TS2Kit: Tensorized Spherical Harmonic Transforms

https://github.com/twmitchel/TS2Kit

Thomas W. Mitchel

tmitchel@jhu.edu

March 2022

TS2Kit (**Version 1.0**) is a self-contained PyTorch library which computes auto-differentiable forward and inverse discrete Spherical Harmonic Transforms (**SHTs**). The routines in *TS2Kit* are based on the seminal *S2Kit* and *SOFT* packages [DH94, HRKM03, KR08], but are designed for evaluation on a GPU. Specifically, the Discrete Legendre Transform (**DLT**) is computed via sparse matrix multiplication in what is essentially a tensorized version of the so-called "semi-naive" algorithm [HRKM03]. This enables parallelization while keeping memory footprint small, and the end result are auto-differentiable forward and inverse SHTs that are fast and efficient in a practical sense. For example, given a spherical signal (tensor) taking values on a 128×128 spherical grid with $b = 4096$ batch dimensions, *TS2Kit* computes a forward SHT followed by an inverse SHT in approximately in tens of milliseconds at floating precision.

Contents

1 Conventions

Of course, this package would be absolutely useless without describing the chosen conventions, so this is where we will begin. Ours deviate slightly from those used in the *S2Kit* and *SOFT* packages, mixing in a bit of Vilenkin [Vil78, VK91] and Varshalovich *et al.* [VMK88].

The SHT is computed with respect to a $Z-Y$ Euler angle spherical coordinate parameterization

$$[0, 2\pi) \times [0, \pi] \ni (\theta, \phi) \mapsto (\cos\theta \sin\phi, \sin\theta \sin\phi, \cos\theta).$$

Note that this corresponds to the parameterization of the Riemann sphere

$$(\theta, \phi) \mapsto \tan\frac{\phi}{2}\, e^{-i\theta}$$

induced by stereographic projection from the north pole.

1.1 Spherical Harmonics

Here we define spherical harmonics with respect to the (normalized) *Wigner-D* functions, with the $(\ell, m, n)-$th function given in $Z-Y-Z$ Euler angles by:

$$D^{\ell}_{mn}(\theta, \phi, \psi) \equiv e^{-im\theta}\, d^{\ell}_{mn}(\phi)\, e^{-in\psi}, \tag{1}$$

where $d^{\ell}_{mn}(\phi)$ are the normalized *Wigner-d* ("little d") functions:

$$\begin{aligned} d^{\ell}_{mn}(\phi) = &\left[\frac{1}{2}\,(2\ell+1)\,(\ell+m)!\,(\ell-m)!\,(\ell+n)!\,(\ell-n)!\right]^{\frac{1}{2}} \\ &\times \sum_{k=\max(0,m-n)}^{\min(\ell+m,\ell-n)} \frac{(-1)^k \left(\cos\frac{\phi}{2}\right)^{2\ell-2k+m-n} \left(\sin\frac{\phi}{2}\right)^{2k-m+n}}{k!\,(\ell+m-k)!\,(\ell-n-k)!\,(n-m+k)!} \end{aligned} \tag{2}$$

The $(l, m)-$th *spherical harmonics* are proportional to the $(l, m, 0)-$th Wigner-D functions, and can be expressed as [VMK88]:

$$\begin{aligned} Y_m^\ell(\theta, \phi) &\equiv \underbrace{(-1)^m (2\pi)^{-\frac{1}{2}}}_{C_m} D_{-m0}^\ell(\theta, \phi, 0) \\ &\equiv C_m \, e^{im\theta} \, d_{-m0}^\ell(\phi) \end{aligned} \quad . \tag{3}$$

Note that the $(l, -m, 0)-$th Wigner-d functions are proportional to the $(l, m)-$th associated Legendre polynomials

$$C_m \, d_{-m0}^\ell(\phi) = \left(\frac{(\ell - m)!}{(\ell + m)!} \right)^{\frac{1}{2}} P_\ell^m(\cos\phi),$$

from which the familiar expression for Y_m^ℓ in terms of P_ℓ^m can be recovered.

