Spectral representation and discretisation

Finn Lindgren

September 2021, with major updates May/June 2022

The theory discussed in this note follows from the materials found in, among many other places, Cramér and Leadbetter (2004), Lindgren (2012) (available online via the university library), and Lindgren, Rootzen, and Sandsten (2013). The materials in the two latter books are later revisions of older teaching materials, and the first is a reprint of the 1967 original.

Key observation: When the correctly folded spectral density for the spatially discretised process is used, the approximation in the direct spectral representation comes from the frequency *resolution*, not from frequency *truncation*.

Other approximate spectral techniques include the "turning bands" method, that relies on Monte Carlo integration of the Fourier integrals instead of lattice FFT computations.

Continuous domains and finite dimensional representations via Fourier theory

The exact spectral representation of the covariance evaluated on a discrete (infinite) lattice is easily derived from the continuous domain representation

$$R(\boldsymbol{s}) = \int_{\mathbb{R}^d} \exp(i\boldsymbol{\omega}\cdot\boldsymbol{s}) S(\boldsymbol{\omega})\,\mathrm{d}\boldsymbol{\omega}.$$

For simplicity, assume the same lattice spacing h in each direction. The stationary covariance function $R(\boldsymbol{s})$ evaluated at lattice points $\boldsymbol{k}h$, $k \in \mathbb{Z}^d$ is given by

$$\begin{aligned}
R(\boldsymbol{j}h) &= \int_{\mathbb{R}^d} \exp(i\boldsymbol{\omega}\cdot\boldsymbol{j}h) S(\boldsymbol{\omega})\,\mathrm{d}\boldsymbol{\omega} \\
&= \int_{[-\pi/h,\pi/h)^d} \sum_{\boldsymbol{k}\in\mathbb{Z}^d} \exp(i(\boldsymbol{\omega}+2\pi\boldsymbol{k}/h)h\boldsymbol{j}) S(\boldsymbol{\omega}+2\pi\boldsymbol{k}/h)\,\mathrm{d}\boldsymbol{\omega} \\
&= \int_{[-\pi/h,\pi/h)^d} \exp(i\boldsymbol{\omega}\cdot\boldsymbol{j}h) \widetilde{S}(\boldsymbol{\omega})\,\mathrm{d}\boldsymbol{\omega}, \quad \boldsymbol{j}\in\mathbb{Z}^d
\end{aligned}$$

where

$$\widetilde{S}(\boldsymbol{\omega}) = \sum_{\boldsymbol{k}\in\mathbb{Z}^d} S(\boldsymbol{\omega}+2\pi\boldsymbol{k}/h), \quad \boldsymbol{\omega}\in[-\pi/h,\pi/h)^d$$

If instead the spatial discretisation should be interpreted as the *cell averages* (which is the more usual case for PDE discretisations and e.g. satellite data, rather than pointwise values), the spectrum is altered by a multiplicative frequency filter with a squared sinc function:

$$\widetilde{S}(\boldsymbol{\omega}) = \sum_{\boldsymbol{k}\in\mathbb{Z}^d} S(\boldsymbol{\omega}+2\pi\boldsymbol{k}/h) \prod_{l=1}^{d} \left\{ \frac{\sin[(\omega_l+2\pi k_l/h)h/2]}{(\omega_l+2\pi k_l/h)h/2} \right\}^2, \quad \boldsymbol{\omega}\in[-\pi/h,\pi/h)^d.$$

Sampling, in theory

According to the spectral representation theory for processes with continuous spectra, we have

$$x(\boldsymbol{s}) = \int_{\mathbb{R}^d} \exp(i\boldsymbol{\omega}\cdot\boldsymbol{s}) S(\boldsymbol{\omega})^{1/2}\,\mathrm{d}Z(\boldsymbol{\omega}), \quad \boldsymbol{s}\in\mathbb{R}^d,$$

where where $\mathrm{d}Z(-\boldsymbol{\omega}) = \overline{\mathrm{d}Z(\boldsymbol{\omega})}$, $\mathsf{Cov}(\mathrm{d}Z(\boldsymbol{\omega}), \mathrm{d}Z(\boldsymbol{\omega}')) = \delta(\boldsymbol{\omega} - \boldsymbol{\omega}')\,\mathrm{d}\boldsymbol{\omega}$ for $\boldsymbol{\omega} \in \mathbb{R}^d$.

With the above theory, for sampling, the integral again reduces to a finite domain, but one needs to construct a spectral white noise process $\mathrm{d}Z(\boldsymbol{\omega})$ with complex conjugate symmetry:

$$x(\boldsymbol{j}h) = \int_{[-\pi/h,\pi/h)^d} \exp(i\boldsymbol{\omega}\cdot\boldsymbol{j}h)\widetilde{S}(\boldsymbol{\omega})^{1/2}\,\mathrm{d}Z(\boldsymbol{\omega}), \quad \boldsymbol{j} \in \mathbb{Z}^d,$$

where $\mathrm{d}Z(-\boldsymbol{\omega}) = \overline{\mathrm{d}Z(\boldsymbol{\omega})}$, $\mathsf{Cov}(\mathrm{d}Z(\boldsymbol{\omega}), \mathrm{d}Z(\boldsymbol{\omega}')) = \delta(\boldsymbol{\omega} - \boldsymbol{\omega}')\,\mathrm{d}\boldsymbol{\omega}$ for $\boldsymbol{\omega} \in [-\pi/h, \pi/h)^d$. This can in principle be discretised with a lattice of frequencies in much the same way as a basic integral approximation method that can be used to compute the deterministic integral for the covariance function, with noise variances equal to the cell area/volume of each frequency lattice point. However, the complex conjugacy can be tricky to work with in practice. In the next section we will solve this issue, and provide a fast computational method.

Practical numerical evaluation with FFT

Above we let the spacings h be the same in all dimensions. For greater generality, we now define discrete spacings $\boldsymbol{h} = \{h_1, \ldots, h_d\}$. Let the number of evaluation points in each dimension be given by $\boldsymbol{n} = \{n_1, \ldots, n_d\}$. For the cell interpretation of the spatial discretisation, the domain with for the different directions are given by $\boldsymbol{L} = \boldsymbol{n} \odot \boldsymbol{h}$. Define the continuous spectral domain $\mathcal{I} = \prod_{l=1}^d [-\pi/h_l, \pi/h_l)$, a semi-symmetric index space as $\mathcal{K}_0 = \prod_{l=1}^d \{-n_l/2, \ldots, -1, 0, 1, 2, \ldots, n_l/2 - 1\}$, and and a non-negative index space $\mathcal{K}_+ = \prod_{l=1}^d \{0, 1, 2, \ldots, n_l - 1\}$, so that if $\boldsymbol{k} \in \mathcal{K}_0$, then $\boldsymbol{k} \bmod \boldsymbol{n} \in \mathcal{K}_+$.

