



# City Research Online

## City St George's, University of London

**Citation:** Černý, A. (2004). Introduction to Fast Fourier Transform in Finance. *Journal of Derivatives*, 12(1), pp. 73-88. doi: 10.3905/jod.2004.434538

This is the accepted version of the paper.

This version of the publication may differ from the final published version. To cite this item please consult the publisher's version.

**Permanent repository link:** <https://openaccess.city.ac.uk/id/eprint/16292/>

**Link to published version:** <https://doi.org/10.3905/jod.2004.434538>

**Copyright and Reuse:** Copyright and Moral Rights remain with the author(s) and/or copyright holders. Copies of full items can be used for personal research or study, educational, or not-for-profit purposes without prior permission or charge, unless otherwise indicated, provided that the authors, title and full bibliographic details are credited, a hyperlink and/or URL is given for the original metadata page and the content is not changed in any way. For full details of reuse please refer to [City Research Online policy](#).

# Introduction to Fast Fourier Transform in Finance

Aleš Černý

*Tanaka Business School, Imperial College London, South Kensington Campus,  
SW7 2AZ, London, UK. Phone ++44 20 7594 9185. Fax ++44 20 7823 7685.  
a.cerny@imperial.ac.uk*

*First draft: July 2003*

*Revised: 18th June 2004*

*Typos in eqs. (38-40) corrected 20th February 2006*

---

## Abstract

The Fourier transform is an important tool in Financial Economics. It delivers real time pricing while allowing for a realistic structure of asset returns, taking into account excess kurtosis and stochastic volatility. Fourier transform is also rather abstract and therefore off-putting to many practitioners. The purpose of this paper is to explain the working of the fast Fourier transform in the familiar binomial option pricing model. We argue that a good understanding of FFT requires no more than some high school mathematics and familiarity with roulette, bicycle wheel, or a similar circular object divided into equally sized segments. The returns to such a small intellectual investment are overwhelming.

*Key words:* fast Fourier transform, option pricing, binomial lattice, chirp- $z$  transform, GAUSS, MATLAB

---

**JEL classification code:** C63, G12

**Mathematics subject classification:** 65T50, 91B70, 91B24

The Fourier transform is becoming an increasingly popular and important tool in Financial Economics because it delivers real time pricing while allowing for important properties of asset returns, such as excess kurtosis, stochastic volatility and leverage effects, discussed in Heston [1993], Carr and Madan [1999], Carr and Wu [2004]. These impressive results come at a price in the form of a considerable abstraction which can be quite off-putting to practitioners. The aim of this paper is to explain the working of the discrete Fourier transform (DFT) and its fast implementation (FFT) in the familiar binomial option pricing model. The binomial model serves two purposes. It highlights, in an accessible way, the usefulness of FFT, which is an important computational tool in its own right, and has many other applications in Finance. It also motivates the passage to continuous time thereby providing intuition behind fast pricing formulae in a very rich class of models used in the industry.

The paper is divided into three parts: I – Discrete Fourier transform and binomial option pricing; II – Efficient implementation of DFT by means of fast Fourier transform, with examples in GAUSS and MATLAB; III – Fourier transform and continuous-time option pricing.

## 1 Discrete Fourier transform and binomial option pricing

This section explains how and why option prices in the binomial model can be computed via discrete Fourier transform. We assume that the reader is familiar with the concept of risk-neutral pricing. To begin with, we introduce complex numbers and discuss their geometric properties, especially as they regard the unit circle; then we define the Discrete Fourier Transform (DFT) and highlight some of its properties. The following section introduces a simple binomial option pricing example and shows how the pricing procedure can be performed on a circle. To conclude, we demonstrate how to transform circular convolutions using DFT and obtain the Fourier transform pricing formula. The resulting formula is put to practice in part II, which shows how to accelerate DFT by means of FFT algorithm and provides simple GAUSS and MATLAB codes for illustration. Real-world applications of the Fourier transform pricing formula are discussed in part III.

### 1.1 Introduction to complex numbers

The discrete Fourier transform is about *evenly spaced points on a circle*. From the mathematical point of view, evenly distributed points on a circle are most easily described by complex numbers. This section reviews the geometry of those numbers, which in turn determine the properties of Fourier transform.