1.2 Expanding the Wigner-d functions

Following Edmonds [Edm55], it can be shown that the $(l, -m, 0)-$th Wigner-d functions can be expanded in either an $\ell-$bandlimited cosine or sine series, depending on whether m is even or odd. Specifically, denoting $[m]_2 \equiv m \bmod 2$, if $[m]_2 = 0$, then $d_{-m0}^\ell(\phi)$ can be expressed as a cosine series with

$$\begin{aligned} d_{-m0}^\ell(\phi) &= \sum_{k=0}^{\ell} \xi_{mk}^\ell \cos k\phi \\ \xi_{mk}^\ell &= (-1)^{\frac{m \bmod 4}{2}} \sqrt{\frac{(2\ell + 1)\,(2 - \delta_{k0})}{2}} \, \delta_{\ell \bmod 2, k \bmod 2} \, d_{k-m}^\ell\left(\frac{\pi}{2}\right) d_{k0}^\ell\left(\frac{\pi}{2}\right), \end{aligned} \tag{4}$$

where δ denotes the Kronecker delta. Similarly, if $[m]_2 = 1$, then $d_{-m0}^\ell(\phi)$ can be expressed as a sine series with

$$\begin{aligned} d_{-m0}^\ell(\phi) &= \sum_{k=0}^{\ell} \zeta_{mk}^\ell \sin(k+1)\phi \\ \zeta_{mk}^\ell &= (-1)^{\frac{(m \bmod 4) - 1}{2} + 1} \sqrt{\frac{(2\ell + 1)\,(2 - \delta_{k\ell})}{2}} \, \delta_{\ell \bmod 2, (k+1) \bmod 2} \, d_{k+1\,-m}^\ell\left(\frac{\pi}{2}\right) d_{k+1\,0}^\ell\left(\frac{\pi}{2}\right). \end{aligned} \tag{5}$$

Note that in both expansions, ξ_{mk}^{ℓ} and ζ_{mk}^{ℓ} vanish whenever k is even or odd (depending on the parity of ℓ).

1.3 The Spherical Harmonic Transform

If ψ is a $B-1$ band-limited function in $L^2(S^2, \mathbb{C})$, then it can be expressed as a band-limited sum of spherical harmonics:

$$\psi(\theta, \phi) = \sum_{\ell=0}^{B-1} \sum_{m=-\ell}^{\ell} \mathbf{\Psi}_{\ell m} Y_m^{\ell}(\theta, \phi), \tag{6}$$

with coefficients given by

$$\mathbf{\Psi}_{\ell m} = \int_0^{2\pi} \int_0^{\pi} \psi(\theta, \phi) \, \overline{Y_m^{\ell}}(\theta, \phi) \, \sin\phi \, d\phi \, d\theta. \tag{7}$$

The mapping from ψ to its SH coefficients $\mathbf{\Psi}$ in Equation (7) is the *Forward Spherical Harmonic Transform*, and the reconstruction of ψ from its SH coefficents in Equation (6) is the *Inverse Spherical Harmonic Transform*.

2 Implementation

2.1 Forward Transform

The discrete forward SHT takes as input a $(B-1)-$bandlimited function $\psi \in L^2(S^2, \mathbb{C})$ sampled on a $2B \times 2B$ Driscoll-Healy spherical grid [DH94]:

$$\theta_j = \frac{2\pi j}{2B}, \; \phi_k = \frac{\pi(2k+1)}{4B} \quad 0 \leq j, k < 2B. \tag{8}$$

Then, defining quadrature weights

$$w_B(k) = \frac{2}{B} \sin\left(\frac{\pi(2k+1)}{4B}\right) \sum_{p=0}^{B-1} \frac{1}{2p+1} \sin\left((2k+1)(2p+1)\frac{\pi}{4B}\right) \tag{9}$$
$$0 \leq k < 2B$$

for $\ell \leq B-1$, the $(\ell, m)-$th spherical harmonic coefficient of ψ is given by [DH94, HRKM03, KR08]

$$\begin{aligned} \mathbf{\Psi}_{\ell m} &= \frac{1}{2B} \sum_{j,k=0}^{2B-1} w_B(k)\, \psi(\theta_j, \phi_k)\, \overline{Y_m^\ell}(\theta_j, \phi_k) \\ &\overset{(3)}{=} C_m \sum_{k=0}^{2B-1} w_B(k)\, d^\ell_{-m0}(\phi_k) \underbrace{\left[\frac{1}{2B} \sum_{j=0}^{2B-1} \psi(\theta_j, \phi_k)\, e^{-im\theta_j} \right]}_{\widehat{\psi}_m(\phi_k)}. \end{aligned} \tag{10}$$