We want to evaluate

$$x(\boldsymbol{j} \odot \boldsymbol{h}) = \int_{\mathcal{I}} \exp(i\boldsymbol{\omega}\cdot\boldsymbol{j} \odot \boldsymbol{h})\widehat{x}(\boldsymbol{\omega})\,\mathrm{d}\boldsymbol{\omega}$$

for all $\boldsymbol{j} \in \boldsymbol{K}_+$ or all $\boldsymbol{j} \in \boldsymbol{K}_0$.

The (inverse) FFT in `fftwtools` is a by default unnormalised sum,

$$x_{\boldsymbol{j}} = \sum_{\boldsymbol{k}\in\mathcal{K}_+} \exp(i2\pi\boldsymbol{k}\cdot\boldsymbol{j} \oslash \boldsymbol{n})\widehat{x}_{\boldsymbol{k}}, \quad \boldsymbol{j} \in \mathcal{K}_+.$$

so we need to make a Riemann sum approximation of the integral and write it on this form.

Define a discretisation of the $\boldsymbol{\omega}$ space via

$$(\boldsymbol{\omega}_{\boldsymbol{k}})_l = \frac{\pi k_l}{h_l n_l/2} = \frac{2\pi k_l}{h_l n_l}, \quad k_l = -n_l/2, \ldots, n_l/2 - 1$$

Then

$$x(\boldsymbol{j} \odot \boldsymbol{h}) \approx \sum_{\{k_l=-n_l/2\}}^{\{n_l/2-1\}} \exp(i2\pi\boldsymbol{k}\cdot\boldsymbol{j} \oslash \boldsymbol{n})\widehat{x}(2\pi\boldsymbol{k} \oslash (\boldsymbol{n} \odot \boldsymbol{h})) \prod_{l=1}^d \frac{2\pi}{h_l n_l}$$

For the negative $\boldsymbol{k}$ indices, we can add a shift by $\boldsymbol{n}$ without altering the result (if we also adjust $\widehat{x}$ accordingly), since $\exp(i2\pi\boldsymbol{n}\cdot\boldsymbol{j} \oslash \boldsymbol{n}) = \exp(i2\pi\boldsymbol{j}) = 1$ for all $\boldsymbol{j}$. We can therefore compute an approximation of the integral by letting

$$\widehat{x}_{\boldsymbol{k} \bmod \boldsymbol{n}} = \widehat{x}(2\pi\boldsymbol{k} \oslash (\boldsymbol{n} \odot \boldsymbol{h})) \prod_{l=1}^d \frac{2\pi}{h_l n_l}, \quad \boldsymbol{k} \in \mathcal{K}_0.$$

Covariance evaluation

To compute an approximation of the covariance function, we define

$$\sigma^2_{\boldsymbol{k}} = \widetilde{S}(2\pi\boldsymbol{k} \oslash (\boldsymbol{n} \odot \boldsymbol{h})) \prod_{l=1}^d \frac{2\pi}{h_l n_l}, \quad \boldsymbol{k} \in \mathcal{K}_0,$$

and let

$$\widehat{x}_{\boldsymbol{k} \bmod \boldsymbol{n}} = \sigma^2_{\boldsymbol{k}}, \quad \boldsymbol{k} \in \mathcal{K}_0.$$

Then $R(\boldsymbol{j} \odot \boldsymbol{h}) \approx x_{\boldsymbol{j} \bmod \boldsymbol{n}}$ for $\boldsymbol{j} \in \mathcal{K}_0$.

Process/field simulation

For simulation, the relation $\mathrm{d}Z(-\boldsymbol{\omega}) = \overline{\mathrm{d}Z(\boldsymbol{\omega})}$ makes direct construction of the discretised integrand difficult, as we need to keep track of the complex conjugate symmetry. A practical alternative can be found in Appendix B.4 of (Lindgren 2012), where it is shown that if the complex conjugate symmetry is ignored, the result of the (inverse) Fourier transform gives a sample from the process as its real part, and a corresponding sample from the process *envelope* as the norms of the absolute values.

We here present a similar method that is easier to show gives the correct result. Let $(\boldsymbol{\omega}_{\boldsymbol{k}})_l = \frac{2\pi k_l}{h_l n_l}$ for $k_l = -n_l/2, 1, \ldots, n_l/2 - 1$. Define

$$\sigma^2_{\boldsymbol{k}} = \widetilde{S}(\boldsymbol{\omega}_{\boldsymbol{k}}) \prod_{l=1}^{d} \frac{2\pi}{h_l n_l}, \quad \boldsymbol{k} \in \mathcal{K}_0,$$

Let $\widehat{z}_{\boldsymbol{k}}$ be iid complex valued $\mathsf{N}(0,1)$ random variables, and let

$$\widehat{x}_{\boldsymbol{k}} = \widehat{z}_{\boldsymbol{k}} \sigma_{\boldsymbol{k}}, \quad \boldsymbol{k} \in \mathcal{K}_0.$$

Then the real part of the Fourier transformed values, $x(\boldsymbol{j} \odot \boldsymbol{h}) = \Re[x_{\boldsymbol{j}}]$ form an approximate sample from the process model, $\boldsymbol{j} \in \mathcal{K}_+$.

[and $|x_{\boldsymbol{j}}|$ is the corresponding sample of the process envelope, for $\boldsymbol{j} \in \mathcal{K}_+$.]