Complex numbers are a convenient way of capturing vectors in a two-dimensional

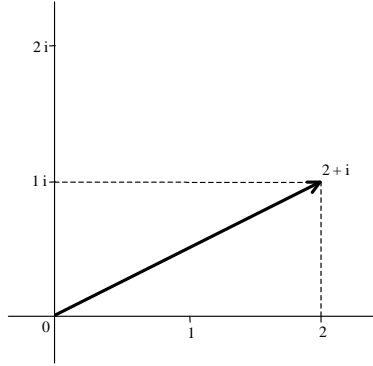


Fig. 1. Complex number as a two-dimensional vector.

space. For example, Exhibit 1 depicts a vector

$$2 + i;$$

it is a point in the plane if we move two units on the *real* (horizontal) *axis* and one unit on the *imaginary* (vertical) *axis*. This terminology is somewhat unfortunate; the imaginary axis is no less real than the real axis. It would be more appropriate to talk about ‘horizontal’ and ‘vertical’ numbers.

The rules for addition of complex numbers are the same as with vectors, for example

$$\begin{bmatrix} 2 \\ 1 \end{bmatrix} + \begin{bmatrix} 3 \\ -4 \end{bmatrix} = \begin{bmatrix} 5 \\ -3 \end{bmatrix}$$

translated into complex notation would read

$$(2 + i) + (3 - 4i) = 5 - 3i.$$

Likewise, multiplication by a scalar (a real number) works like for vectors;

$$-3 \begin{bmatrix} 2 \\ 1 \end{bmatrix} = \begin{bmatrix} -6 \\ -3 \end{bmatrix}$$

translates into complex numbers as

$$-3(2 + i) = -6 - 3i.$$

## 1.2 Complex multiplication

*Complex numbers are very good at describing the movement around a unit circle.* As shown in Exhibit 2a, unit circle intersects the real axis at points  $-1, 1$ , and the imaginary axis at points  $-i$  and  $i$ .

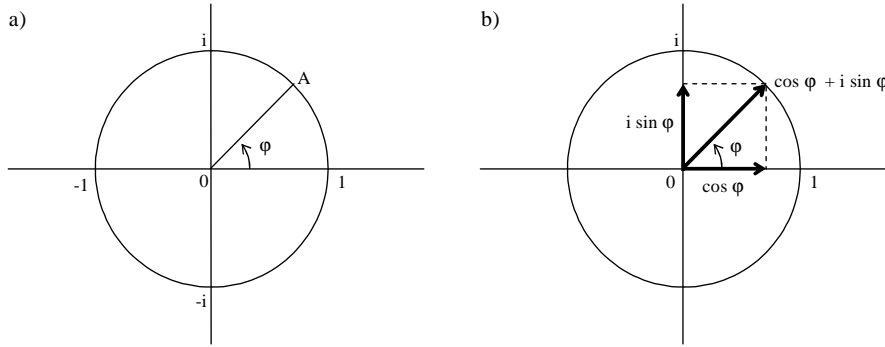


Fig. 2. Point on the unit circle expressed as a complex number.

A point  $A$  on the unit circle is uniquely characterized by its *argument*  $\varphi$  – the angle between the real axis and the line  $OA$ . More specifically, Exhibit 2b shows that the point  $A$  can be expressed as  $\cos \varphi + i \sin \varphi$ .

On most computers the functions  $\sin$  and  $\cos$  are implemented in such a way that the angle  $\varphi$  must be given in *radians*. Radians measure the distance travelled on the perimeter of the unit circle. The entire perimeter of the unit circle has length  $2\pi$  which corresponds to  $360^\circ$ . The angle corresponding to  $i$  is  $90^\circ$  or  $\frac{\pi}{2}$ , the angle corresponding to  $-1$  is  $180^\circ$  or  $\pi$  and so on, as shown in Exhibit 3.

<b>Angle in degrees</b>	0	30	60	90	180	270	360
<b>Angle in radians</b>	0	$\frac{\pi}{6}$	$\frac{\pi}{3}$	$\frac{\pi}{2}$	$\pi$	$\frac{3}{2}\pi$	$2\pi$

Table 3

Conversion table between degrees and radians.