Note that the bracketed term in the second equality – denoted $\widehat{\psi}_m(\phi_k)$ – is exactly the $m-$th Fourier coefficient of ψ with respect to θ, and can be computed via the FFT.

Plugging in the cosine and sine series expansions for d^ℓ_{-m0} in Equations (4-5), for $[m]_2 = 0$ we have

$$\mathbf{\Psi}_{\ell m} = C_m \sum_{n=0}^{\ell} \xi^\ell_{mn} \left[\sum_{k=0}^{2B-1} w_B(k)\, \widehat{\psi}_m(\phi_k)\, \cos n\phi_k \right], \tag{11}$$

and for $[m]_2 = 1$

$$\mathbf{\Psi}_{\ell m} = C_m \sum_{n=0}^{\ell} \zeta^\ell_{mn} \left[\sum_{k=0}^{2B-1} w_B(k)\, \widehat{\psi}_m(\phi_k)\, \sin(n+1)\phi_k \right]. \tag{12}$$

Equations (11-12) give a numerical recipe for computing the forward SHT: 1). Compute the discrete Fourier transform in the variable θ via the FFT; 2). Multiply by the quadrature weights $w_B(k)$ and compute the discrete cosine and sine transforms in the variable ϕ; and 3). Multiply by ξ^ℓ_{mn} and ζ^ℓ_{mn} and sum to get the SH coefficients (up to the normalization factor C_m).

Tensorized Forward SHT

In *TS2Kit*, the $(B-1)-$bandlimited forward SHT takes as input a $(B-1)-$bandlimited signal ψ sampled on a DH spherical grid, given by the $2B \times 2B$ tensor

$$\psi = \begin{bmatrix} \psi(\theta_0, \phi_0) & \cdots & \psi(\theta_0, \phi_{2B-1}) \\ \vdots & \ddots & \vdots \\ \psi(\theta_{2B-1}, \phi_0) & \cdots & \psi(\theta_{2B-1}, \phi_{2B-1}) \end{bmatrix}, \tag{13}$$

where θ and ϕ increment along the columns and rows, respectively. Step 1). – the Fourier transform in θ – is implemented by applying PyTorch's native FFT routine along the columns of ψ, giving the $(2B-1) \times 2B$ tensor

$$\widehat{\psi} = \begin{bmatrix} \widehat{\psi}_{-(B-1)}(\phi_0) & \cdots & \widehat{\psi}_{-(B-1)}(\phi_{2B-1}) \\ \vdots & \ddots & \vdots \\ \widehat{\psi}_{B-1}(\phi_0) & \cdots & \widehat{\psi}_{B-1}(\phi_{2B-1}) \end{bmatrix}. \tag{14}$$

Steps 2). – 3). constitute the Discrete Legendre Transform and are computed as follows: Let $\widehat{\psi}_m$ denote the the $2B \times 1$ tensor corresponding to the transpose of the $(m+B)-$th row of $\widehat{\psi}$:

$$\widehat{\psi}_m = \begin{bmatrix} \widehat{\psi}_m(\phi_0) \\ \vdots \\ \widehat{\psi}_m(\phi_{2B-1}) \end{bmatrix}, \qquad -(B-1) \leq m \leq B-1. \tag{15}$$

