Numerical Methods for Financial Derivatives

Hwan C. Lin
Department of Economics
UNC Charlotte

Lecture 8: Fast Fourier Transform using Python

Introduction to Fast Fourier Transform (FFT)

- FFT is a discrete Fourier transform algorithm; see Wolfram.
- FFT was Developed by Cooley and Tukey (1965)
  - James W. Cooley and John W. Tukey. 1965. An algorithm for the machine calculation of complex Fourier series. Mathematics of Computation, 19 (90): 297-301.
- FFT provides a more efficient algorithm for calculating a set of discrete inverse Fourier transforms with sample points that are powers of two.

- Computers and digital processing systems can work with finite sums only. To turn the continuous into the discrete and finite requires that a signal be both **time-limited** and **band-limited**. That is,

$$f(t) = 0 \quad for \quad t \notin [0, L]$$

$$\hat{f}(s) = 0 \quad for \quad s \notin [-B, B] \quad or \quad s \notin [0, 2B]$$

- Start with a signal $f(t)$ and its Fourier transform $\hat{f}(s)$, both functions of a continuous variable. We want to:
  - Find a discrete version of $f(t)$ that's a reasonable approximation of $f(t)$.
  - Find a discrete version of $\hat{f}(s)$ that's a reasonable approximation of $\hat{f}(s)$.

The Fourier Transform Pair: Continuous vs. Discrete

The continuous Fourier transform pair

$$\hat{f}(s) = \int_{-\infty}^{\infty} f(t)e^{-2\pi ist}dt$$

$$f(t) = \int_{-\infty}^{\infty} \hat{f}(s)e^{2\pi ist}ds$$

The discrete Fourier transform pair

$$\hat{f}_m = \sum_{n=0}^{N-1} f_n e^{-2\pi imn/N}$$

$$f_n = \frac{1}{N}\sum_{m=0}^{N-1} \hat{f}_m e^{2\pi imn/N}$$

where $f_n \equiv f(t_n)$ and $\hat{f}_m \equiv \hat{f}(s_m)$, $n,\ m = 0,\ 1,\ \ldots,\ N-1$

Sample Points and Discrete Fourier Transform (DFT)

- The function $f(t)$ is limited to $0 \leq t \leq L$ and we sample N evenly spaced samples at points t_n, while the function $\hat{f}(s)$ is limited to $0 \leq s \leq 2B$ and the DFT of $f(t_n)$ is $\hat{f}(s_m)$:

$f(t_n)$	$\hat{f}(s_m)$
$t_0 \longrightarrow f(t_0)$	$s_0 \longrightarrow \hat{f}(s_0)$
$t_1 \rightarrow f(t_1)$	$s_1 \rightarrow \hat{f}(s_1)$
$\dots$	$\dots$
$t_{N-1} \longrightarrow f(t_{N-1})$	$s_{N-1} \longrightarrow \hat{f}(s_{N-1})$

- where $t_0 = 0,\ t_1 = \frac{1}{2B},\ t_2 = \frac{2}{2B}, \dots, t_{N-1} = \frac{N-1}{2B};\ s_0 = 0,\ s_1 = \frac{1}{L}$, $s_2 = \frac{2}{L}, \dots, s_{N-1} = \frac{N-1}{L}$; the number of sample points is $N = \frac{L}{1/(2B)} = 2BL$ in the $t - domain$ and $N = \frac{2B}{1/L} = 2BL$ in the $s - domain$.

The DFT

- The discrete version of $f(t)$ is **the list of sampled values** $f(t_0), f(t_1), \ldots, f(t_{N-1})$:

$$f = [f(t_0), f(t_1), \ldots, f(t_{N-1})] \equiv [f_0, f_1, \ldots, f_{N-1}]$$

- The **DFT** of $f = [f(t_0), f(t_1), \ldots, f(t_{N-1})]$ is the $N - tuple$ $\hat{f} = [(\hat{f}(s_0), \hat{f}(s_1), \ldots, \hat{f}(s_{N-1})]$ defined by

$$\hat{f}(s_m) = \sum_{n=0}^{N-1} f(t_n) e^{-2\pi imn/N}, \quad \text{or} \quad \hat{f}_m = \sum_{n=0}^{N-1} f_n e^{-2\pi imn/N}$$

