool = False*)

Add an *Operation* instance to the list of operations of the platform.

**Parameters**

- ***args** (*Operation object*) – Operation to be added to the platform.
- **ignore_msg** (*bool, default False*) – If *False*, messages are printed to inform about any changes made to the platform. If *True*, thoses messages are turned off.

#### Example

```
>>> operation2 = netsse.base.Operation()
>>> platform1.add_operation(operation2,ignore_msg=True)
```

## netsse.base.Vessel.remove_operation

netsse.base.Vessel.**remove_operation**(**args*, *ignore_msg: bool = False*)

Remove an existing *Operation* instance from the list of operations of the platform.

**Parameters**

- ***args** (*Operation object or str*) – Operation to be removed from the network. Identified by the corresponding identifier or *Operation* instance.
- **ignore_msg** (*bool, default False*) – If *False*, messages are printed to inform about any changes made to the platform. If *True*, thoses messages are turned off.

#### Example

```
>>> platform1.remove_operation(platform1,ignore_msg=True)
```

### netsse.base.Vessel.add_seastatecollec

netsse.base.Vessel.**add_seastatecollec**(**args*, *ignore_msg: bool = False*)

Add a *SeaStateCollec* instance to the list of sea state collections of the platform.

**Parameters**

- ***args** (*SeaStateCollec object*) – Collections to be added to the platform.
- **ignore_msg** (*bool, default False*) – If *False*, messages are printed to inform about any changes made to the platform. If *True*, thoses messages are turned off.

#### Example

```
>>> seastatecollec2 = netsse.base.SeaStateCollec()
>>> platform1.add_seastatecollec(seastatecollec2,ignore_msg=True)
```

### netsse.base.Vessel.remove_seastatecollec

netsse.base.Vessel.**remove_seastatecollec**(**args*, *ignore_msg: bool = False*)

Remove an existing *SeaStateCollec* instance from the list of sea state collections of the platform.

**Parameters**

- ***args** (*SeaStateCollec object or str*) – Collections to be removed from the network. Identified by the corresponding identifier or *SeaStateCollec* instance.
- **ignore_msg** (*bool, default False*) – If *False*, messages are printed to inform about any changes made to the platform. If *True*, thoses messages are turned off.

#### Example

```
>>> platform1.remove_seastatecollec(seastatecollec1,ignore_msg=True)
```

**class** netsse.base.**Vessel**(*name: str = None*, *owner: str = 'undefined'*, *operations: list = []*, *seastate_collections: list = []*, *type: str = 'undefined'*, *params: dict = {}*, *raos: list = []*)

Bases: Platform

Initialises the wave radar as a platform.

**Parameters**

- **type** (*str, default 'undefined'*) – Wave radar type.
- **params** (*dict, default {}*) – Vessel main dimensions and other geometrical parameters.
- **raos** (*list, default []*) – Vessel response amplitude operators.

> **Note**
>
> Consult `netsse.base.Platform.__init__()` for information on the other parameters.

### Example

```
>>> vessel1 = netsse.base.Vessel(params={'L':200,'B':25,'T':30})
```

### Methods

| | |
|---|---|
| *add_operation*(*args[, ignore_msg]) | Add an *Operation* instance to the list of operations of the platform. |
| *remove_operation*(*args[, ignore_msg]) | Remove an existing *Operation* instance from the list of operations of the |
| *add_seastatecollec*(*args[, ignore_msg]) | Add a *SeaStateCollec* instance to the list of sea state collections |
| *remove_seastatecollec*(*args[, ignore_msg]) | Remove an existing *SeaStateCollec* instance from the list of sea state collections |

| | |
|---|---|
| *Network* | Initialises the network. |
| *Platform* | Initialises the platform. |
| *Collection* | Initialises the data collection. |
| *Operation* | Initialises the operation. |
| *SeaStateCollec* | Initialises the sea state collection. |
| *SeaState* | Initialises the sea state at a given time and geographical position in the ocean. |
| *WaveSequence* | Initialises the wave sequence. |
| *WaveSpectrum* | Initialises the sea state at a given time and geographical position in the ocean. |
| *Buoy* | Initialises the buoy as a platform. |
| *WaveRadar* | Initialises the wave radar as a platform. |
| *Vessel* | Initialises the wave radar as a platform. |

