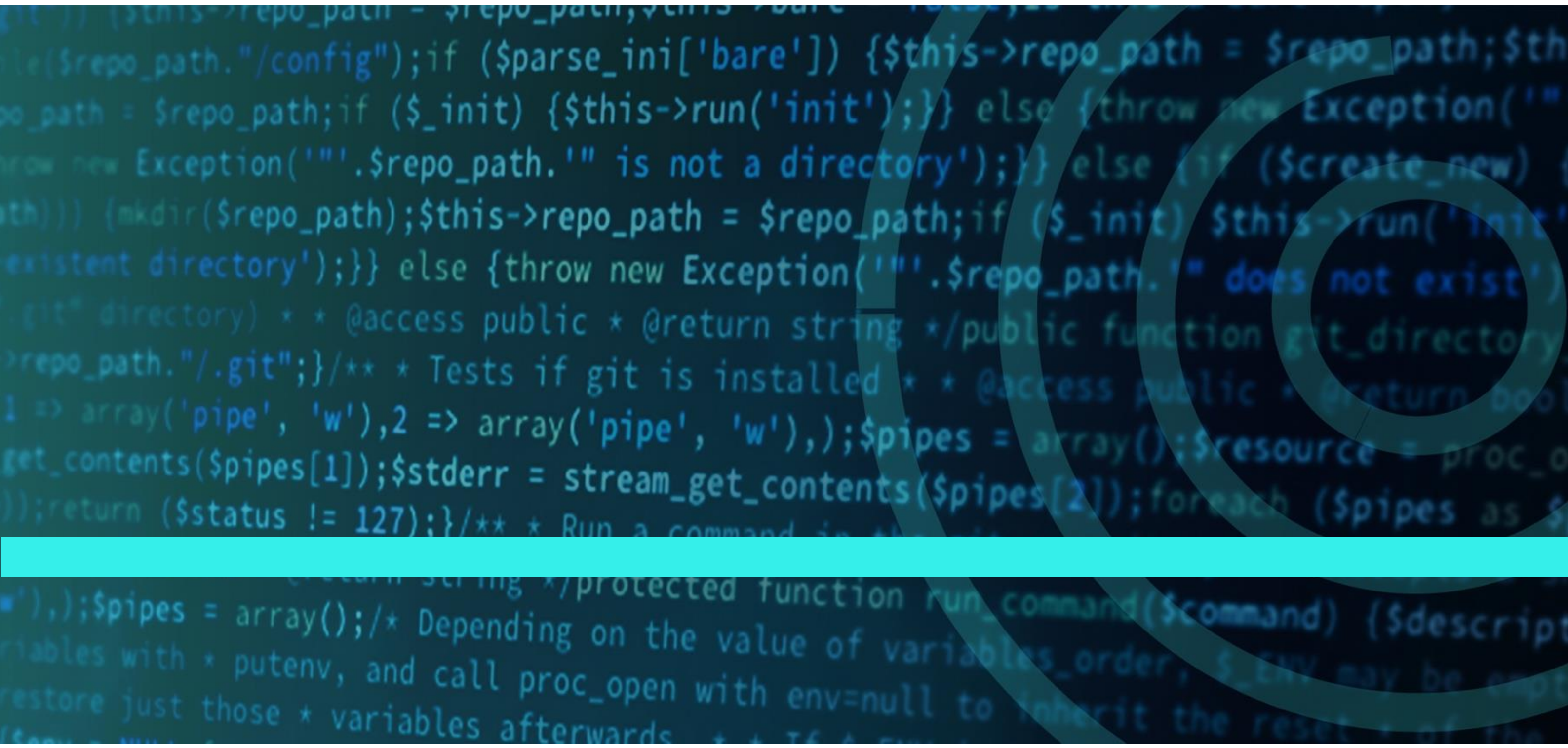
# Compiled Content Module 4

MScFE 630 Computational Finance



# Table of Contents

# Module 4: Fourier Methods for Option Pricing

This module introduces the Fourier Transform and explains how it can be used in pricing financial instruments. At the beginning of the module the Fourier Transform is defined and a number of characteristic Fourier Transform functions used in computational finance are illustrated. The module continues by showing how to implement these characteristics functions in Python to estimate pricing for different types of options. The module ends with the introduction of the Fast Fourier Transform and an explanation on how it is used for pricing financial instruments such as options.

# Unit 1: Fourier Transforms

## Unit 1: Video Transcript

In this module, we are going to be introducing Fourier methods, which are alternative ways of expressing functions (like the payoff of a financial derivative), which are computationally efficient. Our first step in this will be introducing Fourier transforms.

### Fourier transform

A Fourier transform is a manipulation of an integrable function, $f$, and is defined as follows:

$$\hat{f}(\omega) = \int_{-\infty}^{\infty} f(x)e^{i\omega x}\,dx, \tag{1}$$

where $\omega$ is any real number. A nice property of the Fourier transform is the way in which it can be reversed to get back to $f$:

$$f(x) = \frac{1}{2\pi}\int_{-\infty}^{\infty} \hat{f}(\omega)e^{-i\omega x}\,d\omega. \tag{2}$$

There are some conditions which need to hold for this to apply, but for the functions we'll be considering, this will always be the case.

You might notice that the expression given in equation (1) looks familiar to a characteristic function – this is the first use of our Fourier methods.

## Characteristic functions

The characteristic function of a random variable, $X$, is defined as follows:

$$\varphi_X(t) \coloneqq \mathbb{E}\left[e^{itX}\right]. \tag{3}$$

If $X$ has a probability density function, $f_X(x)$, then the Fourier transform of $f_X(x)$ is given by:

$$\widehat{f_X}(\omega) = \int_{-\infty}^{\infty} f_X(x) e^{i\omega x}\, dx = \mathbb{E}\left[e^{i\omega X}\right] = \varphi_X(\omega). \tag{4}$$

In other words, the Fourier transform of a probability density function is the characteristic function of the random variable. We can also use the Fourier inverse transform on our characteristic function to get back to our probability density function:

$$f_X(x) = \frac{1}{2\pi}\int_{-\infty}^{\infty} \varphi_X(\omega) e^{-i\omega x}\, d\omega. \tag{5}$$

It's worth noting at this point that characteristic functions are unique. This means that, if two random variables $X$ and $Y$ have the same characteristic function, they have the same distribution.

In order to use these characteristic functions for pricing, we need a theorem called Gil-Pelaez:

**Theorem 1** [Gil-Pelaez]

If $X$ is a random variable with characteristic function $\varphi_X(t)$, then:

$$\mathbb{P}[X \leq x] = \frac{1}{2} - \frac{1}{\pi}\int_0^{\infty} \frac{\operatorname{Im}\left[e^{-itx}\varphi_X(t)\right]}{t}\, dt,$$

where $\mathrm{Im}[z]$ is the imaginary part of a complex number, $z$.

So, we can calculate the CDF of any random variable given its characteristic function. Let's use this to price a call option under Black-Scholes assumptions. In this case, our call price, $c$, is given by:

$$c = \mathbb{E}^{\mathbb{Q}}[e^{-rT}(S_T - K)^+], \tag{6}$$

where $\mathbb{Q}$ is the risk-neutral measure, $r$ is the continuously-compounded risk-free rate, $T$ is the maturity time, $S_T$ is the value of the stock at time, and $K$ is the strike price.

We can manipulate this expression as follows:

$$\begin{aligned} c &= \mathbb{E}^{\mathbb{Q}}[e^{-rT}(S_T - K)^+] \\ &= e^{-rT}\mathbb{E}^{\mathbb{Q}}\left[S_T \mathbb{I}_{\{S_T>K\}} - K\mathbb{I}_{\{S_T>K\}}\right] \\ &= e^{-rT}\mathbb{E}^{\mathbb{Q}}\left[S_T \mathbb{I}_{\{S_T>K\}}\right] - e^{-rT}K\mathbb{E}^{\mathbb{Q}}\left[\mathbb{I}_{\{S_T>K\}}\right] \\ &= e^{-rT}\mathbb{E}^{\mathbb{Q}}\left[S_T \mathbb{I}_{\{S_T>K\}}\right] - e^{-rT}K\mathbb{Q}[S_T > K]. \end{aligned} \tag{7}$$

The expression $\mathbb{Q}[S_T > K]$ is of the form where we can use Gil-Pelaez, but $\mathbb{E}^{\mathbb{Q}}\left[S_T\, \mathbb{I}_{\{S_T>K\}}\right]$ isn't. To fix this, we implement a change of numeraire, from the risk-free bank account to the stock itself. The notes go into more detail for this, but it results in equation (7) becoming:

$$C = e^{-rT}\mathbb{E}^{\mathbb{Q}}\left[S_T \mathbb{I}_{\{S_T>K\}}\right] - e^{-rT}K\mathbb{Q}[S_T > K] \tag{8}$$

$$= S_0\mathbb{Q}^S[S_T > K] - e^{-rT}K\mathbb{Q}[S_T > K]$$

where $\mathbb{Q}^S$ is the equivalent martingale measure associated with the stock $S$. Again, the notes explain what this means, but it essentially results in $S_T$ having distribution:

$$\ln S_T \sim N\left(\ln S_0 + \left(r - \frac{\sigma^2}{2}\right)T, \sigma^2 T\right),$$

under the risk-neutral measure $\mathbb{Q}$ (where $\sigma$ is the stock's volatility), and:

$$\ln S_T \sim N\left(\ln S_0 + \left(r + \frac{\sigma^2}{2}\right)T, \sigma^2 T\right)$$

under the measure $\mathbb{Q}^S$. With these distributions, we can define characteristic functions of $\ln S_T$ as:

$$\varphi_{M_1}(t) = \exp\left(it\left(\ln S_0 + \left(r - \frac{\sigma^2}{2}\right)T\right) - \frac{1}{2}\sigma^2 T t^2\right)$$

(9)

$$\varphi_{M_2}(t) = \exp(it\sigma^2 T)\,\varphi_{M_1}(t),$$

where $\varphi_{M_1}(t)$ is the characteristic function of $\ln S_T$ under the risk-neutral measure, and $\varphi_{M_2}(t)$ is the characteristic function under the measure $\mathbb{Q}^S$. We can then use these with Gil-Pelaez to find the call price as follows:

$$c = S_0\mathbb{Q}^S[S_T > K] - e^{-rT}K\mathbb{Q}[S_T > K]$$

$$= S_0(1 - \mathbb{Q}^S[S_T \leq K]) - e^{-rT}K(1 - \mathbb{Q}[S_T \leq K])$$

$$= S_0(1 - \mathbb{Q}^S[\ln S_T \leq \ln K]) - e^{-rT}K(1 - \mathbb{Q}[\ln S_T \leq \ln K])$$

$$= S_0\left(\frac{1}{2} + \frac{1}{\pi}\int_0^\infty \frac{\mathrm{Im}\left[e^{-it\ln K}\varphi_{M_2}(t)\right]}{t}dt\right)$$

(10)

$$-e^{-rT}K\left(\frac{1}{2} + \frac{1}{\pi}\int_0^\infty \frac{\mathrm{Im}\left[e^{-it\ln K}\varphi_{M_1}(t)\right]}{t}dt\right),$$

where we've used Gil-Pelaez to go from line 3 to line 4.

It can be a bit tricky to understand all of this at first – it will become clearer when we implement it in Python in a later unit.

# Unit 1 : Notes

In this module, we will be introducing the concept of Fourier methods, and how they can be used to price options. Our ultimate goal will be to implement the fast Fourier transform, but we need to go through the basic theory before we can do that.

## Fourier transforms

Suppose $f(\cdot)$ is a function such that

$$\int_{\mathbb{R}^n} |f(x)| dx < \infty,$$

then the Fourier transform of $f(\cdot)$ is defined as follows:

$$\hat{f}(\omega) := \int_{\mathbb{R}^n} f(x) e^{i\omega \cdot x} \, dx, \tag{11}$$

where $\hat{f}(\omega)$ is the Fourier transform of $f(\cdot)$. This is for an $n$-dimensional $f(\cdot)$, so $\omega \in \mathbb{R}^n$. For our purposes, we will be limiting ourselves to $n = 1$ – the one-dimensional case. The Fourier transform can then be written as:

$$\hat{f}(\omega) = \int_{-\infty}^{\infty} f(x) e^{i\omega x} \, dx. \tag{12}$$

We can convert our Fourier transform, $\hat{f}(\omega)$ back to our original function, $f(x)$ using the Fourier inverse transform. In order to do this, we need the following conditions to hold:

- $\int_{-\infty}^{\infty} |\hat{f}(\omega)| d\omega < \infty$,
- $f(x)$ is continuous at $x$.

If these conditions are met, then

$$f(x) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \hat{f}(\omega) e^{-i\omega x}\, d\omega.$$

(13)

For most of our application of Fourier transforms, these requirements (called integrability and continuity) will always be true, and so we will usually not mention them.