Proof Let $a_{\boldsymbol{k}}$ and $a_{\boldsymbol{k}}$ be the real and imaginary parts of $\widehat{z}_{\boldsymbol{k}}$. Define $c_{\boldsymbol{k}\cdot\boldsymbol{j}} = \cos(\boldsymbol{\omega}_{\boldsymbol{k}} \cdot \boldsymbol{j} \odot \boldsymbol{h})$ and $s_{\boldsymbol{k}\cdot\boldsymbol{j}} = \sin(\boldsymbol{\omega}_{\boldsymbol{k}} \cdot \boldsymbol{j} \odot \boldsymbol{h})$. Then

$$\begin{aligned}
x_{\boldsymbol{j}} &= \sum_{\boldsymbol{k}\in\mathcal{K}_0} \exp(i\boldsymbol{\omega}_{\boldsymbol{k}} \cdot \boldsymbol{j} \odot \boldsymbol{h})(a_{\boldsymbol{k}} + ib_{\boldsymbol{k}})\sigma_{\boldsymbol{k}} \\
&= \sum_{\boldsymbol{k}\in\mathcal{K}_0} (c_{\boldsymbol{k}\cdot\boldsymbol{j}} + is_{\boldsymbol{k}\cdot\boldsymbol{j}})(a_{\boldsymbol{k}} + ib_{\boldsymbol{k}})\sigma_{\boldsymbol{k}} \\
&= \sum_{\boldsymbol{k}\in\mathcal{K}_0} (c_{\boldsymbol{k}\cdot\boldsymbol{j}} a_{\boldsymbol{k}} - s_{\boldsymbol{k}\cdot\boldsymbol{j}} b_{\boldsymbol{k}})\sigma_{\boldsymbol{k}} + i \sum_{\boldsymbol{k}\in\mathcal{K}_0} (c_{\boldsymbol{k}\cdot\boldsymbol{j}} b_{\boldsymbol{k}} + s_{\boldsymbol{k}\cdot\boldsymbol{j}} a_{\boldsymbol{k}})\sigma_{\boldsymbol{k}}.
\end{aligned}$$

The covariance for the real part is given by

$$\begin{aligned}
\mathsf{Cov}[\Re(x_{\boldsymbol{j}}), \Re(x_{\boldsymbol{j}'})] &= \sum_{\boldsymbol{k}\in\mathcal{K}_0} \mathsf{Cov}[(c_{\boldsymbol{k}\cdot\boldsymbol{j}} a_{\boldsymbol{k}} - s_{\boldsymbol{k}\cdot\boldsymbol{j}} b_{\boldsymbol{k}})\sigma_{\boldsymbol{k}}, (c_{\boldsymbol{k}\cdot\boldsymbol{j}'} a_{\boldsymbol{k}} - s_{\boldsymbol{k}\cdot\boldsymbol{j}'} b_{\boldsymbol{k}})\sigma_{\boldsymbol{k}}] \\
&= \sum_{\boldsymbol{k}\in\mathcal{K}_0} (c_{\boldsymbol{k}\cdot\boldsymbol{j}} c_{\boldsymbol{k}\cdot\boldsymbol{j}'} + s_{\boldsymbol{k}\cdot\boldsymbol{j}} s_{\boldsymbol{k}\cdot\boldsymbol{j}'})\sigma^2_{\boldsymbol{k}} \\
&= \sum_{\boldsymbol{k}\in\mathcal{K}_0} c_{\boldsymbol{k}\cdot(\boldsymbol{j}'-\boldsymbol{j})}\sigma^2_{\boldsymbol{k}}.
\end{aligned}$$

where the last step follows from the trigonometric identity for the cosine of the difference between two angles. By considering the definition of $\sigma^2_{\boldsymbol{k}}$, we see that the resulting sum is a Riemann sum approximation of the cosine integral

$$\begin{aligned}
\int_{\prod_{l=0}^{d}[-\pi/h_l,\pi/h_l)} \cos(\boldsymbol{\omega} \cdot (\boldsymbol{j}' - \boldsymbol{j}) \odot \boldsymbol{h})\widetilde{S}(\boldsymbol{\omega})\mathrm{d}\boldsymbol{\omega} &= \int_{\prod_{l=0}^{d}[-\pi/h_l,\pi/h_l)} \cos(\boldsymbol{\omega} \cdot (\boldsymbol{j}' - \boldsymbol{j}) \odot \boldsymbol{h})\widetilde{S}(\boldsymbol{\omega})\mathrm{d}\boldsymbol{\omega} \\
&= \int_{\prod_{l=0}^{d}[-\pi/h_l,\pi/h_l)} \exp(i\boldsymbol{\omega} \cdot (\boldsymbol{j}' - \boldsymbol{j}) \odot \boldsymbol{h})\widetilde{S}(\boldsymbol{\omega})\mathrm{d}\boldsymbol{\omega}
\end{aligned}$$

where in the final step, the integral of the $\sin(i\boldsymbol{\omega} \cdot (\boldsymbol{j}' - \boldsymbol{j}) \odot \boldsymbol{h})$ term from $\exp()$ vanishes due to symmetry. This shows that we have the spectral representation of the covariance function, $R[\boldsymbol{h} \odot (\boldsymbol{j}' - \boldsymbol{j})]$, for the given grid resolution (with $\widetilde{S}$ incorporating the required spectral folding, etc).

Note: The imaginary part of $x_{\boldsymbol{j}}$ leads to the same covariance calculations as the real part. However, the cross-covariance between the real imaginary parts is non-zero, so the real and imaginary parts of the sample are not independent. This is because even though the $s_{\boldsymbol{k}\cdot(\boldsymbol{j}'-\boldsymbol{j})}$ terms cancel out pairwise for $\boldsymbol{k}$ and $-\boldsymbol{k}$, the indices where one or more $k_l = -n_l/2$

have no corresponding $k_l = n_l/2$ in $\mathcal{K}_0$. Thus, in order to generate two independent samples in a single complex FFT evaluation, one needs to set such "lower boundary" $\sigma_{\boldsymbol{k}}$ values to zero. This slightly reduces the accuracy of the sampling, but since it lowers the computational cost by a factor 2, and only affects frequencies that typically have small contribution, this may be a preferable approach in practice.

A note on resolution

When choosing the frequency resolution for the Fourier integral discretisation, we are constrained by the desired space resolution $\boldsymbol{h}$, but we are free to choose the number of intervals in each dimension, $\boldsymbol{n}$. However, as a by-product of the FFT construction, this also dictates that the number of evaluation points in space will *also* be $\boldsymbol{n}$. Thus, even if we only desire to evaluate the covariance or simulation at a small number of spatial points, we still need to use enough discretisation points to obtain an accurate Fourier integral approximation. In practice, this means that we might only use a small portion of the computed results.