Now, take $\mathbf{W}$ to be the $2B \times 2B$ diagonal weight matrix

$$\mathbf{W} = \operatorname{diag}\left[w_B(0), \ldots, w_B(2B-1)\right], \tag{16}$$

let $\mathbf{C}$ and $\mathbf{S}$ be the $2B \times 2B$ (orthogonal) DCT and DST matrices, and denote $\boldsymbol{\xi}_m$ and $\boldsymbol{\zeta}_m$ as the $B \times 2B$ matrices with entries

$$[\boldsymbol{\xi}_m]_{ij} = \begin{cases} \xi_{m(j-1)}^{(i-1)} & |m|, j-1 \leq i-1 \\ 0 & \text{otherwise} \end{cases}, \tag{17}$$

and

$$[\boldsymbol{\zeta}_m]_{ij} = \begin{cases} \zeta^{(i-1)}_{m(j-1)} & |m|, j-1 \leq i-1 \\ 0 & \text{otherwise} \end{cases} . \tag{18}$$

We note that,$\boldsymbol{\xi}_m$ and $\boldsymbol{\zeta}_m$ are *sparse* since $\xi^{(i-1)}_{m(j-1)}$ and $\zeta^{(i-1)}_{m(j-1)}$ will vanish whenever $(j-1)$ is even or odd.

Then, steps 2). and 3). can be computed by multiplying $\widehat{\psi}_m$ by either $\boldsymbol{\xi}_m \mathbf{C}\mathbf{W}$ or $\boldsymbol{\zeta}_m \mathbf{S}\mathbf{W}$, depending on whether m is even or odd, then scaling by C_m to recover the SH coefficients of ψ with polar frequency m. That is,

$$\boldsymbol{\Psi}_m = \begin{bmatrix} \boldsymbol{\Psi}_{0m} \\ \boldsymbol{\Psi}_{1m} \\ \vdots \\ \boldsymbol{\Psi}_{(B-1)m} \end{bmatrix} = \begin{cases} C_m \boldsymbol{\xi}_m \mathbf{C}\mathbf{W}\ \widehat{\psi}_m & [m]_2 = 0 \\ C_m \boldsymbol{\zeta}_m \mathbf{S}\mathbf{W}\ \widehat{\psi}_m & [m]_2 = 1 \end{cases} \tag{19}$$

where $\boldsymbol{\Psi}_{\ell m} = 0$ if $\ell < |m|$. In *TS2Kit*, multiplication of the rows of $\widehat{\psi}$ by $\mathbf{C}\mathbf{W}$ and $\mathbf{S}\mathbf{W}$ is implemented as a linear layer since both matrices are fixed for even and odd values of m. The resulting $(2B-1) \times 2B$ tensor is transposed and vectorized (equivalent to a `torch.reshape`) and multiplied by the $(2B-1)B \times (2B-1)2B$ sparse matrix corresponding to the direct sum over $-(B-1) \leq m \leq (B-1)$ of $\boldsymbol{\xi}_m$ and $\boldsymbol{\zeta}_m$ interleaved. $\mathbf{C}\mathbf{W}$, $\mathbf{S}\mathbf{W}$, and the $(2B-1)B \times (2B-1)2B$ sparse matrix are pre-computed and stored on device memory at inference.

2.2 Inverse Transform

The beauty of the DH sampling theorem is that $\boldsymbol{\xi}_m \mathbf{C}\mathbf{W}$ and $\boldsymbol{\xi}_m \mathbf{C}\mathbf{W}$ are the left generalized inverses of $(\boldsymbol{\xi}_m \mathbf{C})^\top$ and $(\boldsymbol{\zeta}_m \mathbf{S})^\top$, respectively, with

$$\boldsymbol{\xi}_m \mathbf{C}\mathbf{W} (\boldsymbol{\xi}_m \mathbf{C})^\top = \begin{bmatrix} \mathbf{0}_{|m|} & \mathbf{0}_{|m|} \\ \mathbf{0}_{|m|} & I_{(B-1)-|m|} \end{bmatrix},$$
$$\boldsymbol{\xi}_m \mathbf{C}\mathbf{W} (\boldsymbol{\zeta}_m \mathbf{S})^\top = \begin{bmatrix} \mathbf{0}_{|m|} & \mathbf{0}_{|m|} \\ \mathbf{0}_{|m|} & I_{(B-1)-|m|} \end{bmatrix},$$

where $\mathbf{0}_{|m|}$ is the $|m| \times |m|$ zero matrix and $I_{(B-1)-|m|}$ is the $(B-1)-|m| \times (B-1)-|m|$ identity matrix. From this, computing the inverse SHT is straight-forward.

