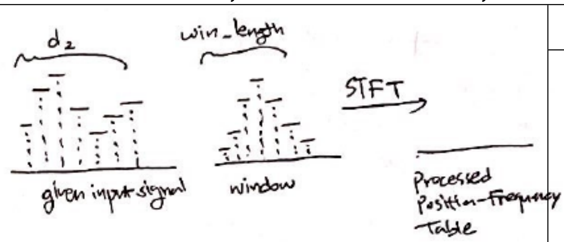
Builtin Fourier Transforms

`torch.stft`

```
torch.stft(input, n_fft, hop_length=None, win_length=None, window=None, center=True,
pad_mode='reflect', normalized=False, onesided=True)
```



Require

- $|input| = (d_1, d_2)$ or (d_2) (`rank` $= 1$ or 2)
- $0 < n\_fft < 2 \cdot d_2$ if $center == True$
- $0 < n\_fft \leq d_2$ if $center == False$
- $hop\_length > 0$
  - if $hop\_length == None$, then let $hop\_length = \lfloor \frac{n\_fft}{4} \rfloor$.
- $0 < win\_length \leq n\_fft$
  - if $win\_length == None$, then let $win\_length = n\_fft$.
- $window == None$ or $|window| = (win\_length)$ (rank 1)

Guarantees

- $|y| = (d_1, a, b, 2)$ or $(a, b, 2)$ ... from the proof tree

Comment

- 음향분석에서 자주 쓰이는 short-time Fourier transform.
- 마지막 텐서 부분은 real과 complex 부분을 가리키는 두 텐서. `real` 함수가 여기서 쓰임

$$\frac{\begin{array}{l} \sigma \vdash E \Rightarrow e, c \\ \sigma \vdash window \Rightarrow w, c_{win} \quad \text{if } window \neq None \text{, otherwise } c_{win} = \emptyset \\ hop\_length = \lfloor \frac{n\_fft}{4} \rfloor \quad \text{if } hop\_length == None \\ win\_length = n\_fft \quad \text{if } win\_length == None \\ n = e[\texttt{rank}(e)] \\ c_{dim} = \{(window = None \vee w = (win\_length)) \wedge (0 < hop\_length) \wedge (0 < win\_length \leq n\_fft)\} \\ c_{fft} = \{(\texttt{if } center \texttt{ then } 0 < n\_fft < 2 \cdot n \texttt{ else } 0 < n\_fft < n)\} \\ a = \texttt{if } onesided \texttt{ then } \lfloor \frac{n\_fft}{2} \rfloor + 1 \texttt{ else } n\_fft \\ b = \texttt{if } center \texttt{ then } \lfloor \frac{n}{hop\_length} \rfloor + 1 \texttt{ else } \lfloor \frac{n - n\_fft}{hop\_length} \rfloor + 1 \\ e' = \texttt{if rank}(e) == 2 \texttt{ then } (e[1], a, b, 2) \texttt{ else } (a, b, 2) \end{array}}{\sigma \vdash \texttt{stft}(E, n\_fft, hop\_length = None, win\_length = None, ..., onesided = True) \Rightarrow e', c \cup c_{win} \cup c_{dim} \cup c_{fft}}$$

Example Codes:

```
a = torch.randn(5, 10000)
print(a.shape)                      # (5, 10000)
print(torch.stft(a, 4).shape)       # (5, 3, 10001, 2)
print(torch.stft(a, 200, 100).shape) # (5, 101, 101, 2)
print(torch.stft(a, 200, 100, center=False).shape) # (5, 101, 99, 2)
print(torch.stft(a, 200, 100, center=False, onesided=False).shape) # (5, 200, 99, 2)
```

torch.rfft

<table>
<tr><td colspan="2"><code>torch.rfft(input, signal_ndim, normalized=False, onesided=True)</code></td></tr>
<tr><td rowspan="6">input as polynomial ft, $j \triangleq \sqrt{-1}$
Compute $ft(e^{\frac{2\pi j k}{N}})$ for each k.
Onesided option is because $\overline{e^{\theta j}} = e^{-\theta j}$</td><td>Require</td></tr>
<tr><td>

- $|input| = (d_1, d_2, \ldots, d_k)$
- $signal\_ndim \in \{1, 2, 3\}$
- $signal\_ndim \leq k$