**Facts:**

- *Multiplying complex numbers on a unit circle means adding angles.* The angle of  $i$  is  $90^\circ$ , the angle of  $i \times i$  will be  $90^\circ + 90^\circ = 180^\circ$  which corresponds to  $-1$ , see Exhibit 4a. In complex number notation this gives the famous formula

$$i \times i = i^2 = -1. \tag{1}$$

- With (1) in hand the general definition of complex multiplication follows naturally

$$\begin{aligned} (a_1 + ib_1) \times (a_2 + ib_2) &= a_1a_2 + i(b_1a_2 + a_1b_2) + b_1b_2i^2 = \\ &= a_1a_2 - b_1b_2 + i(b_1a_2 + a_1b_2). \end{aligned} \tag{2}$$

- It also follows that the ‘multiplication is adding angles’ rule works quite generally on the unit circle

$$\begin{aligned} (\cos \varphi_1 + i \sin \varphi_1) \times (\cos \varphi_2 + i \sin \varphi_2) &= \\ = \cos(\varphi_1 + \varphi_2) + i \sin(\varphi_1 + \varphi_2). \end{aligned} \tag{3}$$

- One can express points on the unit circle more elegantly using the *Euler formula*

$$\cos \varphi + i \sin \varphi = e^{i\varphi}, \tag{4}$$

whereby (3) becomes

$$e^{i\varphi_1} \times e^{i\varphi_2} = e^{i(\varphi_1 + \varphi_2)}, \tag{5}$$

see Exhibit 4b.

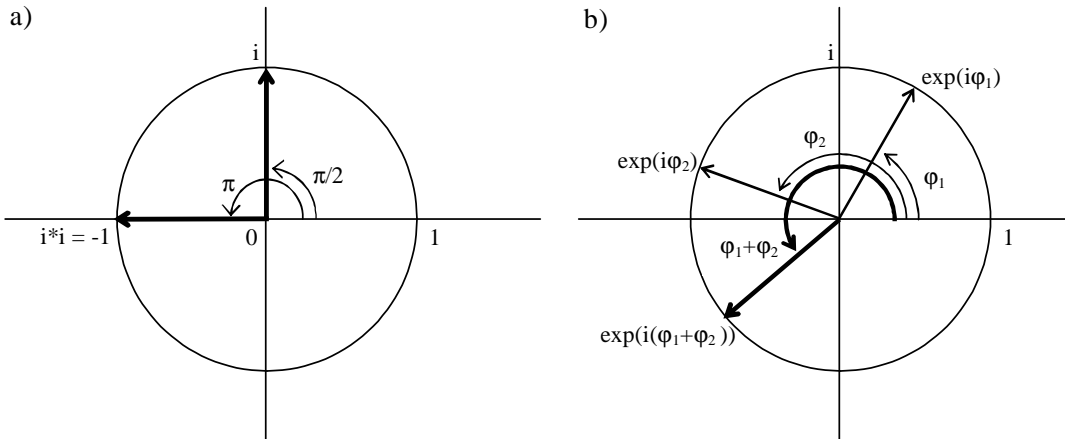


Fig. 4. Complex multiplication on a unit circle means adding angles.

### 1.3 Geometry of spoked wheels

It is very easy to construct a wheel with evenly placed spokes using complex numbers. Suppose we want to place five evenly spaced points on the unit circle.

One fifth of the full circle is characterized by the angle  $\frac{2\pi}{5}$ , hence the first spoke will be placed at  $e^{i\frac{2\pi}{5}}$ . Let us denote this number by  $z_5$  (fifth root of unity)

$$z_5 \equiv e^{i\frac{2\pi}{5}}.$$

Since the multiplication by  $z_5$  causes anticlockwise rotation by one fifth of full circle the second spoke will be  $(z_5)^2$  the third spoke at  $(z_5)^3$  and so on, see Exhibit 5a.