In *TS2Kit*, the $(B-1)-$bandlimited inverse SHT takes as input a $(2B-1) \times B$

array of SH coefficients, with m and ℓ incremented along the columns and rows, respectively:

$$\boldsymbol{\Psi} = \begin{bmatrix} 0 & 0 & \cdots & 0 & \boldsymbol{\Psi}_{(B-1)-(B-1)} \\ 0 & 0 & \cdots & \boldsymbol{\Psi}_{(B-2)-(B-2)} & \boldsymbol{\Psi}_{(B-1)-(B-2)} \\ \vdots & \vdots & \iddots & \vdots & \vdots \\ 0 & \boldsymbol{\Psi}_{1-1} & \cdots & \boldsymbol{\Psi}_{(B-2)-1} & \boldsymbol{\Psi}_{(B-2)-1} \\ \boldsymbol{\Psi}_{00} & \boldsymbol{\Psi}_{10} & \cdots & \boldsymbol{\Psi}_{(B-2)0} & \boldsymbol{\Psi}_{(B-2)0} \\ 0 & \boldsymbol{\Psi}_{11} & \cdots & \boldsymbol{\Psi}_{(B-2)1} & \boldsymbol{\Psi}_{(B-2)1} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & \boldsymbol{\Psi}_{(B-2)(B-2)} & \boldsymbol{\Psi}_{(B-1)(B-2)} \\ 0 & 0 & \cdots & 0 & \boldsymbol{\Psi}_{(B-1)(B-1)} \end{bmatrix}. \tag{20}$$

Letting $\boldsymbol{\Psi}_m$ denote the transpose of the $(m+B)-$th row of $\boldsymbol{\Psi}$ as in Equation (19) – the SH coefficients with polar frequency m – we can compute the inverse DLT by multiplying by either $(\boldsymbol{\xi}_m \mathbf{C})^\top$ or $(\boldsymbol{\zeta}_m \mathbf{S})^\top$ (depending on the parity of m) to recover the transpose of the $(m+B)-$th row of $\widehat{\psi}$ as in Equation (15):

$$\widehat{\psi}_m = \begin{cases} (\boldsymbol{\xi}_m \mathbf{C})^\top \boldsymbol{\Psi}_m & [m]_2 = 0 \\ (\boldsymbol{\zeta}_m \mathbf{S})^\top \boldsymbol{\Psi}_m & [m]_2 = 1 \end{cases}. \tag{21}$$

After recovering $\widehat{\psi}$, the inverse FFT is applied along the columns to get ψ.

2.3 Composing the Transforms

Denote $\mathcal{F}$ and $\mathcal{I}$ as the discrete $(B-1)-$bandlimited forward and inverse SHTs, respectively, and let ψ be a spherical signal sampled on the corresponding $2B \times 2B$ DH spherical grid as in Equation (13). The "catch" in the DH sampling theorem is that

$$(\mathcal{I} \circ \mathcal{F})(\psi) = \psi \quad \Longleftrightarrow \quad \psi \text{ is a } (B-1) - \text{bandlimited function.}$$

In other words, mapping ψ to its SH coefficients via the forward SHT, then applying the inverse SHT to the coefficients returns an exact copy of ψ only in the case when ψ is a $(B-1)-$bandlimited spherical function.

Of course, it will almost never be the case that a real-world signal will be an exact $(B-1)-$bandlimited function and thus cannot be exactly recovered from its

SH coefficients given by the discrete forward SHT. Instead, applying the forward transform followed by the inverse transform will result in some degree of spectral aliasing, *i.e.* $(\mathcal{I} \circ \mathcal{F})(\psi)$ will appear to be a slightly blurred version of ψ. It is worth noting that this is not a "bug" in *TS2Kit*, but rather a "feature" of the existing spherical sampling theorems. If you are aware of a spherical sampling theorem for which $(\mathcal{I} \circ \mathcal{F})(\psi) = \psi$ for *any* signal sampled on the prescribed spherical grid, please reach out to me.

