Introduction to Video Processing

HW5

Hong Shiang Lin

National Taipei University

hongshianglin@gm.ntpu.edu.tw

March 20, 2025

HW5: DFT Decomposition into 1D Fast Fourier Transform

You should complete the following tasks:

- Proof: that 3D DFT and IDFT can be decomposed into multi-pass 1D DFT.
- Utilize numpy 1D FFT in your 3D DFT.
- Implement successive doubling method by yourself.
- Provide a HW report (see report guidelines).

Proof

Show the following formulas can be decomposed into multi-pass of 1D DFTs.

- 3D DFT: $\Psi(f_x, f_y, f_t) = \sum_{x=0}^{w-1} \sum_{y=0}^{h-1} \sum_{t=0}^{n-1} \psi(x, y) e^{-j2\pi(\frac{f_x x}{w} + \frac{f_y y}{h} + \frac{f_t t}{n})}$.
- 3D IDFT: $\psi(x, y, t) = \frac{1}{whn} \sum_{f_x=0}^{w-1} \sum_{f_y=0}^{h-1} \sum_{f_t=0}^{n-1} \Psi(f_x, f_y, f_t) e^{j2\pi(\frac{f_x x}{w} + \frac{f_y y}{h} + \frac{f_t t}{n})}$.

Numpy 1D FFT Integration

(Tracing on revised **fourier_simulator.py**)

```
xformer = FourierTransformer()

# Compute frequency response
print('Computing Fourier transform for the sinusoidal signal...')
frequency_signal = xformer.dft_video(spatial_temporal_signal, fast=True
    )

# Cache the results for testing or debugging
cache_signal(folder_out + '/frequency.sg', frequency_signal)
```

- Add *fast* flag to call fast DFT for the video signal.

```python
# Reconstruct spatial temporal signal via inverse fourier transform
print('Recovering the spatial temporal signal by inverse Fourier
    transform...')
recovered_signal = xformer.fast_idft(frequency_signal)

(Ignore...)

# Generate the corresponding spatial temporal signal, it SHOULD be
    changed.
print('Generating new spatial temporal signal according to the shifted
    frequency signal...')
edited_spatial_temporal_signal = xformer.fast_idft(shifted_frequency_
    signal)
```

- Use *fast_idft* to call fast DFT for the inverse Fourier transform.

(Tracing on revised **fourier_transformer.py**)

```python
    '''
    Inputs:
        - video: 3D numpy array with real numbers
    Outputs:
        - output_signal: 3D numpy array with complex numbers
    '''
    def dft_video(self, video, fast=False):
        # Convert the input video type into complex type
        video_complex = video.astype(complex)
        return self.fast_dft(video_complex) if fast == True else self.dft(video_complex)
```

- Add the *fast* flag for *dft_video*.

(Continue)

```python
    def fast_dft(self, input_signal):
        # Get width, height, frames of the input_signal.
        height, width, frames = input_signal.real.shape[0], input_signal.real.shape[1], input_signal.real.shape[2]

        # Get frequency response by computing fourier dft on the input signals
        output_signal = np.zeros([height, width, frames], dtype=complex)

        # TODO #1: Implement 3D fourier transform for output_signal in terms of multi-pass 1D DFT
        # You can call either np.fft.fft for fast 1D DFT (wrap it into self.fft_numpy) or call your self implementation (define them into
        # self.fft)

        return output_signal
```

- Implement the DFT with multi-pass 1D DFT.
- Call either *self.fft_numpy* (supported by *np.fft.fft*) or *self.fft* (your implementation).

(Continue)

```python
    def fast_idft(self, input_signal):
        # Get width, height, frames of the input_signal.
        height, width, frames = input_signal.real.shape[0], input_signal.real.shape[1], input_signal.real.shape[2]

        # Get frequency response by computing fourier dft on the input signals
        output_signal = np.zeros([height, width, frames], dtype=complex)

        # TODO #2: Implement 3D inverse fourier transform for output_signal in terms of multi-pass 1D DFT and conjugate complex numbers.
        # You can call either np.fft.fft for fast 1D DFT (wrap it into np.fft_numpy) or call your self implementation (define them into
        # self.fft)

        return output_signal
```

- Implement the IDFT with multi-pass 1D DFT.
- Call either *self.fft_numpy* (supported by *np.fft.fft*) or *self.fft* (your implementation).

(Continue)

```
    def fft_numpy(self, input_signal):
        # Get number of samples of the input_signal.
        num_samples = input_signal.real.shape[0]

        # Get frequency response by computing fourier dft on the input
            signals
        output_signal = np.zeros(num_samples, dtype=complex)

        # TODO #3: Incorporate numpy fft.fft API for the fast 1D DFT

        return output_signal
```

- Incorporate numpy *fft.fft* API

Implement successive doubling method

(Continue)

```
    def fft(self, input_signal):
        # Get number of samples of the input_signal.
        num_samples = input_signal.real.shape[0]

        # Get frequency response by computing fourier dft on the input
            signals
        output_signal = np.zeros(num_samples, dtype=complex)

        # TODO #4: Implement the successive doubling method provided in
             the course slides
        # Note that the number of samples should be power of 2 (handle
            the signal shape before calling this API)

        return output_signal
```

- Implement the successive doubling method.

Report Guidelines

- Describe your implementation. (Do NOT paste code screen shot figures)
- Describe your understanding of theories.
- Describe your experiment settings and results.
  - Examine and analyze differences in results of HW4 and HW5.
  - Share any additional noteworthy insights.
  - Raise any additional problems or sharing any additional insights.
- Write your division of labor (same rules as those of previous homework).
- Only .pdf file format is allowed.

Grading Criteria

- Richness of your experiments.
- Readability of the report.
- Clarity of your insights and the division of labor.

Scoring

- Numpy 1D FFT in your 3D DFT (70%).
- Successive doubling method (10%).
- Report (20%)

Submission Guidelines

- Put all the codes into a folder *codes*.
- This time, the required videos/images are automatically put into the subfolder *results* in *codes*.
- Put all the experiemented videos (with additional experiment setting) into a folder *supplementary-material*.
- Put **codes**, **supplementary-material** (optional) and report.pdf into a folder **hw5** (i.e., they are under same level of the file tree)
- Compress **hw5** and specify the file format as hw5.zip.
- Upload hw5.zip onto Digital Platform 3.
- Deadline: 12:00 a.m. on March 27th.

Origin of The Latex Template

The latex template is downloaded from:
https://www.LaTeXTemplates.com

The Author: Vel (vel@latextemplates.com)