If you are familiar with characteristic functions, the form of the integral in equation (12) will look similar to you. This is where we are going to be heading next.

## Characteristic functions

Suppose $X$ is a random variable which only takes on real values. Then the characteristic function of $X$ is defined as:

$$\varphi_X(t) := \mathbb{E}\left[e^{itX}\right].$$

(14)

Note that here we are treating $X$ as a one-dimensional (or univariate) random variable.

Let's further suppose that $X$ has a probability density function, and that it is given by $f_X(x)$. Then we know that $\int_{-\infty}^{\infty}|f_X(x)|dx = 1 < \infty$, and so we can perform a Fourier transform on $f_X(x)$:

$$\widehat{f_X}(\omega) = \int_{-\infty}^{\infty} f_X(x)e^{i\omega x}\, dx = \mathbb{E}\left[e^{i\omega X}\right] = \varphi_X(\omega)$$

(15)

In other words, if $X$ has a probability density function, its characteristic function is given by the Fourier transform of its density.

You can check for yourself, but if we have a probability density function, the requirements for the Fourier inverse transform are met, and so we can get back to our density function using our characteristic function:

$$f_X(x) = \frac{1}{2\pi}\int_{-\infty}^{\infty} \varphi_X(\omega)e^{-i\omega x}\, d\omega$$

(16)

A key property of the characteristic function is its uniqueness. This means that if two random variables, $X$ and $Y$, have the same characteristic function, then they have the exact same distribution. Contrast this to, say, the mean of variable – the mean doesn't tell you everything there is to know about a variable.

In order for us to use these characteristic functions for pricing, we need the following theorem:

**Theorem 1** [Gil-Pelaez]

If $X$ is a random variable with characteristic function $\varphi_X(t)$, then:

$$\mathbb{P}[X \leq x] = \frac{1}{2} - \frac{1}{\pi}\int_0^\infty \frac{\mathrm{Im}\left[e^{itx}\varphi_X(t)\right]}{t}dt,$$

where $\mathrm{Im}[z]$ is the imaginary part of a complex number, $z$.

Theorem (1) gives us a way of calculating probabilities using a characteristic function. We are now going to look at how this could be applied in a Black-Scholes world.

Let's suppose we would like to price a European call option on a stock, assuming a constant risk-free interest rate. Under risk-neutral pricing, the price of the call is given by:

$$c = \mathbb{E}^{\mathbb{Q}}[e^{-rT}(S_T - K)^+] \tag{17}$$

where $\mathbb{Q}$ is the risk-neutral measure, $r$ is the continuously-compounded risk-free rate, $T$ is the maturity time, $S_T$ is the value of the stock at time $T$, and $K$ is the strike price.

We can manipulate the expression in equation (17) as follows:

$$\begin{aligned} c &= \mathbb{E}^{\mathbb{Q}}[e^{-rT}(S_T - K)^+] \\ &= e^{-rT}\mathbb{E}^{\mathbb{Q}}\left[S_T\mathbb{I}_{\{S_T>K\}} - K\mathbb{I}_{\{S_T>K\}}\right] \\ &= e^{-rT}\mathbb{E}^{\mathbb{Q}}\left[S_T\mathbb{I}_{\{S_T>K\}}\right] - e^{-rT}K\mathbb{E}^{\mathbb{Q}}\left[\mathbb{I}_{\{S_T>K\}}\right] \\ &= e^{-rT}\mathbb{E}^{\mathbb{Q}}\left[S_T\mathbb{I}_{\{S_T>K\}}\right] - e^{-rT}K\mathbb{Q}[S_T > K] \end{aligned} \tag{18}$$

where $\mathbb{Q}(A)$ is the probability of $A$ under the measure $\mathbb{Q}$.

We want to rewrite the term $e^{-rT}\mathbb{E}^{\mathbb{Q}}\left[S_T\mathbb{I}_{\{S_T>K\}}\right]$ as a probability so that we can use Theorem (1). To do so, we need to use a change of numeraire. A numeraire is simply any stochastic process, $A_t$, such that $A_t > 0$ for all $t$. For each numeraire $A_t$, we can define an equivalent martingale measure (EMM), $\mathbb{Q}^A$. This is the measure under which:

$$E^{\mathbb{Q}^A}\left[\frac{X_T}{A_T}\middle|\mathcal{F}_t\right] = \frac{X_t}{A_t},$$

(19)

for any attainable claim $X$ i.e. claims 'discounted' by the numeraire are martingales.

With this definition, we have the following theorem:

**Theorem 2**

Suppose $A$ and $B$ are both numeraires with EMMs given by $\mathbb{Q}^A$ and $\mathbb{Q}^B$ respectively. Then:

$$A_0\mathbb{E}^{\mathbb{Q}^A}\left[\frac{X}{A_T}\right] = B_0\mathbb{E}^{\mathbb{Q}^B}\left[\frac{X}{B_T}\right]$$

for any random variable $X$.

This is done using a Radon-Nikodyn derivative equal to the ratio of the numeraires, in conjunction with Bayes' rule. Let's apply this theorem to our problem of rewriting $e^{-rT}\mathbb{E}^{\mathbb{Q}}\left[S_T\mathbb{I}_{\{S_T>K\}}\right]$. In this expression, the numeraire is the risk-free bank account, which has value $e^{rt}$ at time $t$. Note that $S_t$ is a valid numeraire (since we assume asset prices will never be negative). We can thus adjust our expression through theorem (2):

$$e^{r0}\mathbb{E}^{\mathbb{Q}}\left[\frac{S_T\mathbb{I}_{\{S_T>K\}}}{e^{rT}}\right] = S_0\mathbb{E}^{\mathbb{Q}^S}\left[\frac{S_T\mathbb{I}_{\{S_T>K\}}}{S_T}\right] \tag{20}$$

where $\mathbb{Q}^S$ is the EMM associated with the numeraire $S_t$.

Thus, equation (18) can be written as:

$$\begin{aligned} c &= e^{-rT}\mathbb{E}^{\mathbb{Q}}\left[S_T\mathbb{I}_{\{S_T>K\}}\right] - e^{-rT}K\mathbb{Q}[S_T > K] \\ &= S_0\mathbb{E}^{\mathbb{Q}^S}\left[\frac{S_T\mathbb{I}_{\{S_T>K\}}}{S_T}\right] - e^{-rT}K\mathbb{Q}[S_T > K] \\ &= S_0\mathbb{E}^{\mathbb{Q}^S}\left[\mathbb{I}_{\{S_T>K\}}\right] - e^{-rT}K\mathbb{Q}[S_T > K] \\ &= S_0\mathbb{Q}^S[S_T > K] - e^{-rT}K\mathbb{Q}[S_T > K] \end{aligned} \tag{21}$$

With equation (21), we can price our option by using theorem (1). The final step in doing so is finding the characteristic function for $S_T$ under the risk-neutral measure, $\mathbb{Q}$, and the stock numeraire EMM, $\mathbb{Q}^S$.

Let's now assume we are working in a Black-Scholes world. Then, we know that the distribution of our stock process under the risk-neutral measure, $\mathbb{Q}$, is log-normal i.e.

$$\ln S_T \sim N\left(\ln S_0 + \left(r - \frac{\sigma^2}{2}\right)T, \sigma^2 T\right),$$

where $\sigma$ is the stock's volatility. Let $M_1 = \ln S_T$ under the risk-neutral measure. Then:

$$\varphi_{M_1}(t) = \exp\left(it\left(\ln S_0 + \left(r - \frac{\sigma^2}{2}\right)T\right) - \frac{1}{2}\sigma^2 T t^2\right), \tag{22}$$

i.e. the characteristic function of a normal random variable.

If we now consider the stock numeraire EMM, $\mathbb{Q}^S$, the distribution of $S_T$ is still log-normal, but with different mean and variance:

$$\ln S_T \sim N\left(\ln S_0 + \left(r + \frac{\sigma^2}{2}\right)T, \sigma^2 T\right).$$

We will not go through how this is derived here, but it hinges on the fact that kernel for the Girsanov transformation from $\mathbb{Q}$ to $\mathbb{Q}^S$ is the volatility of $S$, $\sigma$.

If we now define $M_2 = \ln S_T$ under the EMM $\mathbb{Q}^S$, we have:

$$\varphi_{M_2}(t) = \exp\left(it\left(\ln S_0 + \left(r + \frac{\sigma^2}{2}\right)T\right) - \frac{1}{2}\sigma^2 T t^2\right) \tag{23}$$

$$= \exp(it\sigma^2 T)\,\varphi_{M_1}(t).$$

And so, solving equation (21) using theorem (1), we have:

$$c = S_0\mathbb{Q}^S[S_T > K] - e^{-rT}K\mathbb{Q}[S_T > K]$$

$$= S_0(1 - \mathbb{Q}^S[S_T \leq K]) - e^{-rT}K(1 - \mathbb{Q}[S_T \leq K])$$

$$= S_0(1 - \mathbb{Q}^S[\ln S_T \leq \ln K]) - e^{-rT}K(1 - \mathbb{Q}[\ln S_T \leq \ln K])$$

$$= S_0\left(\frac{1}{2} + \frac{1}{\pi}\int_0^\infty \frac{\operatorname{Im}\left[e^{-it\ln K}\varphi_{M_2}(t)\right]}{t}dt\right)$$

$$-e^{-rT}K\left(\frac{1}{2} + \frac{1}{\pi}\int_0^\infty \frac{\operatorname{Im}\left[e^{-it\ln K}\varphi_{M_1}(t)\right]}{t}dt\right) \tag{24}$$

Thus, we have an equation which can solve our pricing problem.

This set of notes has included a lot of detail, and much of the mathematics can be difficult to follow. It will be necessary to refer back to these notes as you go through the later units of this module.

.

# Unit 2: An Alternative Characteristic Function Pricing Method

## Unit 2 : Video Transcript

In today's video we are going to go through the Fourier-Cosine method and see how we can use it as an alternative method for using characteristic functions to price financial instruments. The goal with this method will be to split our pricing function into functions which only depend on the characteristic function, and the payoff of the instrument itself.

Firstly, we can define the Fourier-Cosine expansion for some function $f(x)$ over the interval $[0,\pi]$ as:

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n \cos(nx),$$

where $a_n$is defined as:

$$a_n = \frac{2}{\pi}\int_0^{\pi} f(x)\cos(nx)dx.$$

In other words, we can write a correctly defined function in terms of $cos$. The only problem is that we are currently forced to work over a very restrictive interval. We can generalize the expansion further by substituting in:

$$x = \pi\frac{y-b_1}{b_2-b_1}.$$

The Fourier-Cosine expansion then becomes:

$$f(y) = \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n \cos\left(n\pi\frac{y-b_1}{b_2-b_1}\right)$$

$$a_n = \frac{2}{b_2-b_1}\int_{b_1}^{b_2} f(y)\cos\left(n\pi\frac{y-b_1}{b_2-b_1}\right)dy.$$

It would be convenient to first evaluate two functions (in the context of the Fourier-Cosine expansion): $f(x) = e^x$and $f(x) = 1$. These can be shown to be equal:

$$\gamma_n(c,d) := \int_c^d e^s \cos\left(n\pi \frac{s-b_1}{b_2-b_1}\right) ds$$

$$= \frac{cos\left(n\pi\frac{d-b_1}{b_2-b_1}\right)e^d - cos\left(n\pi\frac{c-b_1}{b_2-b_1}\right)e^c + \frac{n\pi}{b_2-b_1}\left(sin\left(n\pi\frac{d-b_1}{b_2-b_1}\right)e^d - sin\left(n\pi\frac{c-b_1}{b_2-b_1}\right)e^c\right)}{1+\left(\frac{n\pi}{b_2-b_1}\right)^2}$$

and

$$\psi_n(c,d) := \int_c^d cos\left(n\pi\frac{s-b_1}{b_2-b_1}\right) ds$$

$$= \begin{cases} \frac{b_2-b_1}{n\pi}\left[sin\left(n\pi\frac{d-b_1}{b_2-b_1}\right) - sin\left(n\pi\frac{c-b_1}{b_2-b_1}\right)\right] & \text{when } n \neq 0 \\ d-c & \text{when } n = 0 \end{cases}$$

These are going to be important in a little bit, so we have assigned specific function names, $\gamma_n(c,d)$ and $\psi_n(c,d)$.

## Applying the Fourier-Cosine expansion to a vanilla European option

The next step is to actually start pricing some option. Using the Fourier-Cosine expansion to price options is known as the COS method. Let's define a few new variables that will help to make the mathematics that is coming a little bit more straight forward. Let $s_T = \ln\left(\frac{S_T}{K}\right)$. In other words, little $s$ is the scaled version of big $S$, which is the share price. Then, let $v(s)$ be the payoff of the option, and let $f_{s_T}(s)$ be the density function of $s_T$.