That said, it is always the case that

$$(\mathcal{F} \circ \mathcal{I})(\mathbf{\Psi}) = \mathbf{\Psi},$$

for *any* $(2B - 1) \times B$ array of SH coefficients $\mathbf{\Psi}$ as in Equation (20).

3 *TS2Kit*

3.1 Set up

To use *TS2Kit*, simply copy the `TS2Kit` folder into your project directory.

Setting the cache path

Several tensors are pre-computed at initialization and at higher bandlimits ($B \geq 64$) this can take some time. To avoid re-computing these quantities every initialization, the modules will check if the tensors have been saved in a cache directory and either A). load the tensors directly from the cache; or B). compute the tensors and save them to the cache directory so they can be loaded next time the modules are initialized.

To enable caching, choose a directory on your machine to serve as the cache folder and set the variable `cacheDir` at the top of the `ts2kit.py` file to the absolute path of the directory, *e.g.*

```
cacheDir = '/absolute/path/to/cache'
```

The cache directory can be cleared (of `.pt` files) at anytime by importing and running the `clearCache` function:

```
from ts2kit.ts2kit import clearCache

clearCache()
```

3.2 The Forward and Inverse SHTs

The front-end of *TS2Kit* consists of the `torch.nn.Module` classes `FTSHT` and`ITSHT`, corresponding to the forward and inverse SHT, respectively. At initialization, the modules are passed an integer argument B which determines the bandlimit of the forward and inverse SHT, *e.g.*

```
from ts2kit.ts2kit import FTSHT, ITSHT

## Bandlimit
B = 64

## Initialize the (B-1)-bandlimited forward SHT
FT = FTSHT(B)

## Initialize the (B-1)-bandlimited inverse SHT
IT = ITSHT(B)
```

`FTSHT`: The Forward SHT

Initialized with bandlimit B, calling the `FTSHT` module applies the forward SHT to a signal composed of several input spheres. Specifically, inputs are $b \times 2B \times 2B$ real or complex `torch` tensors, where b is the batch dimension and the second and third dimensions increment over the values of θ and ϕ in the $2B \times 2B$ DH spherical grid as in Equation (13). For example, given a tensor `psi` of size $100 \times 128 \times 128$ ($b = 100, B = 64$), the element `psi[26, 47, 12]` is the value of the spherical signal in batch dimension 26 at coordinates (θ_{46}, ϕ_{11}) in the DH spherical grid as defined in Equation (8). To assist in sampling to a DH grid, the user can import the `gridDH` function, which takes as input a fixed bandlimit B and returns two $2B \times 2B$ tensors `theta` and `phi` giving the spherical coordinates of the corresponding DH grid indices.

The forward call returns a $b \times (2B-1) \times B$ complex `torch` tensor giving the array

of SH coefficients – with m and ℓ incremented along the second and third dimensions, respectively, as in Equation (20) – of spherical signals for each batch dimension of the input tensor. For example, passing the real or complex $100 \times 128 \times 128$ tensor `psi` to the module returns the complex $100 \times 127 \times 64$ tensor of SH coefficients:

```
F = FTSHT(B)
psiCoeff = F(psi)
```

The $(\ell, m)-$th SH coefficients in batch dimension c can be accessed via `psiCoeff[c, m+B, l]`, *e.g.* for $\ell = 5, m = -5, c = 12$, the corresponding SH coefficient is `psiCoeff[12, 59, 5]`. For $\ell < |m|$, the values in `psiCoeff` will be zero.