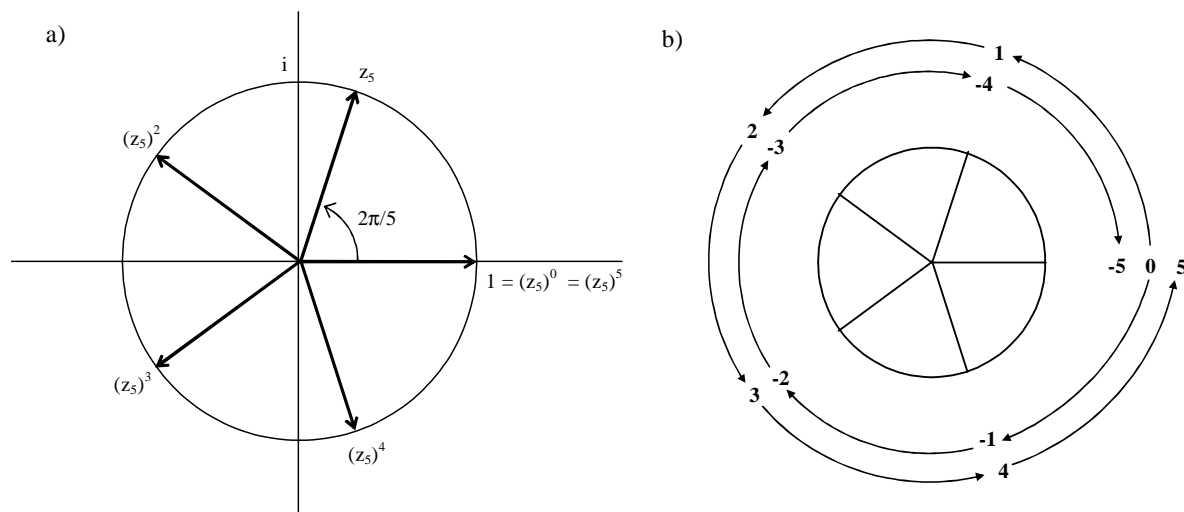


Fig. 5. a) Evenly distributed points on a circle. b) Number of elementary rotations required to reach a particular spoke (+ anticlockwise, - clockwise).

This provides a natural numbering of the spokes, according to how many elementary rotations are needed to reach the particular spoke. Note that since we are moving in a circle we will come back to the starting point after five rotations anticlockwise

$$\begin{aligned} (z_5)^0 &= (z_5)^5 = (z_5)^{10} = (z_5)^{15} = \dots \\ (z_5)^1 &= (z_5)^6 = (z_5)^{11} = (z_5)^{16} = \dots \text{etc.}, \end{aligned}$$

and also after five rotations clockwise

$$\begin{aligned} (z_5)^0 &= (z_5)^{-5} = (z_5)^{-10} = (z_5)^{-15} = \dots \\ (z_5)^1 &= (z_5)^{-4} = (z_5)^{-9} = (z_5)^{-14} = \dots \text{etc.} \end{aligned}$$

Thus the numbering of spokes is ambiguous; for example indices 0, 5, -5 refer to the same spoke, see Exhibit 5b.

The following box summarizes the most important properties of evenly spaced points on the unit circle. These properties are essential for the understanding of the discrete Fourier transform.

- Let  $z_n$  be a rotation by one  $n$ th of a full circle

$$z_n \equiv e^{i\frac{2\pi}{n}}.$$

Then

$$(z_n)^0 + (z_n)^1 + \dots + (z_n)^{n-1} = 0 \quad (6)$$

for any  $n$ . This is because the points  $(z_n)^0, (z_n)^1, \dots, (z_n)^{n-1}$  are evenly distributed on a unit circle and thus the result of summation must not change if we rotate the set of points by one  $n$ th of a full circle. The only vector that remains unchanged after such rotation is zero vector.

- One can generalize this result further. Let  $k$  be an integer between 1 and  $n - 1$ . Then

$$(z_n^k)^0 + (z_n^k)^1 + \dots + (z_n^k)^{n-1} = 0 \quad (7)$$

for any  $n$ . The reason for this result is again rotational symmetry of points  $(z_n^k)^0, (z_n^k)^1, \dots, (z_n^k)^{n-1}$ . The difference from (6) is that in the sequence  $(z_n)^0, (z_n)^1, \dots, (z_n)^{n-1}$  each spoke occurs *exactly once*, whereas in  $(z_n^k)^0, (z_n^k)^1, \dots, (z_n^k)^{n-1}$  the same spoke can occur several times (try  $n = 4, k = 2$ ).