Let's try and price a call option. This means that we can write $p(s) = K(e^s - 1)\mathbb{I}_{\{s\geq 0\}}$ and, the price of a call option as:

$$c = e^{-rT} \int_{-\infty}^{\infty} p(st) f_{s_T}(s) ds$$

$$\approx e^{-rT} \int_{b_1}^{b_2} p(s) f_{s_T}(s) ds$$

(25)

Note that the line with the approximation is just a result of our limiting the upper and lower bounds of the integral. This is known as truncation, and we use it because, while infinity is a convenient mathematical concept, computers don't really know how to handle it.

Now, apply the Fourier-cosine expansion to $f_{s_T}(s)$ and, with some simplification, we get:

$$f_{s_T}(s) = \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n \cos\left(n\pi \frac{s - b_1}{b_2 - b_1}\right)$$

$$a_n \approx \frac{2}{b_2 - b_1} Re\left(\phi_{s_T}\left(\frac{n\pi}{b_2 - b_1}\right) e^{-in\pi\frac{b_1}{b_2-b_1}}\right)$$

(26)

Here, we just use $Re(x)$ to denote the real component of $x$, and $\phi_{s_T}(s)$ as the characteristic function of $s_T$.

Define:

$$v_n = \frac{2}{b_2 - b_1} \int_{b_1}^{b_2} p(s) \cos\left(n\pi \frac{s - b_1}{b_2 - b_1}\right) ds$$

$$= \frac{2}{b_2 - b_1} K\big(\gamma_n(0, b_2) - \psi_n(0, b_2)\big)$$

Then our final option price simplifies to:

$$c \approx e^{-rT}\left(\frac{v_0 \phi_{s_T}(0)}{2} + \sum_{n=1}^{N-1} Re\left(\phi_{s_T}\left(\frac{n\pi}{b_2 - b_1}\right) e^{-in\pi\frac{b_1}{b_2-b_1}}\right) v_n\right)$$

(27)

## Choosing an appropriate interval, $[\boldsymbol{b_1}, \boldsymbol{b_2}]$

You will probably have noticed that we used the interval $[b_1, b_2]$, without giving any idea as to how to find values for $b_2$ and $b_1$. One method for finding these values is to use cumulants. You don't really need to know what a cumulant is beyond the fact that they are like statistical moments but not quite the same. We can estimate reasonable values for our interval through the following equations:

$$b_1 = c_1 - L\sqrt{c_2 + \sqrt{c_4}}$$

$$b_2 = c_1 + L\sqrt{c_2 + \sqrt{c_4}}$$

Where $L = 10$, and $c_1$ refers to the first cumulant of $s_T$, and so on. Within the Black-Scholes framework, the cumulant values are:

$$c_1 = \mu T$$

$$c_2 = \sigma T$$

$$c_4 = 0$$

## Characteristic function of $\boldsymbol{s_T}$

It is important to note that the characteristic function of $s_T$ is:

$$\phi_{s_T}(u) = \exp\left(iu\left(log\left(\frac{S_0}{K} + \left(r - \frac{\sigma^2}{2}\right)T\right) - \frac{\sigma^2 T u^2}{2}\right)\right).$$

## Problems with and uses of the COS method

While the COS method is generally quite accurate, it would be important to note that there are three sources of error in every estimate. This means that every estimate we make will not be perfect. The first source of error comes from the truncation of the integral for the option price. The second is the extending of the interval of the integral for the density function, so that we can evaluate it using a characteristic function. The third source of error comes from the fact that, when we apply the Fourier-Cosine expansion of the call price, we only sum to capital $N$, and not to infinity.

The last thing that would be convenient to note about the COS method is that it is particularly useful for evaluating the Greeks. Thus, if you wanted to relatively easily calculate the sensitivity of an asset to various parameters.

# Unit 2: Notes

In the previous notes, we presented a set of theorems which allowed for us to price European calls and puts using their characteristic functions. In this unit, we will look at a different type of Fourier expansion to achieve the same thing.

A relevant question at this point is why we are presenting so many different methods to achieve, what appears to be, the same goal. The method presented in this unit, as will be discussed a little later on, is relatively easy to adapt to allow us to calculate the greeks. This is useful as it allows us to more easily gain insight into the sensitivities that the option has to different market components.

## The Fourier-Cosine series expansion

We now present a different type of Fourier expansion. The goal of applying this expansion is to produce a result which allows us to work with payoff of the instrument and the underlying process separately. This will make the evaluation of certain instruments significantly easier.

Suppose $f(x)$ is a function which is defined and integrable over the interval $[0, \pi]$. Then we can write the function as:

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n cos(nx), \tag{28}$$

where $a_n$ is defined as:

$$a_n = \frac{2}{\pi} \int_0^{\pi} f(x) cos(nx) dx.$$

The interval $[0, \pi]$ is a bit restrictive. So, we can transform the variable, $x$, so allow for some general bounds $[b_1, b_2]$. This is done by letting:

$$x = \pi \frac{y - b_1}{b_2 - b_1}. \tag{29}$$

We can change variable in (28) using (29) and we would have:

$$f(y) = \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n \cos\left(n\pi \frac{y - b_1}{b_2 - b_1}\right)$$

$$a_n = \frac{2}{b_2 - b_1} \int_{b_1}^{b_2} f(y) \cos\left(n\pi \frac{y - b_1}{b_2 - b_1}\right) dy. \tag{30}$$

Before moving on to pricing, we need to evaluate (in the context of the Fourier-Cosine expansion) two specific functions: $f(x) = e^x$ and $f(x) = 1$. Define:

$$\gamma_n(c, d) := \int_c^d e^s \cos\left(n\pi \frac{s - b_1}{b_2 - b_1}\right) ds \tag{31}$$

and

$$\psi_n(c, d) := \int_c^d \cos\left(n\pi \frac{s - b_1}{b_2 - b_1}\right) ds. \tag{32}$$

You can evaluate the integral in (31) analytically as:

$$\gamma_n(c,d) = \frac{cos\left(n\pi\frac{d-b_1}{b_2-b_1}\right)e^d - cos\left(n\pi\frac{c-b_1}{b_2-b_1}\right)e^c + \frac{n\pi}{b_2-b_1}\left(sin\left(n\pi\frac{d-b_1}{b_2-b_1}\right)e^d - sin\left(n\pi\frac{c-b_1}{b_2-b_1}\right)e^c\right)}{1+\left(\frac{n\pi}{b_2-b_1}\right)^2} \tag{33}$$

and you can evaluate the integral in (32) analytically as:

$$\psi_n(c,d) = \begin{cases} \frac{b_2-b_1}{n\pi}\left[sin\left(n\pi\frac{d-b_1}{b_2-b_1}\right) - sin\left(n\pi\frac{c-b_1}{b_2-b_1}\right)\right] & \text{when } n \neq 0 \\ d-c & \text{when } n = 0 \end{cases} \tag{34}$$

## Applying the Fourier-Cosine Series Expansion to a vanilla European option

Now that we have gone through all the necessary theorems, we can apply what we have learned. We will go through the method presented by Fang (2010).

Consider a vanilla European call or put option. The payoff of this type of option is:

$$[\alpha(S_T - K)]^+$$

Where $\alpha$ is $1$ for a call, and $-1$ for a put, and the rest of the notation is as usual. Now, we let $s_T = \ln\left(\frac{S_T}{K}\right)$. It will be useful to analyze functions using this transformation of $S_T$ (the reason why will become clear later on). Let $p(s)$ be the payoff of the option, and let $f_{s_T}(s)$ be the density function of $s_T$.

We can then write the price of a call option as:

$$c = e^{-rT}\int_{-\infty}^{\infty} p(s) f_{s_T}(s) ds$$

$$\approx e^{-rT}\int_{b_1}^{b_2} p(s) f_{s_T}(s) ds$$

(35)

Where the second line is simply a result of truncating the region that we integrate over. If we apply the Fourier-cosine expansion to $f_{s_T}(s)$, and with some simplification, we get:

$$f_{s_T}(s) = \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n \cos\left(n\pi \frac{s - b_1}{b_2 - b_1}\right)$$

$$a_n \approx \frac{2}{b_2 - b_1} Re\left(\phi_{s_T}\left(\frac{n\pi}{b_2 - b_1}\right) e^{-in\pi\frac{b_1}{b_2 - b_1}}\right)$$

(36)

Where $\mathrm{Re}(x)$ denotes the real component of $x$, and $\phi_{s_T}(s)$ is the characteristic function of $s_T$. In other words, we can approximate $f_{s_T}(s)$ as a function of its characteristic function. You should note that equation (30) is an approximation. This comes in because of how $a_n$ is defined in equation (28). $a_n$ is the integral from $b_1$ to $b_2$, whilst a characteristic function (in this case, $\phi_{s_T}$) is the integral from $-\infty$ to $\infty$. Thus, we are approximating $a_n$ by using the characteristic function for $s_T$.

The equation for a call price in (35) becomes:

$$c \approx e^{-rT}\left(\int_{b_1}^{b_2} p(s)\left[\frac{a_0}{2} + \sum_{n=1}^{\infty} a_n \cos\left(n\pi \frac{s - b_1}{b_2 - b_1}\right)\right]\right) ds.$$

(37)

If we define

$$v_n := \frac{2}{b_2 - b_1} \int_{b_1}^{b_2} p(s) cos\left(n\pi \frac{s-a}{b-a}\right) ds, \tag{38}$$

and substitute in our expression for $a_n$ in (36), and reduce the number of terms we sum over, our call price becomes:

$$c \approx e^{-rT}\left(\frac{v_0 \phi_{s_T}(0)}{2} + \sum_{n=1}^{N-1} \mathrm{Re}\left(\phi_{s_T}\left(\frac{n\pi}{b_2 - b_1}\right) e^{-in\pi\frac{b_1}{b_2-b_1}}\right) v_n\right) \tag{39}$$

The above formula is known as the COS formula. We've developed the above relatively generally. Now let's move forward assuming that we are dealing with a call option. Under the transformation, $s_T$, our payoff function becomes:

$$p(s) = K(e^s - 1)\mathbb{I}_{\{s \geq \mathbb{0}\}}$$

This means that $v_n$ becomes:

$$v_n = \frac{2}{b_2 - b_1} \int_0^{b_2} K(e^s - 1) cos\left(n\pi \frac{s - b_1}{b_2 - b_1}\right) ds$$

$$= \frac{2}{b_2 - b_1} K\big(\gamma_n(0, b_2) - \psi_n(0, b_2)\big).$$

For $n \in \mathbb{N}$. Note that the lower bound of the integral comes from the fact that we are only evaluating the integral over the region where $s \geq 0$.

## Choosing an appropriate interval, $[\boldsymbol{b_1}, \boldsymbol{b_2}]$

As will be noted later, by integrating over finite bounds instead of letting the integrals go to infinity, we are introducing an error into our estimate. As a result, selecting

appropriate values for $b_1$ and $b_2$ is important. Thankfully, Fang (2010) also proposed a method for finding these values. This method for selecting these values is to set:

$$b_1 = c_1 - L\sqrt{c_2 + \sqrt{c_4}}$$
$$b_2 = c_1 + L\sqrt{c_2 + \sqrt{c_4}},$$

where $L = 10$, and $c_i$ refers to the $i^{th}$ cumulant of $s_T$. A cumulant is similar to a statistical moment (but generally not exactly the same). In the case of the call above, we have:

$$c_1 = \mu T$$
$$c_2 = \sigma T$$
$$c_4 = 0$$

where $\mu$ and $\sigma$ is the mean and standard deviation of $s_T$ (which can be easily implied by the characteristic function of $s_T$). Note this only holds for Geometric Brownian motion. You will need to re-derive each term if you use different dynamics for your underlying asset.

## Characteristic function of $\boldsymbol{s_T}$

It can be shown that the characteristic function of $s_T$ is:

$$\phi_{s_T}(u) = \exp\left(iu\left(log\left(\frac{S_0}{K} + \left(r - \frac{\sigma^2}{2}\right)T\right) - \frac{\sigma^2 T u^2}{2}\right)\right).$$

## Problems with applying Fourier-Cosine expansion for pricing

When applying the Fourier-Cosine expansion, we are forced to approximate several terms, which includes a degree of error into our final estimates. Our estimates still have the potential to be very accurate despite these potential sources of error. It's good to be aware of problems with approximation, however.

The first source of error comes from the fact that we reduced the interval that we evaluated the price of the option over, in equation (35). Thus, we aren't actually integrating over all possible values of $s$.

The second source of error is from equation (36). We had to make an approximation in order to be able to write the co-efficient, $a_n$, in terms of the characteristic function of $q_{s_T}(s)$. $a_n$ is defined as being an integral over a finite interval, while the characteristic function of $q_{s_T}(s)$ is an integral over the interval $(-\infty, \infty)$. Thus, we approximated the characteristic function using a finite interval for our integral.

The third source of error comes from equation (39). We are not summing over all possible values of $n$, and restrict ourselves to $N - 1$ discrete values. This is a similar concept to restricting the bounds of integration.

## Uses of the COS method

Because of the way that the final formula is structured, the COS formula is very well suited towards evaluating the greeks. It is important to be aware that this method is useful for this application.