Transformation code

```
#' Numerically stable sinc function, \code{sin(x)/x}
#'
#' @param x A numeric vector
#'
#' @return `sin(x)/x`
#' @export
#'
#' @examples
sinc <- function(x) {
  f <- sin(x)
  # Use Taylor expansion close to x=0
  # Minimiser of the relative error:
  # sin(x)/x = 1-x^2/6+x^4/120-O(x^6)...
  # Truncated Taylor: 1-x^2/6
  # Approximate relative computational error for sin(x)/x :
  #   |(1 + eps_1)/(1 + eps_2) - 1| <= 2 * eps
  # Set errors equal (approximatively):
  # x^4 / 120 = 2 * eps
  # x = (240 * eps)^{1/4}
  x0 <- (240 * .Machine$double.eps)^(1/4)
  ok <- abs(x) > x0
  if (all(ok)) {
    f <- f / x
  } else {
    f[ok] <- f[ok] / x[ok]
    f[!ok] <- 1 - x[!ok]^2 / 6
  }
  f
}

#' Folded spectrum
#'
#' @param omega
#' @param S_fun
#' @param h
#' @param pointwise
#' @param n_folds
#' @param fold If `TRUE`, computes a folded spectrum Stilde. That is, the sum of
```

```r
#' If `FALSE`, computes
#' a truncated spectrum.
#' @param ...
#'
#' @return
#' @export
#'
#' @examples
fold_spectrum <- function(omega, S_fun, h, pointwise = TRUE, n_folds = 10,
                          fold = TRUE,
                          ...) {
  S <- 0
  done <- FALSE
  if (fold) {
    kk <- seq(-n_folds, n_folds, by = 1)
  } else {
    kk <- 0
  }
  k_ <- rep(1, length(h))
  while (!done) {
    # Shifts the frequencies
    omega_ <- omega + kronecker(
      as.matrix(rep(1, NROW(omega))),
      t(2 * pi * kk[k_] / h)
    )
    # Scalin g by default is 1
    scaling <- 1
    if (!pointwise) {
      for (d in seq_along(h)) {
        scaling <- scaling * sinc(omega_[, d] * h[d] / 2)^2
      }
    }
    # Integration step
    S <- S + S_fun(omega_, ...) * scaling

    #Updates for next iteration
    idx <- which(k_ < length(kk))
    if (length(idx) == 0) {
      done <- TRUE
    } else {
      idx <- max(idx)
      k_[idx] <- k_[idx] + 1
      if (idx < length(h)) {
        k_[(idx + 1):length(h)] <- 1
      }
    }
  }
  global_scaling <- 1
  for (d in seq_along(h)) {
    global_scaling <- global_scaling *
      (omega[, d] >= -pi / h[d]) *
      (omega[, d] < pi / h[d])
  }
```

```r
  S <- S * global_scaling
  S
}

#' Shift needed to be able to apply the Fast Fourier tranform
#'
#' @param x
#' @param dim integer vector of dimension extents, suitable for `array()`.
#' The length of `dim` is the dimension for the intended fft transformation
#'
#' @return
#' @export
#'
#' @examples
fftshift <- function(x, dim = length(x)) {
  stopifnot(prod(dim) == length(x))
  x <- array(as.vector(x), dim = dim)
  if (length(dim) == 1) {
    x <- x[
      c(seq_len(dim[1] / 2) + dim[1] / 2, seq_len(dim[1] / 2)),
      drop = FALSE
    ]
  } else if (length(dim) == 2) {
    x <- x[
      c(seq_len(dim[1] / 2) + dim[1] / 2, seq_len(dim[1] / 2)),
      c(seq_len(dim[2] / 2) + dim[2] / 2, seq_len(dim[2] / 2)),
      drop = FALSE
    ]
  } else {
    stop(paste0("Dimension ", length(dim), " not implemented."))
  }
  x
}
#' Calculates the covariance function from the spectrum
#'
#' @param S
#' @param dim
#' @param h
#'
#' @return
#' @export
#'
#' @examples
S2C <- function(S, dim, h) {
  if (length(dim) == 1) {
    fft <- fftwtools::fftw
  } else if (length(dim) == 2) {
    fft <- fftwtools::fftw2d
  } else {
    stop(paste0("Dimension ", length(dim), " is not implemented."))
  }
  C <- fftshift(Re(fft(fftshift(S, dim))), dim)
  C * prod(2 * pi / h / dim)
```

```
}

#' Spectral sampling
#'
#' @param S Evaluated spectrum
#' @param dim Number of frequency space integration points for each dimension
#' @param h Spatial step sizes
#' @param seed The random seed
#' @param conjugate If `TRUE`, use the complex conjugate property of the
#'   spectral process to generate a single realisation. If `FALSE`, the
#'   real and imaginary parts will be two independent realisations
#'
#' @return
#' @export
#'
#' @examples

S2sample <- function(S, dim, h, seed = NULL, conjugate = FALSE) {
  if (!is.null(seed)) {
    set.seed(seed)
  }
  if (length(dim) == 1) {
    fft <- fftwtools::fftw
  } else if (length(dim) == 2) {
    fft <- fftwtools::fftw2d
  } else {
    stop(paste0("Dimension ", length(dim), " is not implemented."))
  }
  SD <- sqrt(S * prod(2 * pi / h / dim))
  z <- (rnorm(prod(dim)) + 1i*rnorm(prod(dim))) * SD
  if (conjugate) {
    k <- expand.grid(lapply(dim, function(d) seq_len(d) - 1 - d/2))
    # Find symmetry pairs and make them complex conjugate
    if (length(dim) == 1) {
      # Note: this can be implemented much more efficiently...
      pair <- rep(NA_integer_, nrow(k))
      for (idx in seq_len(nrow(k))) {
        pair_ <- k[idx,1] == -k[,1]
        if (any(pair_)) {
          pair[idx] <- which(pair_)
        }
      }
      idx_ <- (k[,1] < 0) & !is.na(pair)
      z[idx_] <- Re(z[pair[idx_]]) - 1i * Im(z[pair[idx_]])
      idx_ <- (k[,1] == 0) | is.na(pair)
      z[idx_] <- Re(z[idx_])
    } else { # length(dim) == 2
      # Note: this can be implemented much more efficiently...
      pair <- rep(NA_integer_, nrow(k))
      for (idx in seq_len(nrow(k))) {
        pair_ <- (k[idx,1] == -k[,1]) & (k[idx,2] == -k[,2])
        if (any(pair_)) {
          pair[idx] <- which(pair_)
```

```
        }
      }
      idx_ <- (k[,1] < 0) & !is.na(pair)
      z[idx_] <- Re(z[pair[idx_]]) - 1i * Im(z[pair[idx_]])
      idx_ <- (k[,1] == 0) & (k[,2] < 0) & !is.na(pair)
      z[idx_] <- Re(z[pair[idx_]]) - 1i * Im(z[pair[idx_]])
      idx_ <- ((k[,1] == 0) & (k[,2] == 0)) | is.na(pair)
      z[idx_] <- Re(z[idx_])
    }
  } else { # !conjugate
    k <- expand.grid(lapply(dim, function(d) seq_len(d) - 1 - d/2))
    # Find lower edges and set to zero
    if (length(dim) == 1) {
      edge <- k[,1] == -dim[1]/2
      z[edge] <- 0
    } else { # length(dim) == 2
      edge <- (k[,1] == -dim[1]/2) | (k[,2] == -dim[2]/2)
      z[edge] <- 0
    }
  }

  z <- fftshift(z, dim)
  if (length(dim) == 1) {
    sample <- (fft(z, dim))
  } else {
    sample <- (fft(matrix(z, dim[1], dim[2])))
  }
  sample
}



#' Title
#'
#' @param dim
#' @param L
#'
#' @return
#' @export
#'
#' @examples
make_x <- function(dim, L) {
  x <- list()
  for (d in seq_along(dim)) {
    x[[paste0("x", d)]] <-
      (seq_len(dim[d]) - 1 - dim[d] / 2) / dim[d] * L[d]
  }
  x
}

#' Locations for sampling results
#'
#' @param dim Number of spectral integration points, for each dimension
```