`ITSHT`: The Inverse SHT

Initialized with bandlimit B, calling the `ITSHT` module applies the inverse SHT to a signal composed of several arrays of SH coefficients. Inputs are $b \times (2B-1) \times B$ complex `torch` tensors consisting of b channels of SH coefficent arrays, structured in exactly the same way as the output of the `FTSHT` module. The forward call returns a $b \times 2B \times 2B$ *complex* `torch` tensor corresponding to the spherical signals reconstructed from the SH coefficients in each batch dimension:

```
I = ITSHT(B)
psi = I(psiCoeff)
```

The output tensor is complex-valued, so if the input SH coefficient tensor corresponds to a real-valued signal then the imaginary part of the output tensor will be zero and it can be cast to a real tensor (*e.g.* `psi.real`) without loss of information.

3.3 Typing

The `FTSHT` and `ITSHT` modules are initialized at double precision. That is, the forward call of `FTSHT` maps tensors of type `torch.double` (real-valued) or `torch.cdouble` (complex-valued) to tensors of type `torch.cdouble`. Similarly, the forward call of `ITSHT` maps tensors of type `torch.cdouble` to tensors of the same type.

If desired, the modules can be cast to floating precision at initialization, *i.e.* `FTSHT(B).float()` and `ITSHT(B).float()`. In this case, the forward call of `FTSHT`

maps tensors of type `torch.float` and `torch.cfloat` to tensors of type `torch.cfloat` and that of `ITSHT` maps tensors of type `torch.cfloat` to tensors of the same type.

Casting to floating precision results in half the memory overhead and about an order of magnitude decrease in run-time at the cost of several orders of magnitude in accuracy. For example, given a tensor of double-precision SH coefficients on the GPU

```
device = torch.device('cuda')

Psi = torch.view_as_complex(2*(torch.rand(b, 2*B -1, B, 2).double() -
0.5)).to(device)

for m in range(-(B-1), B):
    for l in range(0, B):
        if (l * l < m * m):
            F[:, m + (B-1), l] = 0.0;
```

one can expect the following error to be very, very small

```
F = FTSHT(B).to(device)
I = ITSHT(B).to(device)

Psi2 = F(I(Psi))

## This error should be very, very  small
error = torch.sum(torch.abs(Psi-Psi2)) / torch.sum(torch.abs(Psi))
```

Casting to floating precision will result in a significant speed up and less overhead, but a larger error:

```
Psi_f = Psi.cfloat();
F_f = FTSHT(B).to(device).float()
I_f = ITSHT(B).to(device).float()

## This should run about an order of magnitude faster
```

```
Psi2_f = F_f(I_f(Psi_f))

## This error will be much larger
error = torch.sum(torch.abs(Psi-Psi2)) / torch.sum(torch.abs(Psi))
```

This is not say that the `FSHT` and `ISHT` modules are "slow" at double precision any more than they are "inaccurate" at floating precision. Rather, it all depends on the application. The `test_ts2kit.ipynb` notebook included in the `ts2kit` folder can be used to compare the transforms at different precisions and bandlimits to see see what makes sense for your use case.

References

[DH94] James R Driscoll and Dennis M Healy. Computing Fourier Transforms and Convolutions on the 2-Sphere. *Advances in Applied Mathematics*, 15(2):202–250, 1994.

[Edm55] Alan R Edmonds. Angular Momentum in Quantum Mechanics. Technical report, CERN, 1955.

[HRKM03] Dennis M Healy, Daniel N Rockmore, Peter J Kostelec, and Sean Moore. FFTs for the 2-Sphere-Improvements and Variations. *Journal of Fourier Analysis and Applications*, 9(4):341–385, 2003.

[KR08] Peter J Kostelec and Daniel N Rockmore. FFTs on the Rotation Group. *Journal of Fourier Analysis and Applications*, 14(2):145–179, 2008.

[Vil78] N. Ja. Vilenkin. *Special Functions and the Theory of Group Representations*, volume 22. American Mathematical Soc., 1978.

[VK91] N. Ja. Vilenkin and A. U. Klimyk. Representation of Lie Groups and Special Functions: Volume 1: Simplest Lie Groups, Special Functions and Integral Transforms (Mathematics and its Applications), 1991.

[VMK88] Dmitri Aleksandrovich Varshalovich, Anatolij Nikolaevic Moskalev, and Valerii Kel'manovich Khersonskii. *Quantum Theory of Angular Momentum*. World Scientific, 1988.