.

# Unit 3: Fourier Transform and Cosine Fourier

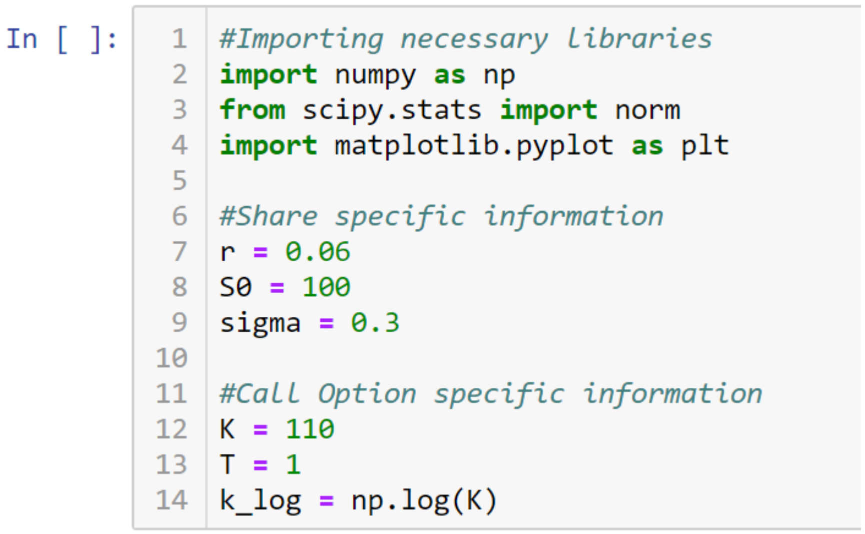
## Unit 3: Video Transcript

In this video, we will be going through how to implement the techniques introduced in the first two sets of notes. We will be finding the price of a call option with the following conditions:

- Continuously-compounded interest rate, $r$, of 6%
- Initial stock price, $S_0$, of \$100
- Stock volatility, $\sigma$, 30%
- Strike price, $K$, of \$110
- Maturity time, $T$, of one year

As per usual, we make all the assumptions of the Black-Scholes model.

Our first step is to create variables for the above:



```
In [ ]:
#Importing necessary libraries
import numpy as np
from scipy.stats import norm
import matplotlib.pyplot as plt

#Share specific information
r = 0.06
S0 = 100
sigma = 0.3

#Call Option specific information
K = 110
T = 1
k_log = np.log(K)
```

We can calculate the analytical price of this option using our closed-form solution.

```
In [ ]:
# Code for analytical solution for vanilla European Call option
d_1_stock = (np.log(S0/K)+(r + sigma**2/2)*(T))/(sigma*np.sqrt(T))
d_2_stock = d_1_stock - sigma*np.sqrt(T)

analytic_callprice = S0*norm.cdf(d_1_stock)-K*np.exp(-r*(T))*norm.cdf(d_2_stock)
```

We will now go through the code for each technique in the order of the notes.

## Fourier transforms and characteristic functions

Remember that, using our characteristic function, the formula for our call price is given by:

$$c = S_0\left(\frac{1}{2}+\frac{1}{\pi}\int_0^\infty \frac{Im\left[e^{-it\ln K}\varphi_{M_2}(t)\right]}{t}dt\right)$$

$$-e^{-rT}K\left(\frac{1}{2}+\frac{1}{\pi}\int_0^\infty \frac{Im\left[e^{-it\ln K}\varphi_{M_1}(t)\right]}{t}dt\right), \tag{40}$$

where

$$\varphi_{M_1}(t) = \exp\left(it\left(\ln S_0 + \left(r-\frac{\sigma^2}{2}\right)T\right)-\frac{1}{2}\sigma^2 T t^2\right) \tag{41}$$

$$\varphi_{M_2} = \exp(it\sigma^2 T)\,\varphi_{M_1}(t).$$

We first write the code for the characteristic functions given in equation (41).

```
In [ ]:
#Characteristic functions
def c_M1(t):
    return np.exp(1j*t*(np.log(S0)+(r-sigma**2/2)*T) - (sigma**2)*T*(t**2)/2)

def c_M2(t):
    return np.exp(1j*t*sigma**2*T)*c_M1(t)
```

You can see here that we use $1j$ to create an imaginary $i$.

We can't calculate the integrals in equation (40) analytically, and so we are going to use a midpoint Riemann sum.

$$\begin{aligned}&\int_0^\infty \frac{Im\left[e^{-it\ln K}\varphi_M(t)\right]}{t}dt \\ &\approx \int_0^{t_{max}} \frac{Im\left[e^{-it\ln K}\varphi_M(t)\right]}{t}dt \\ &\approx \sum_{n=1}^{N} \frac{Im\left[e^{-it_n\ln K}\varphi_M(t_n)\right]}{t_n}\Delta t\end{aligned} \tag{42}$$

So, we are only integrating up to $t_{max}$, and are using capital $N$ rectangles to approximate the area under the graph. $\Delta t$ is equal to $\frac{t_{max}}{N}$nd, since we are using a midpoint formula, $t_n = \left(n-\frac{1}{2}\right)\Delta t$. We can sub either of our characteristic functions, $\varphi_{M_1}$ and $\varphi_{M_2}$ into $\varphi_M$.

We choose to use $t_{max} = 20$ and capital $N = 100$, and can then implement this equation in Python.

```
In [ ]:
#Choosing t_max and N
t_max = 20
N = 100

#Calculating delta and constructing t_n
delta_t = t_max/N
from_1_to_N = np.linspace(1,N,N)
t_n = (from_1_to_N-1/2)*delta_t

#Approximate integral estimates
first_integral = sum((((np.exp(-1j*t_n*k_log)*c_M2(t_n)).imag)/t_n)*delta_t)
second_integral = sum((((np.exp(-1j*t_n*k_log)*c_M1(t_n)).imag)/t_n)*delta_t)
```

Finally, we can use these integrals in our original call price formula:

```
In [ ]: fourier_call_val = S0*(1/2 + first_integral/np.pi)-np.exp(-r*T)*K*(1/2 + second_integral/np.pi)
```

The closed-form solution is 10.4241004587, and our estimate is 10.4241004430 – a very good approximation.

## Fourier-Cosine series expansion

We are keeping all of our call option properties the same (risk-free rate, strike, etc.). We first need to create functions for:

$$\upsilon_n(c,d) = \frac{cos\left(n\pi\frac{d-b_1}{b_2-b_1}\right)e^d - cos\left(n\pi\frac{c-b_1}{b_2-b_1}\right)e^c + \frac{n\pi}{b_2-b_1}\left(sin\left(n\pi\frac{d-b_1}{b_2-b_1}\right)e^d - sin\left(n\pi\frac{c-b_1}{b_2-b_1}\right)e^c\right)}{1+\left(\frac{n\pi}{b_2-b_1}\right)^2} \tag{43}$$

$$\psi_n(c,d) = \begin{cases} \frac{b_2-b_1}{n\pi}\left[sin\left(n\pi\frac{d-b_1}{b_2-b_1}\right) - sin\left(n\pi\frac{c-b_1}{b_2-b_1}\right)\right] & \text{when } n \neq 0 \\ d-c & \text{when } n = 0 \end{cases} \tag{44}$$

Which we do in Python:

```
In[ ]:
#General functions for valuations
def upsilon_n(b2,b1,d,c,n):
    npi_d = np.pi*n*(d-b1)/(b2-b1)
    npi_c = np.pi*n*(c-b1)/(b2-b1)
    val_one = (np.cos(npi_d)*np.exp(d)-np.cos(npi_c)*np.exp(c))
    val_two = (n*np.pi*(np.sin(npi_d)*np.exp(d)-np.sin(npi_c)*np.exp(c))/(b2-b1))
    return (val_one + val_two)/(1+(n*np.pi/(b2-b1))**2)

def psi_n(b2,b1,d,c,n):
    if n == 0:
        return d-c
    else:
        return (b2-b1)*(np.sin(n*np.pi*(d-b1)/(b2-b1))-np.sin(n*np.pi*(c-b1)/(b2-b1)))/(n*np.pi)
```

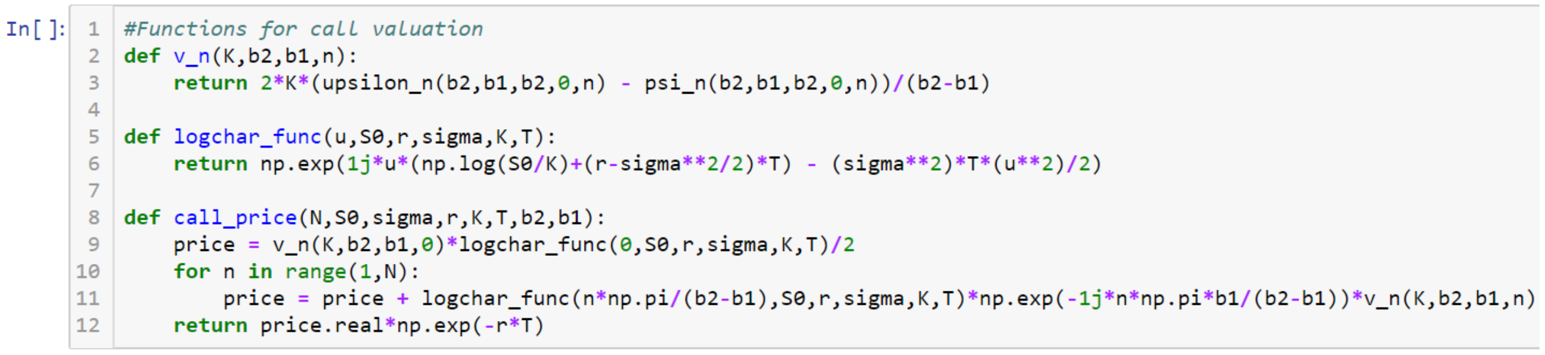
We then use the transformation detailed in the notes, little $s_T = \ln\frac{S_T}{K}$. Then, by setting $\phi_{s_T}(\cdot)$ to be the characteristic function of $s_T$ and

$$v_n = \frac{2}{b_2 - b_1} K\big(\upsilon_n(0, b_2) - \psi_n(0, b_2)\big), \tag{45}$$

our call price is given by:

$$c \approx e^{-rT}\left(\frac{v0\phi_{s_T}(0)}{2} + \sum_{n=1}^{N-1} Re\left(\phi_{s_T}\left(\frac{n\pi}{b_2 - b_1}\right)e^{-in\pi\frac{b_1}{b_2-b_1}}\right)v_n\right). \tag{46}$$

We create three functions in Python which do these calculations. Note that they require a $b_1$ and $b_2$ as arguments – we will define these now.


```
In[ ]:
#Functions for call valuation
def v_n(K,b2,b1,n):
    return 2*K*(upsilon_n(b2,b1,b2,0,n) - psi_n(b2,b1,b2,0,n))/(b2-b1)

def logchar_func(u,S0,r,sigma,K,T):
    return np.exp(1j*u*(np.log(S0/K)+(r-sigma**2/2)*T) - (sigma**2)*T*(u**2)/2)

def call_price(N,S0,sigma,r,K,T,b2,b1):
    price = v_n(K,b2,b1,0)*logchar_func(0,S0,r,sigma,K,T)/2
    for n in range(1,N):
        price = price + logchar_func(n*np.pi/(b2-b1),S0,r,sigma,K,T)*np.exp(-1j*n*np.pi*b1/(b2-b1))*v_n(K,b2,b1,n)
    return price.real*np.exp(-r*T)
```

By setting:

$$L = 10$$
$$c_1 = r$$
$$c_2 = \sigma^2 T$$
$$c_4 = 0.$$

We can define:

$$b_1 = c_1 - L\sqrt{c_2 + \sqrt{c_4}}$$

$$b_2 = c_1 + L\sqrt{c_2 + \sqrt{c_4}}$$

```
In [ ]:
#b1, b2 for call
c1 = r
c2 = T*sigma**2
c4 = 0
L = 10

b1 = c1-L*np.sqrt(c2-np.sqrt(c4))
b2 = c1+L*np.sqrt(c2-np.sqrt(c4))
```

And now we find our price for the call. We're trying different values for capital $N$ from $1$ to $50$.

```
In [ ]:
#Calculating COS for various N
COS_callprice = [None]*50

for i in range(1,51):
    COS_callprice[i-1] = call_price(i,S0,sigma,r,K,T,b2,b1)
```

Lastly, we can plot the estimate versus the analytical call price for different values of $N$:

```
In [ ]:
#Plotting the results
plt.plot(COS_callprice)
plt.plot([analytic_callprice]*50)
plt.xlabel("N")
plt.ylabel("Call price")
```

which should produce figure (1), below.