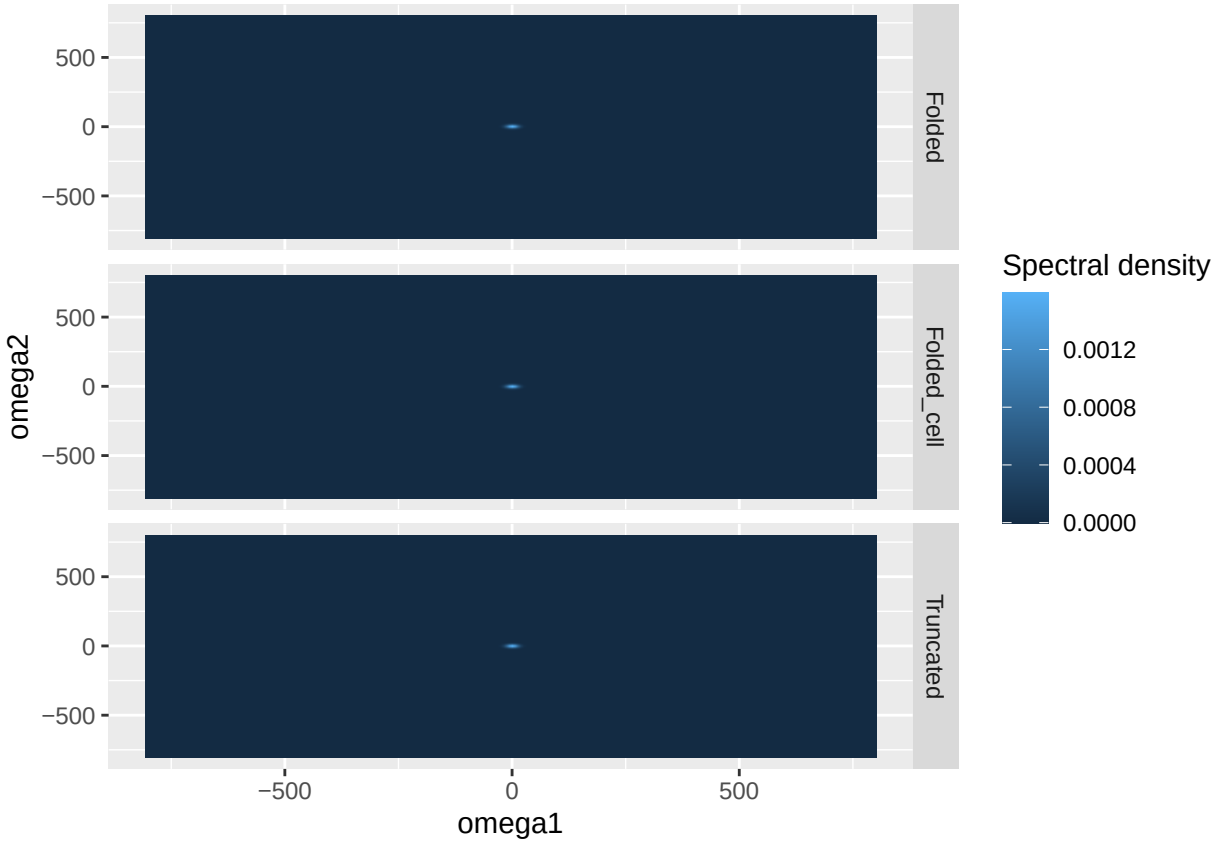
```r
#' @param h The spatial step lengths, for each dimension
#'
#' @return
#' @export
#'
#' @examples
make_x_sampling <- function(dim, h) {
  L <- dim * h
  x <- list()
  for (d in seq_along(dim)) {
    x[[paste0("x", d)]] <-
      (seq_len(dim[d]) - 1) / dim[d] * L[d]
  }
  x
}

#' Frequencies for covariance calculations
#'
#' @param dim
#' @param L
#'
#' @return
#' @export
#'
#' @examples
make_omega <- function(dim, L) {
  h <- L / dim
  w <- list()
  for (d in seq_along(dim)) {
    w[[paste0("w", d)]] <-
      seq(-(dim[d] / 2), dim[d] / 2 - 1, by = 1) / (dim[d] / 2) * pi / h[d]
  }
  w
}

#' Frequencies for sampling
#'
#' @param dim Number of spectral integration points, for each dimension
#' @param h The spatial step lengths, for each dimension
#'
#' @return
#' @export
#'
#' @examples
make_omega_sampling <- function(dim, h) {
  w <- list()
  for (d in seq_along(dim)) {
    w[[paste0("w", d)]] <-
      seq(-(dim[d] / 2), dim[d] / 2 - 1, by = 1) / (dim[d] / 2) * pi / h[d]
  }
  w
}
```