</td></tr>
<tr><td>Guarantees</td></tr>
<tr><td>

- $|y| = (d_1, d_2, \ldots, d_{k-1}, d'_k, 2)$ where
  - $d'_k = \texttt{if } onesided \texttt{ then } \lfloor \frac{d_k}{2} \rfloor + 1 \texttt{ else } d_k$

</td></tr>
<tr><td>Comment</td></tr>
<tr><td>

- Real-to-complex Discrete Fourier Transform
- 마지막 텐서 부분은 real과 complex 부분을 가리키는 두 텐서. `real` 함수가 여기서 쓰임

</td></tr>
</table>

$$\frac{\begin{array}{l} \sigma \vdash E \Rightarrow e, c \\ k = \texttt{rank}(e) \\ c_{dim} = \{(signal\_ndim \in \{1, 2, 3\}) \wedge (signal\_ndim \leq k)\} \\ f = \texttt{if } onesided \texttt{ then } \lfloor \frac{e[k]}{2} \rfloor + 1 \texttt{ else } e[k] \\ e' = (e[1], e[2], \ldots, e[k-1], f, 2) \end{array}}{\sigma \vdash \texttt{rfft}(E, signal\_ndim, normalized = False, onesided = True) \Rightarrow e', c \cup c_{dim}}$$

torch.fft

<table>
<tr><td colspan="2"><code>torch.fft(input, signal_ndim, normalized=False)</code></td></tr>
<tr><td rowspan="6"></td><td>Require</td></tr>
<tr><td>

- $|input| = (d_1, d_2, \ldots, d_k)$
- $d_k = 2$ (real and complex values)
- $signal\_ndim \in \{1, 2, 3\}$
- $signal\_ndim \leq k - 1$

</td></tr>
<tr><td>Guarantees</td></tr>
<tr><td>

- $|y| = (d_1, d_2, \ldots, d_{k-1}, d_k) = |input|$ (same shape)

</td></tr>
<tr><td>Comment</td></tr>
<tr><td>

- Complex-to-complex Discrete Fourier Transform
- 마지막 텐서 부분은 real과 complex 부분을 가리키는 두 텐서. `real` 함수가 여기서 쓰임

</td></tr>
</table>

$$\frac{\begin{array}{l} \sigma \vdash E \Rightarrow e, c \\ k = \texttt{rank}(e) \\ c_{last} = \{(e[k] = 2)\} \\ c_{dim} = \{(signal\_ndim \in \{1, 2, 3\}) \wedge (signal\_ndim \leq k - 1)\} \end{array}}{\sigma \vdash \texttt{fft}(E, signal\_ndim, normalized = False) \Rightarrow e, c \cup c_{last} \cup c_{dim}}$$

Window Declarations

`torch.hann_window`

`torch.hann_window(window_length, periodic=True, dtype=None, layout=torch.strided, device=None, requires_grad=False)`	
	Require
	• $window\_length \geq 0$
	Guarantees
	• $\lvert y \rvert = (window\_length)$
	Comment
	• $\sin^2$ 가중치의 주기성 윈도우 선언

$$\frac{}{\sigma \vdash \texttt{hann\_window}(window\_length, ...) \Rightarrow (window\_length), \{(window\_length \geq 0)\}}$$

`torch.bartlett_window`

`torch.bartlett_window(window_length, periodic=True, dtype=None, layout=torch.strided, device=None, requires_grad=False)`	
	Require
	• $window\_length \geq 0$
	Guarantees
	• $\lvert y \rvert = (window\_length)$
	Comment
	• 절대값함수 가중치 윈도우 선언

$$\frac{}{\sigma \vdash \texttt{bartlett\_window}(window\_length, ...) \Rightarrow (window\_length), \{(window\_length \geq 0)\}}$$

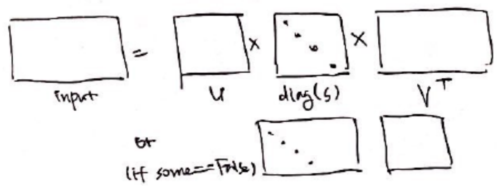
`torch.hamming_window`

`torch.hamming_window(window_length, periodic=True, alpha=0.54, beta=0.46, dtype=None,layout=torch.strided, device=None, requires_grad=False)`	
	Require
	• $window\_length \geq 0$
	Guarantees
	• $\lvert y \rvert = (window\_length)$
	Comment
	• 삼각함수 가중치의 주기성 윈도우 선언

$$\frac{}{\sigma \vdash \texttt{hamming\_window}(window\_length, ...) \Rightarrow (window\_length), \{(window\_length \geq 0)\}}$$