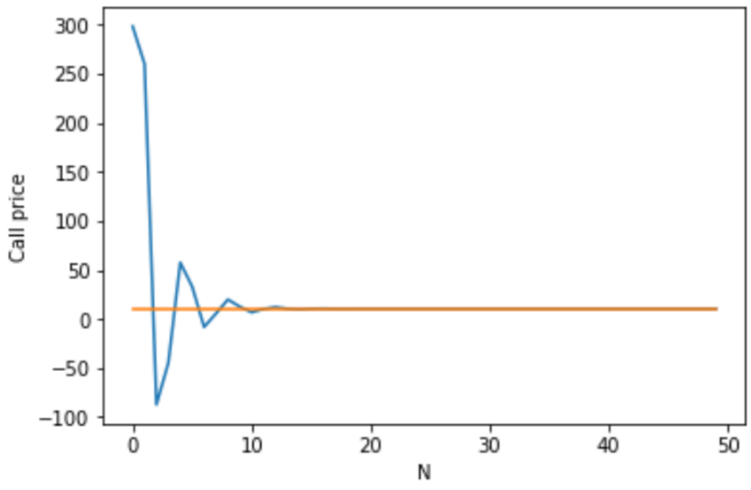


**Figure 1: COS call estimates for different $N$**

You can see that our estimate gets more accurate very quickly as we increase capital $N$. The notes show an additional plot of how the error changes over time. Make sure you are able to code these techniques up yourself, as they will be useful later on.

## Unit 3 : Notes

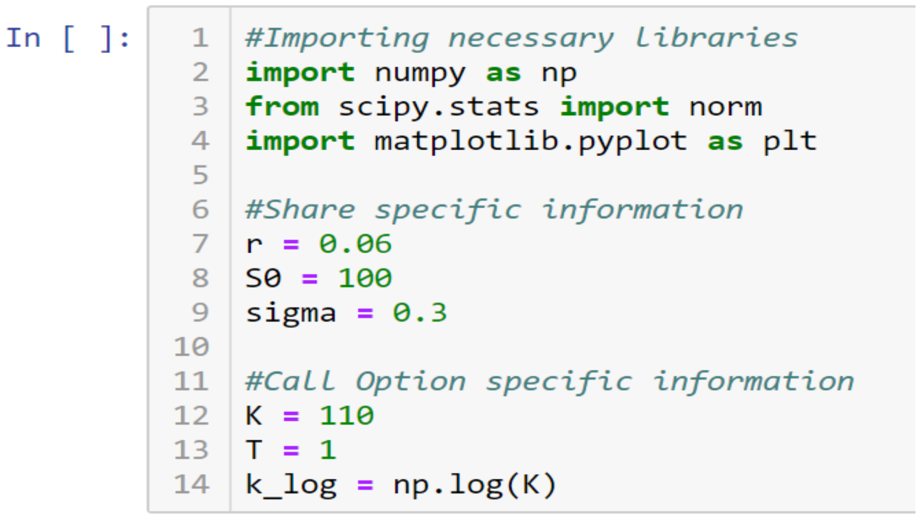
In this module, we have so far introduced two techniques for pricing vanilla options using Fourier transform techniques. In this unit, we will be going through the Python code which implements these techniques. We will be using Fourier transforms and the Fourier-Cosine expansion. It will be useful to have the two sets of notes on hand, and to make sure you can understand how the formulas are being translated into code.

We will be pricing a vanilla European call option on a single stock under the following conditions:

- Continuously-compounded interest rate, $r$, of 6%
- Initial stock price, $S_0$, of \$100
- Stock volatility, $\sigma$, 30%
- Strike price, $K$, of \$110
- Maturity time, $T$, of one year

As per usual, we make all the assumptions of the Black-Scholes model.

Our first step in coding up these Fourier techniques is to create variables for all of the information above – this information is the same for each technique, and we will be keeping variable names constant throughout.



```
In [ ]:
#Importing necessary libraries
import numpy as np
from scipy.stats import norm
import matplotlib.pyplot as plt

#Share specific information
r = 0.06
S0 = 100
sigma = 0.3

#Call Option specific information
K = 110
T = 1
k_log = np.log(K)
```

We can calculate the analytical price of this option using our closed-form solution.

```
In [ ]:
# Code for analytical solution for vanilla European Call option
d_1_stock = (np.log(S0/K)+(r + sigma**2/2)*(T))/(sigma*np.sqrt(T))
d_2_stock = d_1_stock - sigma*np.sqrt(T)

analytic_callprice = S0*norm.cdf(d_1_stock)-K*np.exp(-r*(T))*norm.cdf(d_2_stock)
```

We will now go through the code for each technique in the order of the notes.

## Fourier transforms and characteristic functions

Our formula for our call price is given by:

$$c = S_0\left(\frac{1}{2}+\frac{1}{\pi}\int_0^\infty \frac{Im\left[e^{-it\ln K}\varphi_{M_2}(t)\right]}{t}dt\right)$$
$$-e^{-rT}K\left(\frac{1}{2}+\frac{1}{\pi}\int_0^\infty \frac{Im\left[e^{-it\ln K}\varphi_{M_1}(t)\right]}{t}dt\right), \tag{47}$$

where

$$\varphi_{M_1}(t) = \exp\left(it\left(\ln S_0+\left(r-\frac{\sigma^2}{2}\right)T\right)-\frac{1}{2}\sigma^2Tt^2\right) \tag{48}$$

$$\varphi_{M_2} = \exp(it\sigma^2T)\,\varphi_{M_1}(t).$$

We first write the code for the characteristic functions given in equation (48).

```
In [ ]:
#Characteristic functions
def c_M1(t):
    return np.exp(1j*t*(np.log(S0)+(r-sigma**2/2)*T) - (sigma**2)*T*(t**2)/2)

def c_M2(t):
    return np.exp(1j*t*sigma**2*T)*c_M1(t)
```

Note that we use $1j$ to create an imaginary $i$.

Next, we can go about solving the integral in equation (47). Note that we could do this directly using the scipy.integrate module, but we are going to show a technique for approximating the value of the integral. We will be doing so using areas of rectangles.

$$\int_0^\infty \frac{Im\left[e^{-it\ln K}\varphi_M(t)\right]}{t} d$$

$$\approx \int_0^{t_{max}} \frac{Im\left[e^{-it\ln K}\varphi_M(t)\right]}{t} dt \tag{49}$$

$$\approx \sum_{n=1}^{N} \frac{Im\left[e^{-it_n\ln K}\varphi_M(t_n)\right]}{t_n} \Delta t$$

where $t_{max}$ some sufficiently large number, $N$ is an integer, $\Delta t = \frac{t_{max}}{N}$ and $t_n = \left(n - \frac{1}{2}\right)\Delta t$. So, we are splitting the region we are integrating over into $N$ parts, and then calculating the sum of the areas of the rectangles made from the function in the integral. Equation (49) is in terms of $\varphi_M(t)$, where $M = M_1$ or $M_2$.

To do this in Python, we first choose $t_{max}$ and $N$, and then calculate $\Delta t$ and create an array for $t_n$:

In [ ]:
```python
#Choosing t_max and N
t_max = 20
N = 100

#Calculating delta and constructing t_n
delta_t = t_max/N
from_1_to_N = np.linspace(1,N,N)
t_n = (from_1_to_N-1/2)*delta_t
```

We've chosen $t_{max} = 20$ and $N = 100$ – these choices are sufficiently large for approximating our integral. In practice, you would probably need to try a few different combinations to get good approximations, but this method converges fairly quickly in general. We can then calculate the sum as given in equation (49):

In [ ]:
```python
#Approximate integral estimates
first_integral = sum((((np.exp(-1j*t_n*k_log)*c_M2(t_n)).imag)/t_n)*delta_t)
second_integral = sum((((np.exp(-1j*t_n*k_log)*c_M1(t_n)).imag)/t_n)*delta_t)
```

The variable `first_integral` equates to the first integral in equation (47), while `second_integral` equates to the second integral in the same equation. Note that, in order to find the imaginary part of a complex number in Python, we use .imag at the end of the number.

We can now sub our integral estimates into the formula in equation (47) to find our price:

In [ ]:
```python
fourier_call_val = S0*(1/2 + first_integral/np.pi)-np.exp(-r*T)*K*(1/2 + second_integral/np.pi)
```

The closed-form solution for the call price is 10.4241004587, while the Fourier transform gives a solution of 10.4241004430 – we see we get a high level of accuracy with only $N$ (100) calculations.

## Fourier-Cosine series expansion

We keep all of our option, stock, and market parameters the same, and do not import any new libraries. We then need to create the following formulas in Python:

$$\upsilon_n(c,d) = \frac{cos\left(n\pi\frac{d-b_1}{b_2-b_1}\right)e^d - cos\left(n\pi\frac{c-b_1}{b_2-b_1}\right)e^c + \frac{n\pi}{b_2-b_1}\left(sin\left(n\pi\frac{d-b_1}{b_2-b_1}\right)e^d - sin\left(n\pi\frac{c-b_1}{b_2-b_1}\right)e^c\right)}{1+\left(\frac{n\pi}{b_2-b_1}\right)^2} \tag{50}$$

$$\psi_n(c,d) = \begin{cases} \frac{b_2-b_1}{n\pi}\left[sin\left(n\pi\frac{d-b_1}{b_2-b_1}\right) - sin\left(n\pi\frac{c-b_1}{b_2-b_1}\right)\right] & \text{when } n \neq 0 \\ d-c & \text{when } n = 0 \end{cases} \tag{51}$$

We do so by defining a function for each:

```
In [ ]:
#General functions for valuations
def upsilon_n(b2,b1,d,c,n):
    npi_d = np.pi*n*(d-b1)/(b2-b1)
    npi_c = np.pi*n*(c-b1)/(b2-b1)
    val_one = (np.cos(npi_d)*np.exp(d)-np.cos(npi_c)*np.exp(c))
    val_two = (n*np.pi*(np.sin(npi_d)*np.exp(d)-np.sin(npi_c)*np.exp(c))/(b2-b1))
    return (val_one + val_two)/(1+(n*np.pi/(b2-b1))**2)

def psi_n(b2,b1,d,c,n):
    if n == 0:
        return d-c
    else:
        return (b2-b1)*(np.sin(n*np.pi*(d-b1)/(b2-b1))-np.sin(n*np.pi*(c-b1)/(b2-b1)))/(n*np.pi)
```

$\upsilon_n(c,d)$ is calculated by the function `upsilon_n`, with the appropriate $b_1$ and $b_2$ given as input arguments. $\psi_n(c,d)$ is calculated by $\psi_n$, again with the appropriate $b_1$ and $b_2$. These are two general Fourier-Cosine expansions, which one might use in a variety of different problems.

Secondly, we need functions that are particular to our problem of pricing the call. Under the transformation $s_T = \ln\left(\frac{S_T}{K}\right)$, the call price is given by:

$$c \approx e^{-rT}\left(\frac{v_0 \phi_{s_T}(0)}{2} + \sum_{n=1}^{N-1} Re\left(\phi_{s_T}\left(\frac{n\pi}{b_2 - b_1}\right) e^{-in\pi\frac{b_1}{b_2-b_1}}\right) v_n\right) \tag{52}$$

where $\phi_{s_T}(\cdot)$ is the characteristic function of $s_T$, and

$$v_n = \frac{2}{b_2 - b_1} K\big(\upsilon_n(0, b_2) - \psi_n(0, b_2)\big). \tag{53}$$

In Python:

In [ ]:
```
#Functions for call valuation
def v_n(K,b2,b1,n):
    return 2*K*(upsilon_n(b2,b1,b2,0,n) - psi_n(b2,b1,b2,0,n))/(b2-b1)

def logchar_func(u,S0,r,sigma,K,T):
    return np.exp(1j*u*(np.log(S0/K)+(r-sigma**2/2)*T) - (sigma**2)*T*(u**2)/2)

def call_price(N,S0,sigma,r,K,T,b2,b1):
    price = v_n(K,b2,b1,0)*logchar_func(0,S0,r,sigma,K,T)/2
    for n in range(1,N):
        price = price + logchar_func(n*np.pi/(b2-b1),S0,r,sigma,K,T)*np.exp(-1j*n*np.pi
        *b1/(b2-b1))*v_n(K,b2,b1,n)
    return price.real*np.exp(-r*T)
```

The characteristic function, $\phi_{s_T}(\cdot)$, is calculated in `logchar_func` (remember that this is the characteristic function of $\ln\frac{S_T}{K}$); $v_n$ is calculated by the function `v_n`; and finally the call price (as per equation (52) is found by the function `call_price`. Note that you can choose how many discrete $N$'s are used – the larger this number, the more accurate the estimate will be.