CHAPTER

FOUR

# RELEASE NOTES

This is the list of changes to NetSSE between each release. For full details, see the commit logs on GitLab. For install and upgrade instructions, see *Installing NetSSE*.

| Version | Release date | New features |
|---|---|---|
| 1.0 | 27-07-2023 | *Initial modules*: analys.buoy, analys.spec, model.ship, simul.ship_resp, tools.envir_cond, tools.misc_func. |
| 2.0 (latest) | 26-07-2024 | *New modules*: base, analys.emep, analys.enc_spec, analys.sawb. |

CHAPTER

FIVE

# ABOUT THE PROJECT

**NetSSE** is the result of ongoing work carried out from 2020 onwards by researchers at the Technical University of Denmark (DTU), Department of Civil and Mechanical Engineering, Section of Fluid Mechanics, Coastal and Maritime Engineering.

## 5.1 Motivation

Ocean waves represent the main source of dynamic excitation of surface vessels and floating marine structures. The scarcity of wave data from vast areas of the world's oceans is yet an ongoing problem. The crucial quantification and mitigation of uncertainties inherent to wave monitoring is increasingly recognised by the shipping, offshore, and renewable energy industries, as well as in coastal engineering[1].

Fusing data from multiple and heterogeneous observation platforms enables reducing the uncertainties of measurements from individual platforms. In this framework, the **ship-as-a-wave-buoy** concept shows high relevance, simply by accounting for the sheer number of ships in transit on the open sea. The wave-induced responses of modern vessels are often monitored by inexpensive off-the-shelf sensors, therefore constituting data that can be used for estimating sea states in near real-time, in a fundamentally similar way to traditional wave-rider buoys. **Vessel response-based estimates of the sea state**, combined with measurements from other met-ocean sensors, can be integrated into observation networks to improve the reliability and availability of wave data[1].

With this in mind, **NetSSE** – as the contraction of "Network" and "Sea State Estimation" – can be seen as a toobox to help scientists and engineers experiment with networks of wave sensor systems, especially integrating **vessels as "sailing wave buoys"**. NetSSE can be used for instance to generate and process synthetic data of vessel responses, to extract sample results from model-scale experiments, and to analyse and merge together real-world data collected by seagoing vessels and other observation platforms.

The project's overall aim is to improve the sampling and modelling of the ocean wave environment around the globe, thereby contributing to scientific understanding, as well as to the sustainability and operational safety of ocean-based human activities. This shall positively contribute to the Sustainable Development Goals (SDGs) postulated by the United Nations, having the main effects aligned with SDG 7 ("Affordable and clean energy"), SDG 13 ("Climate action"), SDG 14 ("Life below water"), and SDG 17 ("Partnerships for the goals").

[1] Mounet, R. E. G. *Sea state estimation based on measurements from multiple observation platforms*. Ph.D. dissertation, Technical University of Denmark, 2023.

## 5.2 Funding acknowledgment

### 5.2.1 2024-now

The main developer's postdoctoral work is financially supported by the Independent Research Fund of Denmark (grant ID: 10.46540/3164-00087B) and Orients Fond (case: SEAFUSION under the framework agreement 'Orients Fond 2021-2025').

### 5.2.2 2020-2023

The main developer's doctoral work was partly supported by the Research Council of Norway through the Centres of Excellence funding scheme [project number 223254, NTNU AMOS, Center for Autonomous Marine Operations and Systems]. Moreover, the financial support from the Danish Maritime Fund received in 2017 and 2020 is greatly acknowledged.

## 5.3 Contents

### 5.3.1 The NetSSE team

Raphaël E. G. Mounet, *DTU Construct*

Ulrik D. Nielsen, *DTU Construct*

### 5.3.2 Citing and logo

If you find NetSSE useful and use it in your research or project, we kindly ask you to credit the developers using below reference:

```
Mounet, R. E. G., & Nielsen, U. D. NetSSE: An open-source Python package for network-
↪based sea state estimation from ships, buoys, and other observation platforms (version␣
↪2.0). Technical University of Denmark, GitLab. July 2024. https://doi.org/10.11583/DTU.
↪26379811.
```

#### BibTeX reference

```
@software{NetSSE2024,
  author       = {Raphaël E. G. Mounet and Ulrik D. Nielsen},
  title        = {{NetSSE: An open-source Python package for network-based sea state
                  estimation from ships, buoys, and other observation platforms}},
  month        = {8},
  year         = {2024},
  publisher    = {Technical University of Denmark, GitLab},
  version      = {version 2.0},
  url          = {https://gitlab.gbar.dtu.dk/regmo/NetSSE},
  doi          = {https://doi.org/10.11583/DTU.26379811},
}
```

### Logo

The current logo of NetSSE was designed by Jennifer Mounet in 2024.