- The **inverse DFT** of $\hat{f}(s_m)$ is

$$f(t_n) = \frac{1}{N} \sum_{m=0}^{N-1} \hat{f}(s_m) e^{2\pi mn/N} \quad \text{or} \quad f_n = \frac{1}{N} \sum_{m=0}^{N-1} \hat{f}_m e^{2\pi imn/N}$$

A Heuristic Proof for the DFT

$$
\begin{aligned}
\mathcal{F}\{f\}(s_m) &= \int_0^L f(t)e^{-2\pi i s_m t}dt \\
&\approx \sum_{n=0}^{N-1} f(t_n)e^{-2\pi i(\frac{m}{L})t_n} \triangle t \\
&= \sum_{n=0}^{N-1} f(t_n)e^{-2\pi i(\frac{m}{L})(\frac{n}{2B})} \triangle t \\
&= \frac{1}{2B} \underbrace{\sum_{n=0}^{N-1} f(t_n)e^{-2\pi imn/N}} \equiv \frac{1}{2B} \underbrace{\hat{f}(s_m)}
\end{aligned}
$$

Thus,

$$
\hat{f}(s_m) = \sum_{n=0}^{N-1} f(t_n)e^{-2\pi imn/N}
$$

A Heuristic Proof for the Inverse DFT

$$
\begin{aligned}
f(t_n) &= \frac{1}{2B}\int_0^{2B} \hat{f}(s)e^{2\pi i s t_n}\,ds \\
&\approx \frac{1}{2B}\sum_{m=0}^{N-1} \hat{f}(s_m)e^{2\pi i(\frac{m}{L})t_n}\triangle s \\
&= \frac{1}{2B}\sum_{m=0}^{N-1} \hat{f}(s_m)e^{2\pi i(\frac{m}{L})(\frac{n}{2B})}\triangle s \\
&= \frac{1}{2BL}\sum_{m=1}^{N-1} \hat{f}(s_m)e^{2\pi i mn/N} \\
&= \frac{1}{N}\sum_{m=1}^{N-1} \hat{f}(s_m)e^{2\pi i mn/N}
\end{aligned}
$$

- We can implement fast Fourier transform (FFT) using Python's module *numpy.fft*.
- In Python, the DFT is defined as

$$A_m = \sum_{n=0}^{N-1} a_n \exp\left\{-2\pi i \frac{mn}{N}\right\}, \quad m = 0, 1, \ldots, N-1$$

  and the inverse DFT is defined as

$$a_n = \frac{1}{N} \sum_{m=0}^{N-1} A_m \exp\left\{2\pi i \frac{mn}{N}\right\}, \quad n = 0, 1, \ldots, n-1 \qquad (1)$$

- The fast Fourier transform algorithm developed by Cooley and Tukey (1965) and later extended by many others provide a more efficient algorithm for calculating DFT or inverse DFT **with sample points that are powers of two**. That is, $N = 2^j, \quad j \in \{1, 2, \ldots, \}$. Note: in Python, a different set of indexes are used.
- The Cooley-Tukey FFT algorithm can reduce the number of multiplications from N^2 to $N \log N$.

Computing European Options using Trapezoidal rule

- From Lecture 7, we obtain

$$V(k) = \frac{e^{-\alpha k}}{2\pi}\int_{-\infty}^{\infty} e^{i\omega k}\hat{\nu}(\omega)d\omega$$

or

$$V(k) = \text{Re}\left\{\frac{e^{-\alpha k}}{\pi}\int_{0}^{\infty} e^{i\omega k}\hat{\nu}_P(\omega)d\omega\right\}$$

- As demonstrated in Lecture 7, we can apply the Trapezoidal rule to compute $V(k_n)$

$$V(k_n) \approx \text{Re}\left\{\frac{e^{-\alpha k}}{\pi}\sum_{m=0}^{N} e^{i\omega_m k_n}\hat{\nu}(\omega_m)\triangle\omega_m\right\} \tag{2}$$

where

$$\triangle\omega_m \begin{cases} = \frac{h}{2}, & m = 0 \text{ or } N \\ = h, & m \neq 0 \text{ or } N \end{cases}$$

$$\hat{\nu}(\omega) = \frac{e^{-r(T-t_0)}\cdot\hat{q}(\omega+(\alpha+1)i)}{(\alpha-i\omega)(\alpha-i\omega+1)}$$