The final step before we can actually calculate our call price is to choose $b_1$ and $b_2$. We are going to use the values given by:

$$b_1 = c_1 - L\sqrt{c_2 + \sqrt{c_4}}$$

$$b_2 = c_1 + L\sqrt{c_2 + \sqrt{c_4}}$$

with

$$L = 10$$
$$c_1 = \mu T$$
$$c_2 = \sigma_{s_T} T$$
$$c_4 = 0,$$

where $c_1 = \ln\frac{S_T}{K} + \left(r - \frac{\sigma^2}{2}\right)T$ and $c_2 = \sigma^2 T$ for $s_T$ in this case.

```
In [ ]:
#b1, b2 for call
c1 = r
c2 = T*sigma**2
c4 = 0
L = 10

b1 = c1-L*np.sqrt(c2-np.sqrt(c4))
b2 = c1+L*np.sqrt(c2-np.sqrt(c4))
```

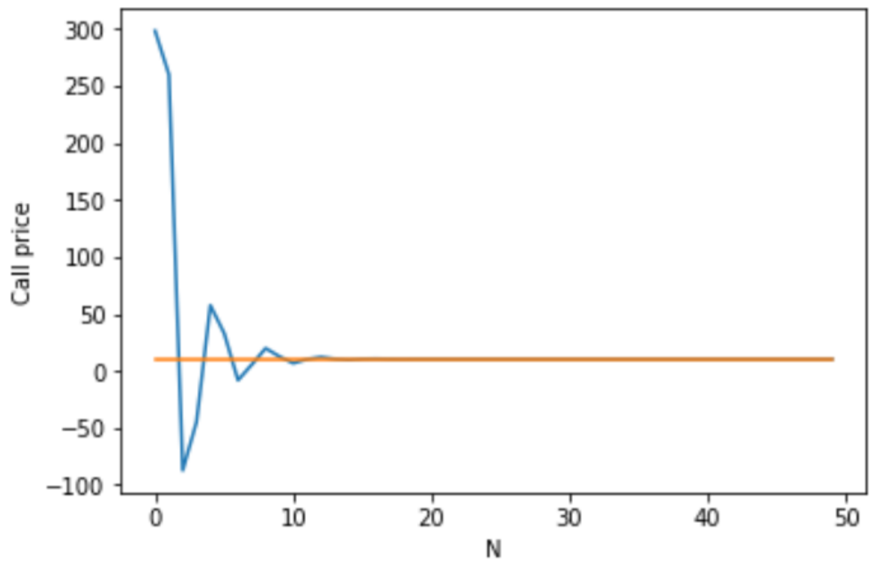


**Figure 2: COS call estimates for different *N***

Finally, we implement the COS pricing method for various values of $N$ – from $N = 1$ to $N = 50$:

```
In [ ]:
#Calculating COS for various N
COS_callprice = [None]*50

for i in range(1,51):
    COS_callprice[i-1] = call_price(i,S0,sigma,r,K,T,b2,b1)
```

The `COS_callprice` array captures the price of our call for the different values of $N$. In order to show how increasing $N$ improves the estimate, we plot the value of COS estimates versus the analytical price:

```
In [ ]:
#Plotting the results
plt.plot(COS_callprice)
plt.plot([analytic_callprice]*50)
plt.xlabel("N")
plt.ylabel("Call price")
```

Figure (2) shows the estimates versus $N$. As you can see, there is a fast convergence, and even $N$ as small as 20 gives a reasonable estimate. We can plot the figure to show how the COS estimate converges as follows:

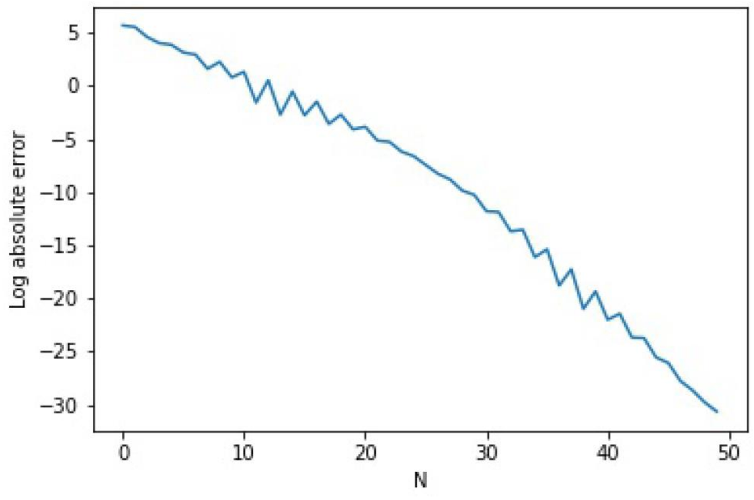
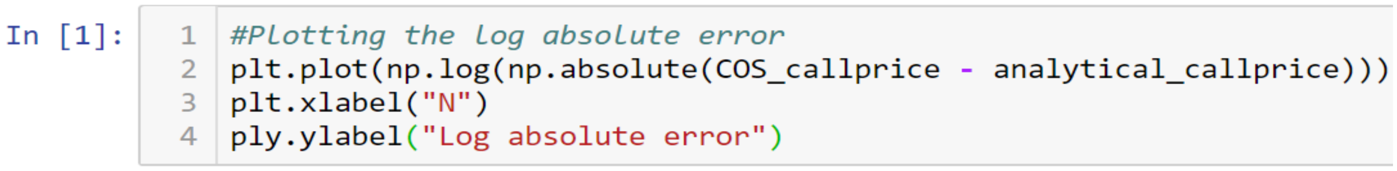


**Figure 3: COS call estimates for different $N$**



```
In [1]:
#Plotting the log absolute error
plt.plot(np.log(np.absolute(COS_callprice - analytical_callprice)))
plt.xlabel("N")
ply.ylabel("Log absolute error")
```

We plot the log of the absolute error between the analytical solution and our COS estimates. This will give the plot shown in figure (3). As you can see, the log error grows small quickly – demonstrating the fast convergence the COS estimates exhibit.

In the final unit of this module, we will introduce Fast Fourier Transforms, and show Python code of how they can be implemented.

# Unit 4 : The Fast Fourier Transform

## Unit 4 : Video Transcript

In this video we will go through the mathematics underlying the Fast Fourier Transform (FFT). This is an algorithm that allows us to quickly evaluate the Discrete Fourier Transform (DFT) so that we can quickly calculate the value of certain financial instruments.

### The mathematics behind the FFT

The DFT transforms a sequence of numbers of length capital $N$, $x_0, x_1, \dots, x_N$. We can write this as:

$$X_k = \sum_{n=0}^{N-1} x_n \, e^{-\frac{2i\pi}{N}kn},$$

where capital $X_k$ is our transformed number.

The logic behind the FFT is to re-write the DFT in terms of its odd- and even-indexed components, so that we can use the periodicity of complex numbers in order to reduce the total number of evaluations we would have to make in order to apply the DFT. In other words, we can write capital $X_k$ and $X_{k+\frac{N}{2}}$ as:

$$X_k = E_k + e^{-\frac{2i\pi}{N}k} O_k$$

$$X_{k+\frac{N}{2}} = E_k - e^{-\frac{2i\pi}{N}k} O_k,$$

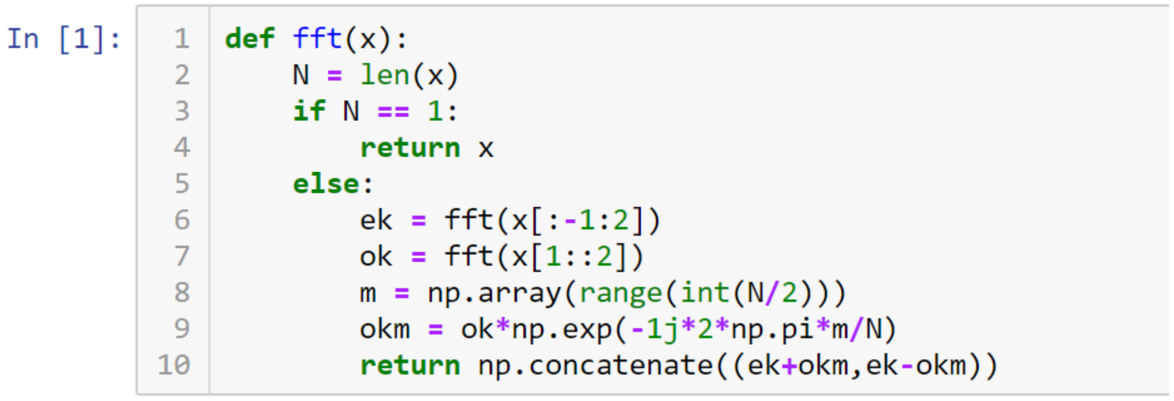
where:

$$E_k = \sum_{n=0}^{\frac{N}{2}-1} x_{2n} \, e^{-\frac{2i\pi}{N}k2n}$$

$$O_k = \sum_{n=0}^{\frac{N}{2}-1} x_{2n+1}\, e^{-\frac{2i\pi}{N}k2n}.$$

## The FFT algorithm

We can apply the FFT in Python using the following function:



In [1]:

```python
def fft(x):
    N = len(x)
    if N == 1:
        return x
    else:
        ek = fft(x[:-1:2])
        ok = fft(x[1::2])
        m = np.array(range(int(N/2)))
        okm = ok*np.exp(-1j*2*np.pi*m/N)
        return np.concatenate((ek+okm,ek-okm))
```

## Getting from the FFT to a price

We will now look at how to price a call option for a range of strikes. Before we continue, we need to define some notation. Let $k := log(K)$, $s_t := log(S_t)$, $q_{s_T}(s)$ be the density of $s_T$, and $C_T(k)$ be the price of a call option with strike $e^k$ and with maturity T. We can write the price of our call option as:

$$C_T(k) = e^{-rT}\int_k^{\infty}(e^s - e^k)q_{s_T}(s)ds.$$

In the notes, we go through how to simply this down to a point where we can apply the FFT. For now, we are just going to give the final results necessary to price a call in Python. All we need to apply the FFT to pricing a call option is the following set of equations:

$$C_T(k_m) \approx \frac{e^{-\alpha k_m}}{\pi} Re(X_m),$$

where:

$$X_m = \frac{e^{-\alpha k}}{\pi}\left(\frac{\upsilon_T(0)\delta_u}{2} + \sum_{n=1}^{N-1} e^{in\delta_u k}\,\upsilon_T(n\delta_u)\delta_u\right)$$

$$\upsilon_T(u) = \frac{e^{-rT}\phi_{S_T}(u-(\alpha+1)i)}{\alpha^2+\alpha-u^2+i(2\alpha+1)u}$$

$$\phi_{S_T}(u) = \exp\left(iu\left[log(S_0) + \left(r - \frac{\sigma^2}{2}\right)T\right] - \frac{u^2 T\sigma^2}{2}\right).$$

Note that $X_m$ is a Discrete Fourier Transform, which means that we can apply the Fast Fourier Transform.

## Applying the Fast Fourier Transform in Python

We are going to be using the following parameter values to price a call option:

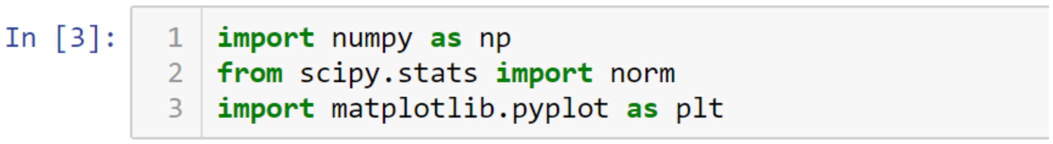
```
In [2]:
#Share info
S0 = 100
sigma = 0.3
T = 1
r = 0.06

#Algorithm info
N = 2**10
delta = 0.25
alpha = 1.5
```

where $S0$ is the initial share price, sigma is our volatility, $r$ is the risk-free interest rate, $T$ is the term of the option, $N$ is the size of the sequence of $x_n$ values that we are going to use, delta is the

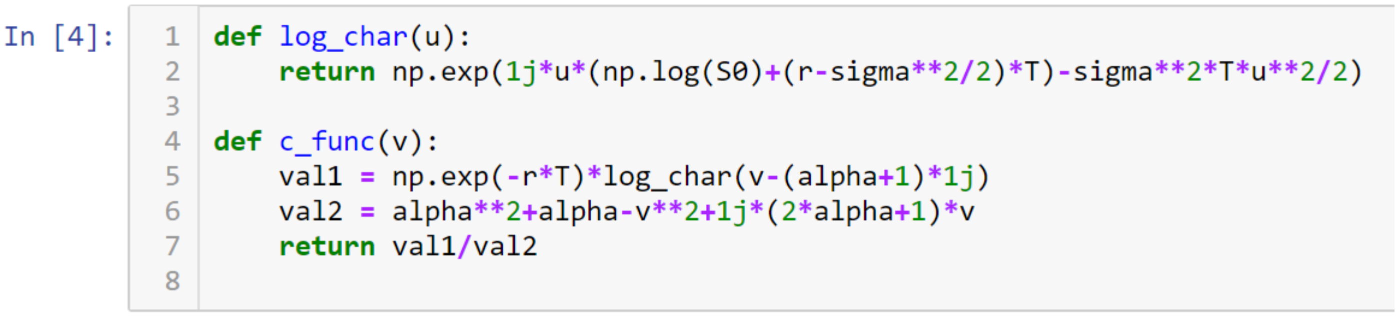
$\delta_u$ variable in the equations above, and alpha is the $\alpha$ parameter in the equations above. Remember that we will be pricing the call option for a range of strike values.