Linear Algebra

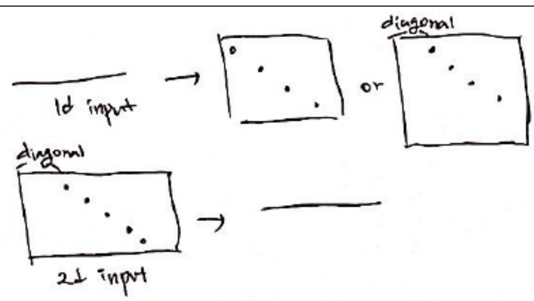
`torch.svd`

<table>
<tr><td colspan="2"><code>torch.svd(input, some=True, compute_uv=True, out=None)</code></td></tr>
<tr><td rowspan="6">
</td><td>Require</td></tr>
<tr><td>• $|input| = (*, m, n) \quad (\texttt{rank} \geq 2)$</td></tr>
<tr><td>Guarantees</td></tr>
<tr><td>• $|U| = (*, m, m), |S| = (*, m), |V| = ...$
  – if $(compute\_uv) \wedge (some)$ then $|V| = (*, n, m) \quad$ (as default)
  – otherwise, $|V| = (*, n, n)$
• $|y| = (|U|, |S|, |V|)$</td></tr>
<tr><td>Comment</td></tr>
<tr><td>• (U, S, V) 형태의 $(Tensor, Tensor, Tensor)$ 출력 반환
• $input = U \times \mathrm{diag}(S) \times V^T$
• out-텐서 인자가 있는 함수</td></tr>
</table>

$$\frac{\begin{array}{l} \sigma \vdash E \Rightarrow e, c \\ k = \texttt{rank}(e) \\ u = e[1{:}k-2]@(e[k-1], e[k-1]) \\ s = e[1{:}k-2]@(e[k-1]) \\ v_{tail} = \texttt{if } (compute\_uv) \wedge (some) \texttt{ then } (n, m) \texttt{ else } (n, n) \\ v = e[1{:}k-2]@v_{tail} \\ c_{dim} = \{(k \geq 2)\} \end{array}}{\sigma \vdash \texttt{svd}(E, some = True, compute\_uv = True, out = None) \Rightarrow (u, s, v), c \cup c_{dim}}$$

3-tensor tuple로 반환

`torch.diag`

<table>
<tr><td colspan="2"><code>torch.diag(input, diagonal=0, out=None)</code></td></tr>
<tr><td rowspan="6">
</td><td>Require</td></tr>
<tr><td>• $|input| = (d_1)$ or (d_1, d_2). (rank 1 or 2)
• if $\texttt{rank} = 1$, then there is no requirement.
• if $\texttt{rank} = 2$, then $-d_1 \leq diagonal \leq d_2$</td></tr>
<tr><td>Guarantees</td></tr>
<tr><td>• if $\texttt{rank} = 1$, then $|y| = (d_1 + |diagonal|, d_1 + |diagonal|)$
• if $\texttt{rank} = 2$, then $|y| = (\min\{d_1, d_2, d_1 + diagonal, d_2 - diagonal\})$</td></tr>
<tr><td>Comment</td></tr>
<tr><td>• 1차원, 2차원 텐서 입력에 따라 역할이 달라짐
• out-텐서 인자가 있는 함수</td></tr>
</table>

$$\frac{\begin{array}{l} \sigma \vdash E \Rightarrow e, c \\ k = \texttt{rank}(e) \\ c_{dim} = \{(k \in \{1, 2\}) \wedge (k = 1 \vee (-e[1] \leq diagonal \leq e[2]))\} \\ e' = \texttt{if } k = 1 \texttt{ then } (e[1] + |diagonal|, e[1] + |diagonal|) \texttt{ else } (\min\{e[1], e[2], e[1] + diagonal, e[2] - diagonal\}) \end{array}}{\sigma \vdash \texttt{diag}(E, diagonal = 0, out = None) \Rightarrow e', c \cup c_{dim}}$$