#### Colour

This subsection is under development.

### 5.3.3 License

**NetSSE** is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

NetSSE is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see the license terms.

### 5.3.4 Publications

The following research publications have used the NetSSE package.

#### Journal papers

1. Mounet, R.E.G., Chen, J., Nielsen, U.D., Brodtkorb, A.H., Pillai, A.C., Ashton, I.G.C., Steele, E.C.C. (2023). **Deriving Spatial Wave Data from a Network of Buoys and Ships**. Ocean Engineering, 281, 114892. https://doi.org/10.1016/j.oceaneng.2023.114892.
2. Mounet, R.E.G., Nielsen, U.D., Brodtkorb, A.H., Øveraas, H., Dallolio, A., Johansen, T.A. (2024). **Data-Driven Method for Hydrodynamic Model Estimation Applied to an Unmanned Surface Vehicle**. Measurement, 234, 114724. https://doi.org/10.1016/j.measurement.2024.114724.

#### Conference papers

3. Mounet, R.E.G., Nielsen, U.D., Brodtkorb, A.H. (2023). **Doppler Shift Approximation for Predicting the Wave-Induced Response of Advancing Vessels in Following Waves**. Proceedings of the ASME 2023 42nd International Conference on Ocean, Offshore and Arctic Engineering (OMAE'23). https://doi.org/10.1115/OMAE2023-107733.
4. Mounet, R.E.G., Nielsen, U.D. (2024). **An Application of the Extended Maximum Entropy Principle for the Spectral Analysis of Ship Motions**. Proceedings of the 2024 IEEE International Workshop on Metrology for the Sea. (*Under review*).

#### Other publications

5. Mounet, R.E.G. (2023). **Sea State Estimation Based on Measurements from Multiple Observation Platforms**. PhD Thesis. Technical University of Denmark. https://orbit.dtu.dk/en/publications/sea-state-estimation-based-on-measurements-from-multiple-observat

### 5.3.5 Contact

For any enquiry about the software, please contact the main developer Raphaël E. G. Mounet.

#### Mailing list

NetSSE has a newsletter through which occasional updates are sent when new functionalities are added to the package. The mailing list receives up to 1-3 emails yearly.

If you wish to subscribe to the newsletter, please fill in this sign-up form:

Sign-up

#### References

CHAPTER

SIX

# WHAT IS NETSSE?

**NetSSE** is an open-source Python package for network-based sea state estimation from ships, buoys, and other observation platforms. The objective of NetSSE is to provide a scientific framework to develop, test, and distribute new sea state estimation methods. All code is available at the NetSSE GitLab.

> **Note**
>
> This project is under active development and will be running until December 2027.

User Guide User guide on installation and the basic concepts of NetSSE.

*NetSSE user guide* Examples Examples of NetSSE usage.

*Examples* API Reference NetSSE application programming interface (API) reference.

*API Reference* Publications Find an overview of scientific peer-reviewed studies that used NetSSE.

*Publications* Contact us Want to contribute to the project? Get to know the developer team and reach out to us.

*Contact*

CHAPTER

SEVEN

# QUICK EXAMPLE

**Todo**

This section is still under development.

CHAPTER

# EIGHT

# USING NETSSE? PLEASE CITE US!

If you find NetSSE useful and use it in your research or project, we kindly ask you to credit the developers using below reference:

```
Mounet, R. E. G., & Nielsen, U. D. NetSSE: An open-source Python package for network-
↪based sea state estimation from ships, buoys, and other observation platforms (version␣
↪2.0). Technical University of Denmark, GitLab. July 2024. https://doi.org/10.11583/DTU.
↪26379811.
```

CHAPTER

NINE

# DOCUMENTATION IN .PDF FORMAT

A .PDF version of the latest NetSSE documentation is available for download here. As a disclaimer, the .PDF document is automatically generated and may therefore contain errors or inconsistencies. To avoid any confusion, the latest online documentation should always be consulted, if possible.

# PYTHON MODULE INDEX

## n

# INDEX

## I



## J



## M



## N

## P



## R



## S



## U



## W