2D example

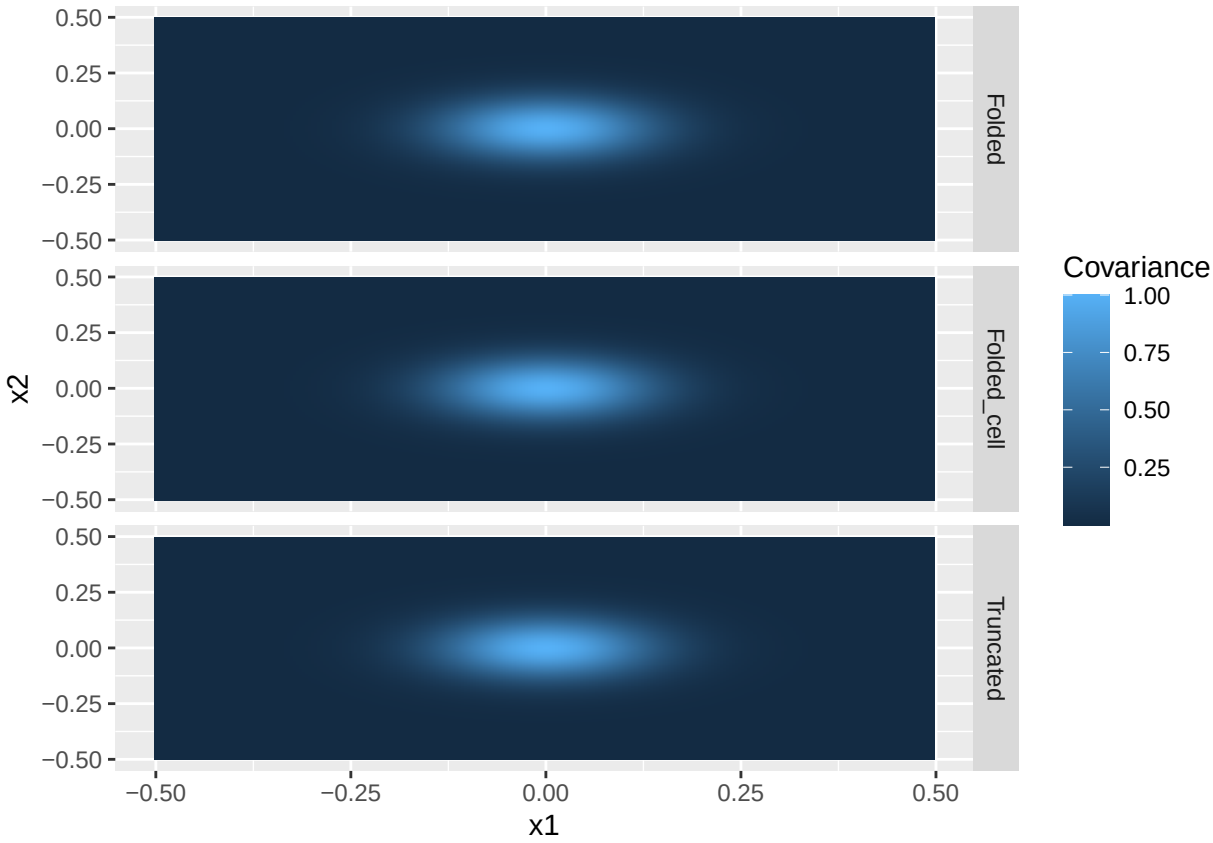
```
S_fun <- function(omega) {
  rho <- 10
  exp(-rowSums(omega^2) / rho^2 / 2) / rho^2 / (2 * pi)
}
n <- c(256, 256)
L <- c(1, 1)
h <- L / n
x_ <- make_x(n, L)
omega_ <- make_omega(n, L)
omega <- as.matrix(expand.grid(omega_))
S_truncated <- S_fun(omega = omega)
S_folded <- fold_spectrum(omega = omega, S_fun = S_fun, h)
S_folded_cell <- fold_spectrum(omega = omega, S_fun = S_fun, h, pointwise = FALSE)
```

```
S_df <-
    cbind(
      as.data.frame(as.matrix(expand.grid(omega_))),
      data.frame(
        Truncated = S_truncated,
        Folded = S_folded,
        "Folded_cell" = S_folded_cell
      )
    ) %>%
      pivot_longer(
        cols = c("Truncated", "Folded", "Folded_cell"),
        names_to = "Type", values_to = "Value"
      )

C_df <-
    cbind(
      as.data.frame(as.matrix(expand.grid(x_))),
      data.frame(
        Truncated = as.vector(S2C(S_truncated, n, h)),
        Folded = as.vector(S2C(S_folded, n, h)),
        "Folded_cell" = as.vector(S2C(S_folded_cell, n, h))
      )
    ) %>%
      pivot_longer(
        cols = c("Truncated", "Folded", "Folded_cell"),
        names_to = "Type", values_to = "Value"
      )
```

```
ggplot(S_df) +
  geom_raster(aes(w1, w2, fill = Value)) +
  facet_grid(vars(Type)) +
  xlab("omega1") + ylab("omega2") +
  labs(fill = "Spectral density")
```




```
ggplot(C_df) +
  geom_raster(aes(x1, x2, fill = Value)) +
  facet_grid(vars(Type)) +
  xlab("x1") + ylab("x2") +
  labs(fill = "Covariance")
```




```
S_df %>% group_by(Type) %>% summarise(N = sum(Value > 0))
```

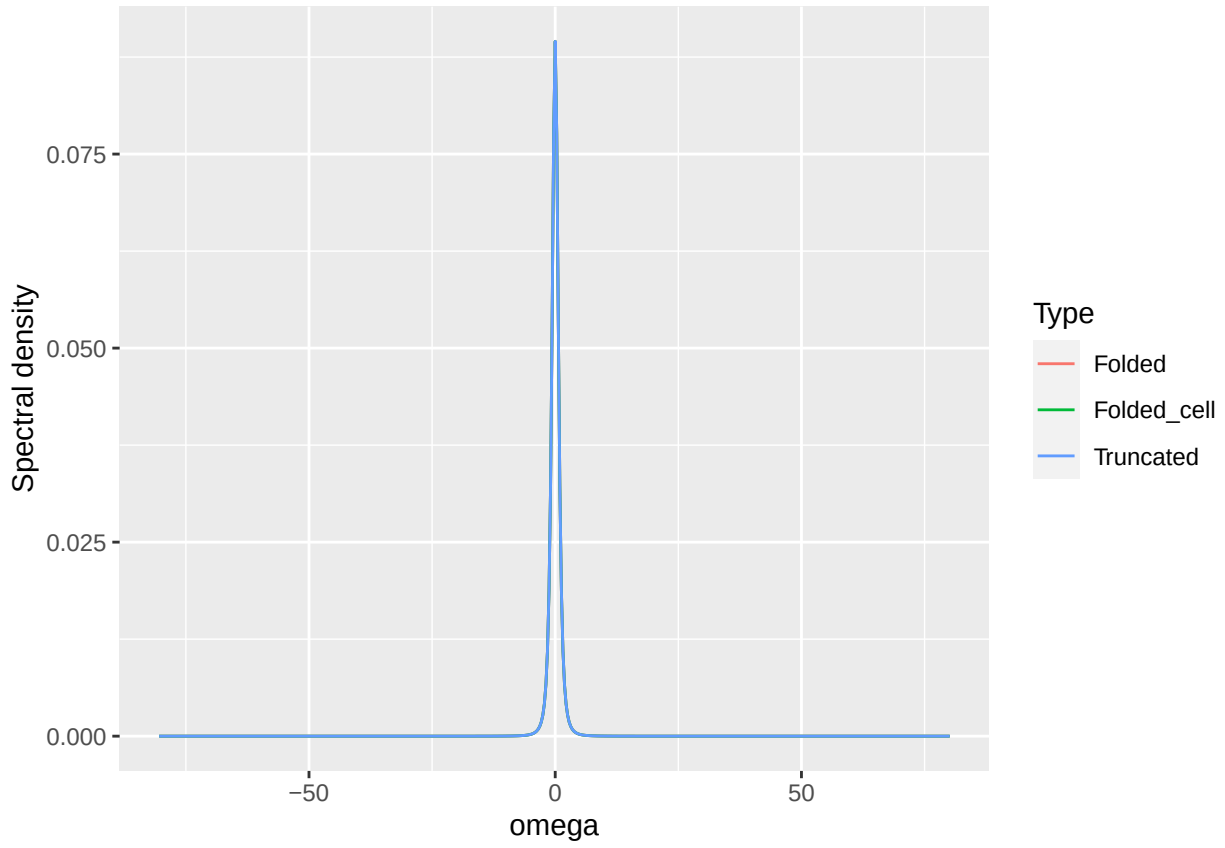
```
## # A tibble: 3 x 2
##   Type            N
##   <chr>       <int>
## 1 Folded      11761
## 2 Folded_cell 11761
## 3 Truncated   11761
```