- To apply Python's FFT to (2), we need to rewrite (2) to fit in the form of (1). To do so, we do the following substitutions:
  - We drop the very last term in the summation series:

$$\frac{e^{-\alpha k}}{\pi}\sum_{m=0}^{N} e^{i\omega_m k_n}\hat{\nu}(\omega_m)\triangle\omega_m \approx \frac{e^{-\alpha k}}{\pi}\sum_{m=0}^{N-1} e^{i\omega_m k_n}\hat{\nu}(\omega_m)\triangle\omega_m$$

  - We rewrite $e^{i\omega_m k_n}$ as

$$e^{i\omega_m k_n} = e^{i(mh)(k_0+n\triangle k)} = e^{i(mh)(n\triangle k)}e^{i(mh)k_0}$$

  - We set $h\triangle k = \frac{2\pi}{N}$ so that

$$e^{i\omega_m k_n} = e^{2\pi i\left(\frac{mn}{N}\right)}e^{i\omega_m k_0}$$

  - Then (2) can be rewritten as

$$V(k_n) \approx \mathrm{Re}\left\{\frac{e^{-\alpha k_n}}{\pi}\sum_{m=0}^{N-1} e^{2\pi i\left(\frac{mn}{N}\right)}\cdot e^{i\omega_m k_0}\hat{\nu}(\omega_m)\triangle\omega_m\right\} \quad (3)$$

Adjusting Pricing Integral for Implementation of FFT (2)

- Define $A_m = e^{i\omega_m k_0}\hat{\nu}(\omega_m)\triangle\omega_m \cdot N$. Then

$$V(k_n) = \frac{e^{-\alpha k_n}}{\pi} \cdot \mathrm{Re}\left\{ \frac{1}{N} \sum_{m=0}^{N-1} A_m e^{2\pi i\left(\frac{mn}{N}\right)} \right\}$$

- That is,

$$a_n = \frac{1}{N} \sum_{m=0}^{N-1} A_m e^{2\pi i\left(\frac{mn}{N}\right)}$$

and

$$V(k_n) = \frac{e^{-\alpha k_n}}{\pi} \cdot \mathrm{Re}(a_n)$$

- Using Python,

$$a_n = np.fft.ifft(A_m)$$

- Choose $h = B/(N-1)$ and $N = 2^p$ with p being an even integer.
- Choose $\triangle k$ base on $h\triangle k = \frac{2\pi}{N}$.
- Set $\omega_m = mh,\ m = 0, \ldots N-1$; therefore, $\omega_{N-1} = (N-1)h = B$
- Set $k_n = k_0 + n\triangle k,\ n = 0, \ldots, N-1$; therefore, $k_{max} = k_0 + (N-1)\triangle k$
- Set $\alpha > 0$ for European calls and $\alpha < 0$ for European puts.

$$A = \begin{pmatrix} A_0 \\ A_1 \\ \vdots \\ A_{N-1} \end{pmatrix} = \begin{pmatrix} e^{i\omega_0 k_0}\hat{\nu}(\omega_0)\frac{h}{2} \cdot N \\ e^{i\omega_1 k_0}\hat{\nu}(\omega_1)h \cdot N \\ \vdots \\ e^{i\omega_{N-1} k_0}\hat{\nu}(\omega_{N-1})h \cdot N \end{pmatrix}$$

where

$$\hat{\nu}(\omega_m) = \frac{e^{-rT} \cdot \hat{q}(\omega_m + (\alpha+1)i)}{(\alpha - i\omega_m)(\alpha - i\omega_m + 1)}, \quad m = 0, \ldots, N-1$$

and $\hat{q}(\omega_m + (\alpha+1)i) = \overline{\varphi}_X(\omega_m + (\alpha+1)i)$, a complex conjugate of the characteristic function.

- *import numpy as np*
- $a = np.fft.ifft(A)$
- Calculate $V(k_n) = \frac{e^{-\alpha k_n}}{\pi} \cdot \mathrm{Re}(a_n)$:

$$V = \begin{pmatrix} V(k_0) \\ V(k_1) \\ \vdots \\ V(k_{N-1}) \end{pmatrix} = \begin{pmatrix} \frac{e^{-\alpha k_0}}{\pi}\mathrm{Re}(a_0) \\ \frac{e^{-\alpha k_1}}{\pi}\mathrm{Re}(a_1) \\ \vdots \\ \frac{e^{-\alpha k_{N-1}}}{\pi}\mathrm{Re}(a_{N-1}) \end{pmatrix}$$