We can code this as:



```
In [3]:
import numpy as np
from scipy.stats import norm
import matplotlib.pyplot as plt
```

Which are the relevant libraries.

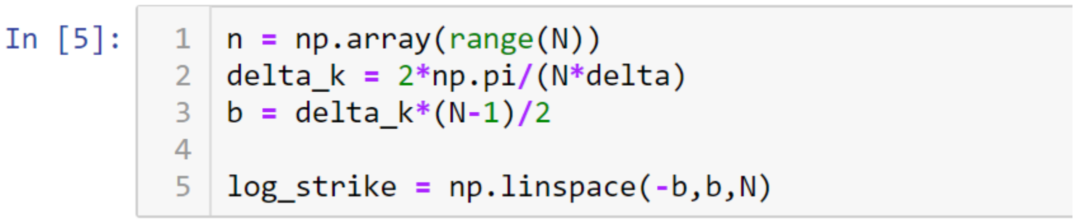


```
In [4]:
def log_char(u):
    return np.exp(1j*u*(np.log(S0)+(r-sigma**2/2)*T)-sigma**2*T*u**2/2)

def c_func(v):
    val1 = np.exp(-r*T)*log_char(v-(alpha+1)*1j)
    val2 = alpha**2+alpha-v**2+1j*(2*alpha+1)*v
    return val1/val2

```

Which are the $\phi_{s_T}(u)$ and $\upsilon_T(u)$ defined above.



```
In [5]:
n = np.array(range(N))
delta_k = 2*np.pi/(N*delta)
b = delta_k*(N-1)/2

log_strike = np.linspace(-b,b,N)
```

Which are the variables required to vectorize the final problem.

```
In [6]:
x = np.exp(1j*b*n*delta)*c_func(n*delta)*delta
x[0] = x[0]*0.5
x[-1] = x[-1]*0.5

xhat = fft(x).real

fft_call = np.exp(-alpha*log_strike)*xhat/np.pi
```

Which calculates our final call price estimates.

Which should produce the following graphs:

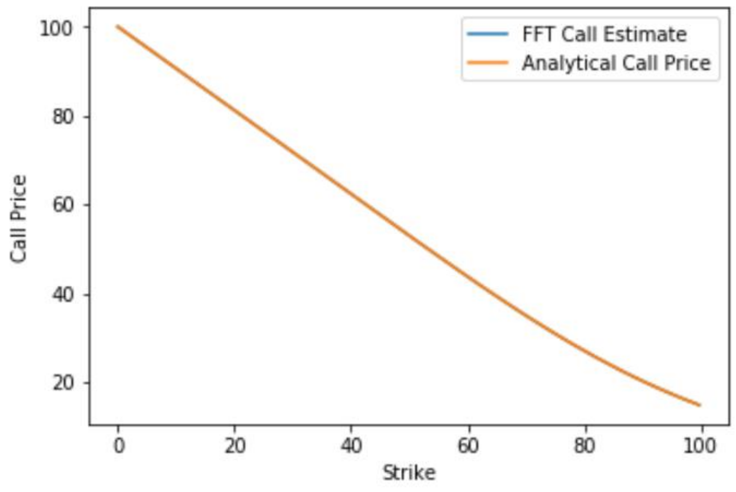


**Figure 4: FFT call price estimates for different strikes**

To show how close the estimate is, we plot the absolute error of the estimate and the analytical price in figure (5).

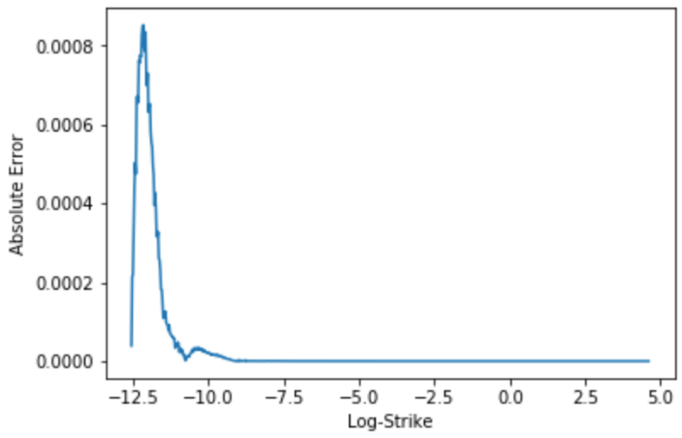


**Figure 5: Absolute difference between analytical call price and estimated FFT price**

# Unit 4: Notes

This unit will derive the mathematics underlying the Fast Fourier Transformation (FFT). As the name would suggest, the FFT is able to evaluate functions very quickly. This can be particularly important if, for example, you spot an arbitrage in a market: the person who can evaluate their expected price and spot an arbitrage fastest, will be able to make most use of the arbitrage. The FFT is also particularly useful when we get to calibrating models. We will essentially be guessing values for our parameters, so methods that allow for efficient evaluation of the function (with these parameter guesses) will allow for us to efficiently calibrate our model.

## Discrete Fourier transform

Before we move on to the FFT itself, we first need to go through what the Discrete Fourier transform (DFT) is. The DFT is a transformation which is particularly well suited for efficient calculations. In fact, it is a clever application of the DFT that makes the FFT so fast.

Now, let's define the DFT. Suppose that we have a sequence of numbers, $x_0, x_1, \dots, x_{N-1}$. We can convert this sequence into a new sequence via the following transformation:

$$X_k = \sum_{n=0}^{N-1} x_n \, e^{-\frac{2i\pi}{N}kn}. \tag{54}$$

Equation (54) is known as the standard form of the DFT.

## Fast Fourier transform

Now that we have defined what the DFT is, we can move on to developing the FFT. Note that we will be referring to some of the mathematics in the first and second set of notes (so make sure you are reasonably comfortable with what we have been doing up until this point).

The FFT takes the DFT and applies it in a “smarter” way. In other words, the FFT is just an algorithm for applying DFT. A very popular FFT algorithm is the one suggested by Cooley and Tukey (1965). This applies a divide and conquer approach to recursively breakdown the DFT into smaller, and more manageable, pieces.

## The mathematics for Cooley and Tukey's algorithm

Before we get to the algorithm itself, let's first go through the mathematics on which the method is built. We can rewrite the equation in (54) in terms of its odd- and even-indexed components:

$$X_k = \sum_{n=0}^{\frac{N}{2}-1} x_{2n}\, e^{-\frac{2i\pi}{N}k2n} + \sum_{n=0}^{\frac{N}{2}-1} x_{2n+1}\, e^{-\frac{2i\pi}{N}k(2n+1)}$$

$$= \sum_{n=0}^{\frac{N}{2}-1} x_{2n}\, e^{-\frac{2i\pi}{N}k2n} + e^{-\frac{2i\pi}{N}k} \textstyle\sum_{n=0}^{\frac{N}{2}-1} x_{2n+1} e^{-\frac{2i\pi}{N}k2n}$$

$$= E_k + e^{-\frac{2i\pi}{N}k} O_k,$$

where $E_k$ denotes the even-indexed terms, and $O_k$ denotes the odd-indexed terms.

Now, note that complex exponentials are periodic. This is the basis of the Cooley-Tukey FFT because it allows us to conclude that:

$$e^{-\frac{2i\pi}{N}k} = -e^{-\frac{2i\pi}{N}\left(k+\frac{N}{2}\right)}$$

$$E_k = E_{k+\frac{N}{2}}$$

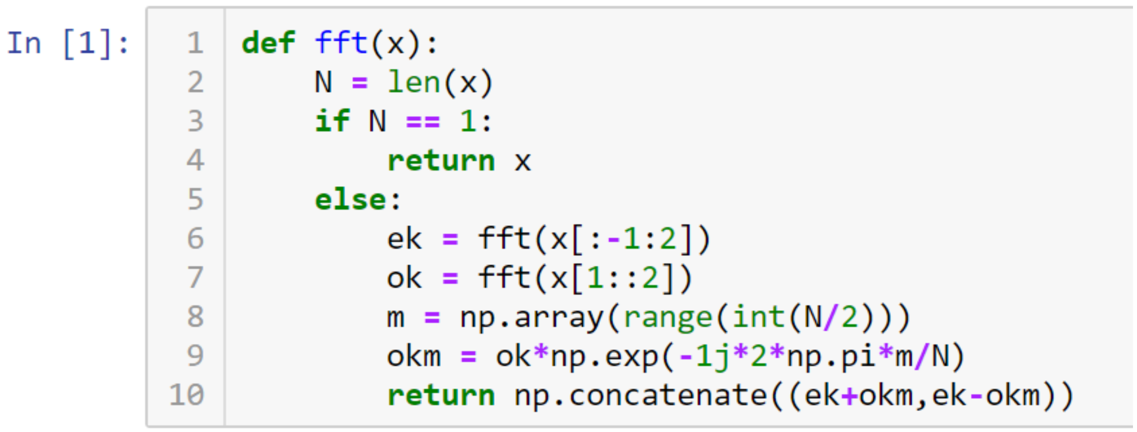
$$O_k = O_{k+\frac{N}{2}}$$

$$X_{k+\frac{N}{2}} = E_k - e^{-\frac{2i\pi}{N}k}O_k. \tag{55}$$

In the last line of (55), if we calculate $E_k$ and $e^{-\frac{2i\pi}{N}k}O_k$ once, we get the 2 DFT values out. This essentially works to almost half the total number of evaluations of the initial DFT process.

## The algorithm suggested by Cooley and Tukey

We will present the Python code for this algorithm in this unit. The next section will include code that applies this algorithm. Note that the algorithm works for a sequence of $N$ numbers.



```
In [1]:
def fft(x):
    N = len(x)
    if N == 1:
        return x
    else:
        ek = fft(x[:-1:2])
        ok = fft(x[1::2])
        m = np.array(range(int(N/2)))
        okm = ok*np.exp(-1j*2*np.pi*m/N)
        return np.concatenate((ek+okm,ek-okm))
```

The function above works recursively. It can be difficult to visualize, but the basic idea that is that it keeps calling itself until the initial input vector has been partitioned to a single element. Thereafter, it applies the result of (55). Note that the NumPy package does come with its own FFT function. In the same way that you should always use the NumPy functions for the sake of sampling from distributions like the normal distribution, you should also use the NumPy FFT function. This can be called with the following code (where $x$ is the sequence of numbers):

```
In [ ]:   np.fft.fft(x)
```

## Pricing using the FFT

We will now go through how to apply the above to pricing some vanilla option. The method we are going to go through was first proposed by Carr and Madan (1999).

Before we move on, we need to define a few new symbols. Define $k := log(K)$, $s_t := log(S_t)$, $q_{s_T}(s)$ to be the density of $s_T$, and $C_T(k)$ to be the price of a call option with strike $e^k$ and with maturity $T$. We can write:

$$C_T(k) = e^{-rT} \int_k^{\infty} (e^s - e^k) q_{s_T}(s) ds.$$

Given our current definition of the call price, the function isn't strictly integrable. So, we need to introduce a dampening factor which controls the behaviour of our integral for extreme values. Define $c_T(k) := e^{\alpha k} C_T(k)$. Let's look at the Fourier transform of $c_T(k)$:

$$\widehat{c_T}(k) = \int_{-\infty}^{\infty} e^{iuk}\, c_T(k) dk.$$

This means that we can write (using the inverse transform):

$$C_T(k) = \frac{e^{-\alpha k}}{2\pi}\int_{-\infty}^{\infty} e^{-iuk}\,\widehat{c_T}(u)du$$

$$= \frac{e^{-\alpha k}}{\pi}\int_{0}^{\infty} e^{-iuk}\,\widehat{c_T}(u)du. \tag{56}$$

The second line of (56) comes from the fact that the call price is strictly real so, when considering Euler's formula, the remaining function is an even function. Now, we can calculate $\widehat{c_T}(u)$ by substituting our expression for $c_T(k)$ and $C_T(k)$ as:

$$\widehat{c_T}(u) = \int_{-\infty}^{\infty} e^{iuk}\, e^{-rT} e^{\alpha k}\int_{k}^{\infty} (e^s - e^k) q_{s_T}(s)dsdk$$

$$= \int_{-\infty}^{\infty} e^{-rT}\, q_{s_T}(s)\int_{-\infty}^{s} e^{\alpha k}(e^s - e^k)e^{iuk}\, dkds$$

$$= \frac{e^{-rT}\phi_{s_T}(u-(\alpha+1)i)}{\alpha^2+\alpha-u^2+i(2\alpha+1)u}.$$

We now need to handle the upper bound of the integral for the call price, and the fact that we don't know the solution to the integral. The first problem can be handled relatively easily: just place an upper bound on the integral (a sufficiently large $N$ will suffice). The second problem can be solved by estimating the integral through a simple quadrature approach. Putting these together, we have:

$$C_T(k) \approx \frac{e^{-\alpha k}}{\pi}\left(\sum_{n=0}^{N-1} e^{in\delta_u k}\,\widehat{c_T}(n\delta_u)\delta_u\right)$$

where $\delta_u$ is the step length (normally some small value).