1D example

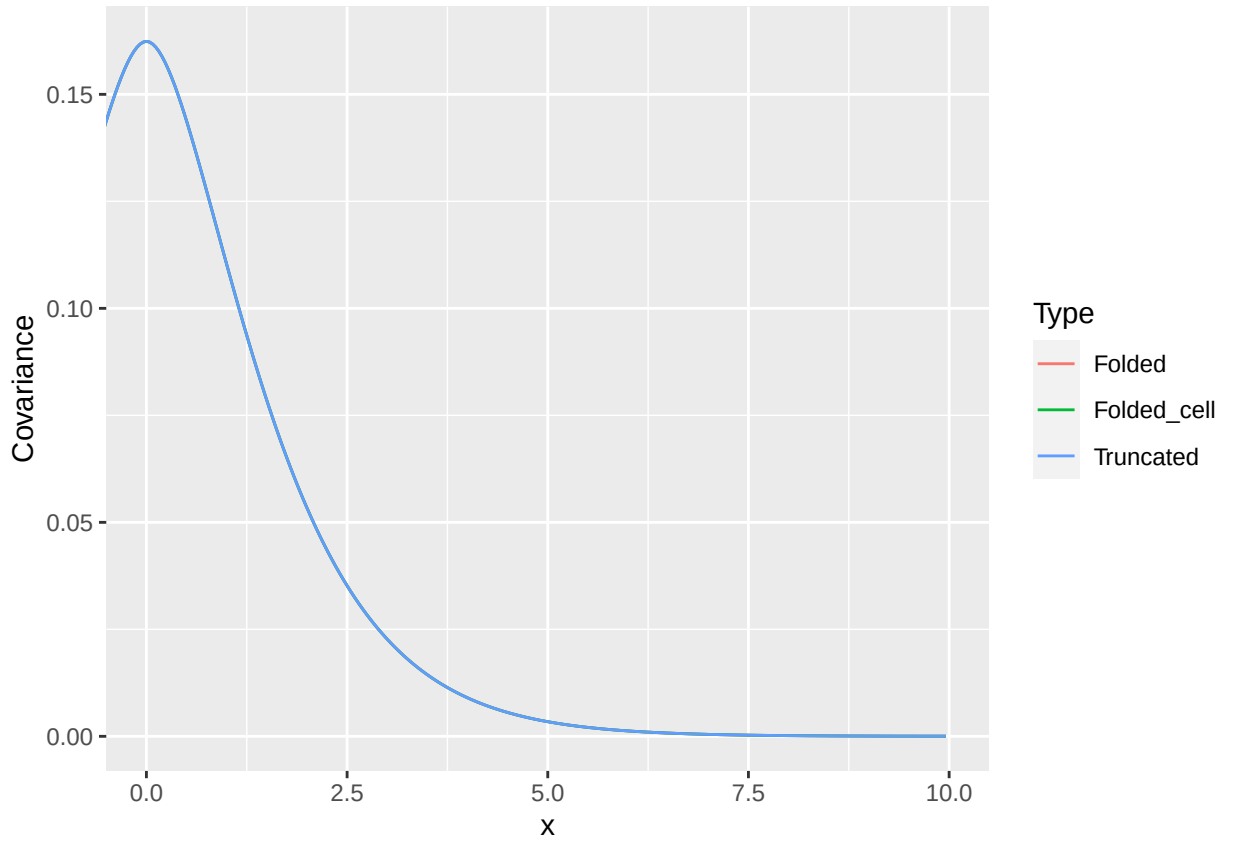
```
#S_fun <- function(omega) {
#  rho <- 2
#  exp(-rowSums(omega^2) * rho^2 / 2) / sqrt(2 * pi) * rho
#}
S_fun <- function(omega) {
  nu <- 1.5
  rho <- 3 / sqrt(8 * nu)
  v <- rho^(2 * nu) * gamma(nu) / gamma(nu + 0.5) / sqrt(4 * pi)
  rho^(2 * nu + 1) / (2 * pi) /
    (1 + rowSums(omega^2) * rho^2)^(nu + 0.5)
}
#S_fun <- function(omega) {
#  rho <- 0.1
#  1/(1-0.9*2*(rowSums(omega^2) * rho^2)^2+(rowSums(omega^2) * rho^2)^4)^2
#}
```

```r
n <- c(512)
L <- c(20)
h <- L / n
x_ <- make_x(n, L)
omega_ <- make_omega(n, L)
omega <- as.matrix(expand.grid(omega_))
S_truncated <- fold_spectrum(omega = omega, S_fun = S_fun, h, fold = FALSE)
S_folded <- fold_spectrum(omega = omega, S_fun = S_fun, h)
S_folded_cell <- fold_spectrum(omega = omega, S_fun = S_fun, h, pointwise = FALSE)


S_df <-
    cbind(
      as.data.frame(as.matrix(expand.grid(omega_))),
      data.frame(
        Truncated = S_truncated,
        Folded = S_folded,
        "Folded_cell" = S_folded_cell
      )
    ) %>%
      pivot_longer(
        cols = c("Truncated", "Folded", "Folded_cell"),
        names_to = "Type", values_to = "Value"
      )

C_df <-
    cbind(
      as.data.frame(as.matrix(expand.grid(x_))),
      data.frame(
        Truncated = as.vector(S2C(S_truncated, n, h)),
        Folded = as.vector(S2C(S_folded, n, h)),
        "Folded_cell" = as.vector(S2C(S_folded_cell, n, h))
      )
    ) %>%
      pivot_longer(
        cols = c("Truncated", "Folded", "Folded_cell"),
        names_to = "Type", values_to = "Value"
      )
```

```r
ggplot(S_df, aes(w1, Value)) +
  geom_line(aes(col = Type)) +
  xlab("omega") + ylab("Spectral density")
```




```
ggplot(C_df, aes(x1, Value)) +
  geom_line(aes(col = Type)) +
  coord_cartesian(xlim = c(0, L / 2)) +
  xlab("x") + ylab("Covariance")
```




```
S_df %>% group_by(Type) %>% summarise(N = sum(Value > 0))
```

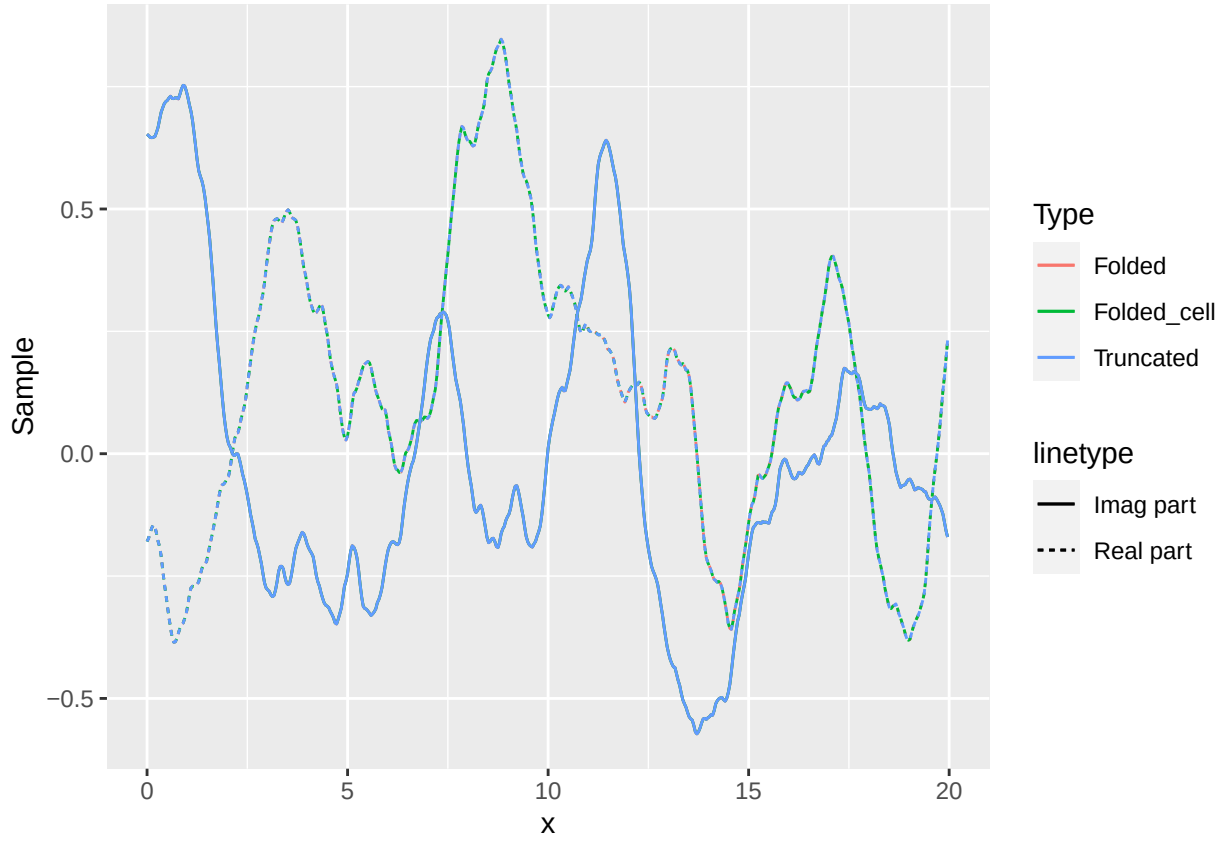
```
## # A tibble: 3 x 2
##   Type            N
##   <chr>       <int>
## 1 Folded        512
## 2 Folded_cell   512
## 3 Truncated     512
```

Sampling example

```
n_sampling <- n * 2
x_ <- make_x_sampling(n_sampling, h)
omega_ <- make_omega_sampling(n_sampling, h)
omega <- as.matrix(expand.grid(omega_))
S_truncated <- S_fun(omega = omega)
S_folded <- fold_spectrum(omega = omega, S_fun = S_fun, h)
S_folded_cell <- fold_spectrum(omega = omega, S_fun = S_fun, h, pointwise = FALSE)
samples_df <-
    cbind(
      as.data.frame(as.matrix(expand.grid(x_))),
      data.frame(
        Truncated = as.vector(S2sample(S_truncated, n_sampling, h, seed = 123L)),
        Folded = as.vector(S2sample(S_folded, n_sampling, h, seed = 123L)),
        "Folded_cell" = as.vector(S2sample(S_folded_cell, n_sampling, h, seed = 123L))
      )
```

```
    ) %>%
      pivot_longer(
        cols = c("Truncated", "Folded", "Folded_cell"),
        names_to = "Type", values_to = "Value"
      )
```

```
ggplot(samples_df %>% filter(x1 < L), aes(x1)) +
  geom_line(aes(y = Re(Value), col = Type, lty="Real part")) +
  geom_line(aes(y = Im(Value), col = Type, lty="Imag part")) +
  coord_cartesian(xlim = c(0, L)) +
  xlab("x") + ylab("Sample")
```



(for long correlation range, the three versions are superimposed, since they used the same random seed, and the spectra are almost identical even after folding/truncation)

References

Cramér, Harald, and M. R. Leadbetter. 2004. *Stationary and Related Stochastic Processes*. Dover Publications, Inc., Mineola, NY.

Lindgren, Georg. 2012. *Stationary Stochastic Processes: Theory and Applications*. CRC Press, Chapman; Hall.

Lindgren, Georg, Holger Rootzen, and Maria Sandsten. 2013. *Stationary Stochastic Processes for Scientists and Engineers*. CRC Press, Chapman; Hall.