Now, suppose we evaluate the call function for log-strike values over the interval $[-b, b]$. Define a step length for the log-strike such that $k_m := -b + \delta_k m$.

If we restrict our value for $\delta_k$ so that the following is satisfied: $\delta_u \delta_k = \frac{2\pi}{N}$, then we can re-rewrite our call price as:

$$C_T(k_m) \approx \frac{e^{-\alpha k_m}}{\pi} \left( \sum_{n=1}^{N-1} e^{-i\frac{2\pi}{N}nm} x_n \right),$$

where $x_n = e^{ibn\delta_u} \widehat{c_T}(n\delta_u)\delta_u$.

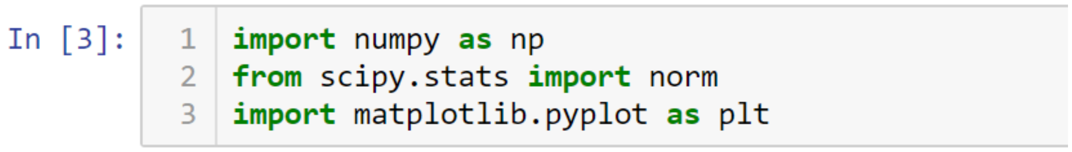
Remember equation (54)? This allows to write the above as a function of an FFT generated variable:

$$C_T(k_m) \approx \frac{e^{-\alpha k_m}}{\pi} Re(X_m)$$

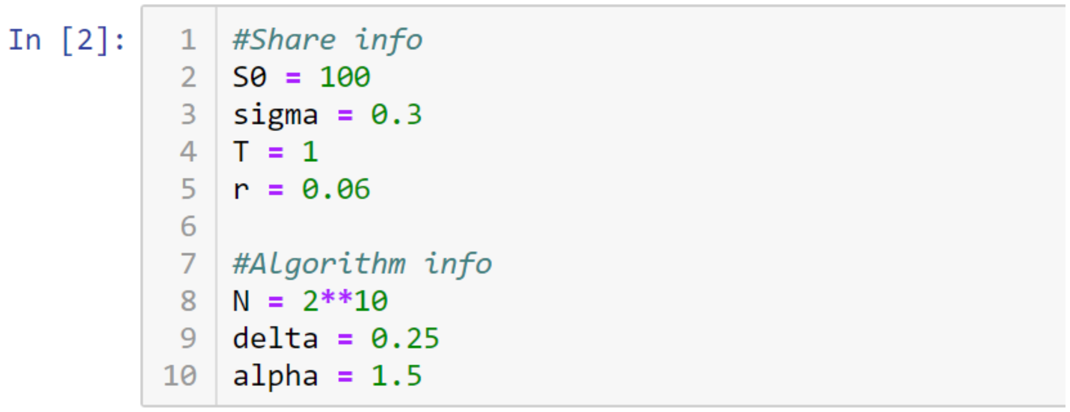
for $m = 0,1, \dots, N-1$. Note that, in applying the above approximation, we need to use a quadrature scheme that is more precise than simple quadrature. The trapezoidal approximation is generally better than a simple quadrature approach, and just requires the first and last $x_n$ to be halved (this will be made clearer when this is applied in Python below).

## Applying the Fast Fourier Transform in Python

We will be using the Fast Fourier Transform (FFT) algorithm presented in unit 3 in order to price a call option. We will be pricing a call option using the following libraries and parameter values:



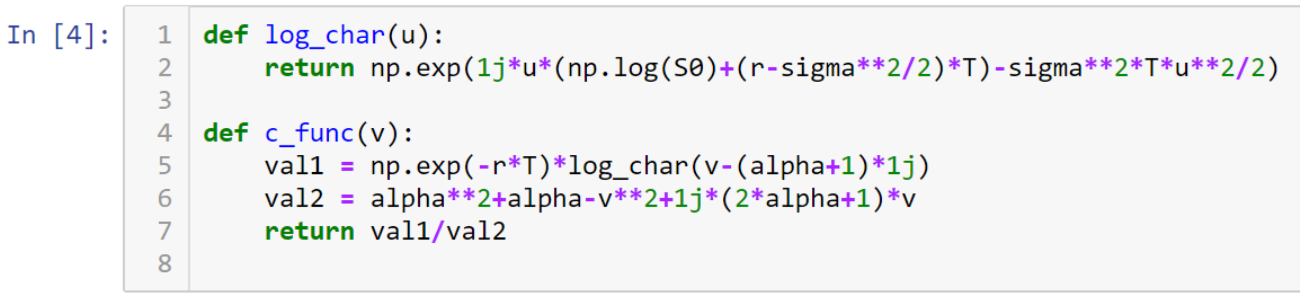
```
In [3]:
import numpy as np
from scipy.stats import norm
import matplotlib.pyplot as plt
```



```
In [2]:
#Share info
S0 = 100
sigma = 0.3
T = 1
r = 0.06

#Algorithm info
N = 2**10
delta = 0.25
alpha = 1.5
```

Now we need to define two functions that we will use to calculate the call price. These are:



```
In [4]:
def log_char(u):
    return np.exp(1j*u*(np.log(S0)+(r-sigma**2/2)*T)-sigma**2*T*u**2/2)

def c_func(v):
    val1 = np.exp(-r*T)*log_char(v-(alpha+1)*1j)
    val2 = alpha**2+alpha-v**2+1j*(2*alpha+1)*v
    return val1/val2

```

$\widehat{c_T}(u)$ is determined by the `c_func` function, whilst `log_char` evaluates the characteristic function of $s_T$, where $s_T = log(S_T)$.

The next step is to create a few variables that we are going to use to vectorize our pricing function:

```
In [5]:
n = np.array(range(N))
delta_k = 2*np.pi/(N*delta)
b = delta_k*(N-1)/2

log_strike = np.linspace(-b,b,N)
```

The variable, $n$, is just the set $[0,1, \ldots, N-1]$. The variable `delta_k` is $\delta_k$, and is used to find $b$, the upper bound of the strike segmentation interval (which is calculated in the next line).

Finally, we calculate our $x_n$ values, divide the first $x_n$ value by $2$, and calculate our FFT of $x_n$.

```
In [6]:
x = np.exp(1j*b*n*delta)*c_func(n*delta)*delta
x[0] = x[0]*0.5
x[-1] = x[-1]*0.5

xhat = fft(x).real

fft_call = np.exp(-alpha*log_strike)*xhat/np.pi
```

Note that the variable `fft_call` gives us a vector of call prices over different strike values.

Now we just need to calculate Black-Scholes prices for our range of strikes (so that we can compare our estimates):

```
In [7]:
#Call price
d_1 = (np.log(S0/np.exp(log_strike))+(r+sigma**2/2)*T)/(sigma*np.sqrt(T))
d_2 = d_1 - sigma*np.sqrt(T)
analytic_callprice = S0*norm.cdf(d_1)-np.exp(log_strike)*np.exp(-r*(T))*norm.cdf(d_2)
```

And last, we plot our estimate and the analytical price (note that it's not just one graph: the estimate is so precise that the graphs are essentially superimposed). This can be seen in figure (6).

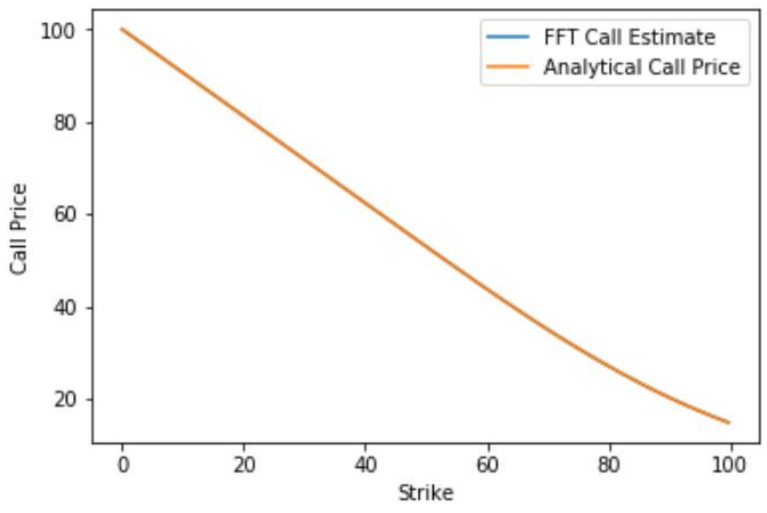


**Figure 6: FFT call price estimates for different strikes**

To further show how close the estimate is, we plot the absolute error of the estimate and the analytical price in figure (7)

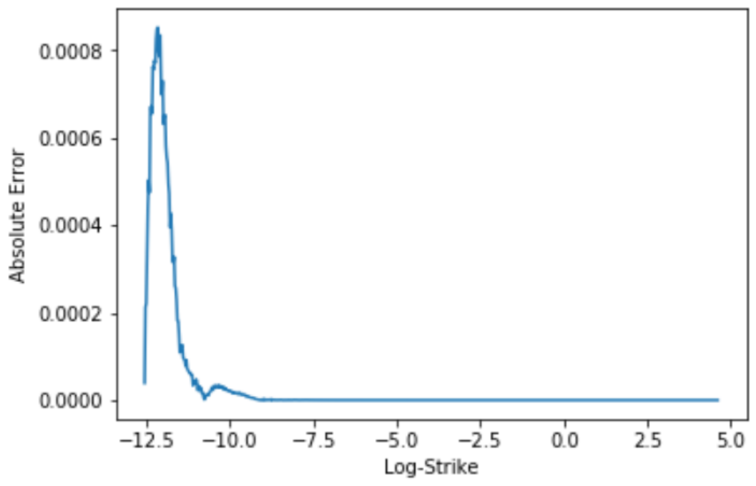


**Figure 7: Absolute difference between analytical call price and estimated FFT price**

Note that the FFT doesn't perform as well for lower strikes but performs consistently well for the money strikes.

## Bibliography

Carr, P. and Madan, D. (1999). “Option Valuation Using the Fast Fourier Transform”, *Journal of Computational Finance* 2(4): 61–73.

Cooley, J.W. and Tukey, J.W. (1965). “An Algorithm for the Machine Calculation of Complex Fourier Series”, *Mathematics of Computation* 19(90): 297–301.

Fang, F. (2010). *The Cos Method: An Efficient Fourier Method for Pricing Financial Derivatives.*

Hilpisch, Y. (2015). *Derivatives analytics with Python: data analysis, models, simulation, calibration and hedging*. John Wiley & Sons.

Rouah, F. D. (2013). *The Heston Model and Its Extensions in Matlab and C*. John Wiley & Sons.

# Collaborative Review Task

In this module, you are required to complete a collaborative review task that is designed to test your ability to apply and analyze the knowledge you have learned in the module.

## Case Study

In this assignment, you are going to be required to price a European put option using the three different techniques introduced in this module. The parameters are as follows:

- Risk-free continuously-compounded interest rate, r, of 10%
- Strike price, K, of \$100
- Initial stock price, $S_0$, of \$120
- Time to maturity, $T$, of 2 years
- Stock volatility, σ, of 25%

You may make all the assumptions of the standard Black-Scholes model.

1 Initialize the relevant parameters.

2 Calculate the analytical price for the put option using the closed-form solution.

3 Calculate the estimate for the put price using the characteristic function and technique described in note set 1. Use a maximum integral bound of 40 and $N = 200$ rectangles to approximate your integral.

4 Observe that the payoff of a put option for the variable

$s_T = log\left(\frac{S_T}{K}\right)$ is $p(s) = K(1-e^s)\mathbb{I}_{s\leq 0}$ This means that our $v_n$ co-efficients in the COS method become:

$$v_n = \frac{2}{b_2-b_1}\int_{b_1}^{0} K(1-e^s)cos\left(n\pi\frac{s-a}{b-a}\right)ds$$

$$= \frac{2}{b_2-b_1}K\left(\psi_n(b_{1,}0) - \gamma_n(b_{1,}0)\right)$$

Using this information, use the COS method to price a European put option using the given parameters.

Calculate put prices for increasing upper limits on the COS method sum.

Plot your results against the analytical put price.

**5** Use the fast Fourier transform (FFT) to price a European put option with the given parameters. You will have to price your put option for a range of strikes. Note that the only difference when applying the FFT to a put, when compared to a call, is that your value for $\alpha$ must be less than -1. Use a sequence length for your $x_n$ of $N = 2^{10}$, and use $\delta_u = 0.25$. Plot the estimated put prices against the analytical put prices (across the range of strikes).