- The case with  $k = 0$  requires special attention. Since  $(z_n^0)^j = 1$  for all  $j$  we have

$$(z_n^0)^0 + (z_n^0)^1 + \dots + (z_n^0)^{n-1} = n.$$

To summarize,

$$(z_n^k)^0 + (z_n^k)^1 + \dots + (z_n^k)^{n-1} = n \quad \text{for } k = 0, \pm n, \pm 2n, \dots \quad (8)$$

$$(z_n^k)^0 + (z_n^k)^1 + \dots + (z_n^k)^{n-1} = 0 \quad \text{for } k \neq 0, \pm n, \pm 2n, \dots \quad (9)$$

#### 1.4 Reverse order on a circle

Given a sequence of  $n$  numbers  $a = [a_0, a_1, \dots, a_{n-1}]$  we can say that

$$\text{rev}(a) \equiv [a_0, a_{n-1}, \dots, a_1]$$

is  $a$  in *reverse order*. If  $a$  is written around a circle in *anticlockwise* direction then  $\text{rev}(a)$  is found by reading from  $a_0$  in clockwise direction, see Exhibit 6. Note that  $\text{rev}(a)$  is *not* equal to  $[a_{n-1}, \dots, a_1, a_0]$ .

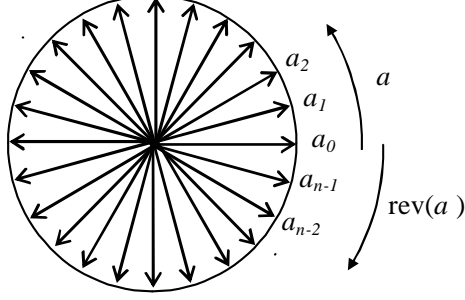


Fig. 6. Reverse order on a circle.

For any  $k$  the sequence  $(z_n^k)^0, (z_n^k)^1, \dots, (z_n^k)^{n-1}$  is the same as the sequence  $(z_n^{-k})^0, (z_n^{-k})^1, \dots, (z_n^{-k})^{n-1}$  taken in the reverse order:

$$\text{rev} \left( (z_n^{-k})^0, (z_n^{-k})^1, \dots, (z_n^{-k})^{n-1} \right) = (z_n^k)^0, (z_n^k)^1, \dots, (z_n^k)^{n-1}. \quad (10)$$

This is because  $(z_n^{-k})^{n-j} = z_n^{-kn+kj} = z_n^{kj} = (z_n^k)^j$  for any  $j$ .

### 1.5 Discrete Fourier Transform (DFT)

As in the previous section take  $z_n \equiv e^{i\frac{2\pi}{n}}$  (this number is called the  $n$ th root of unity). Let  $a_0, a_1, \dots, a_{n-1}$  be a sequence of  $n$  (in general complex) numbers. The *discrete Fourier transform* of  $a_0, a_1, \dots, a_{n-1}$  is the sequence  $b_0, b_1, \dots, b_{n-1}$  such that

$$\begin{aligned} b_k &= \frac{a_0 (z_n^k)^0 + a_1 (z_n^k)^1 + \dots + a_{n-1} (z_n^k)^{n-1}}{\sqrt{n}} = \\ &= \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} a_j z_n^{jk} = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} a_j e^{i\frac{2\pi}{n}jk} \end{aligned} \quad (11)$$

We write

$$\mathcal{F}(a) = b.$$

Equation (11) represents the *forward transform*. The *inverse transform* is

$$\begin{aligned} \tilde{a}_l &= \frac{\tilde{b}_0 (z_n^{-l})^0 + \tilde{b}_1 (z_n^{-l})^1 + \dots + \tilde{b}_{n-1} (z_n^{-l})^{n-1}}{\sqrt{n}} = \\ &= \frac{1}{\sqrt{n}} \sum_{k=0}^{n-1} \tilde{b}_k z_n^{-kl} = \frac{1}{\sqrt{n}} \sum_{k=0}^{n-1} \tilde{b}_k e^{-i\frac{2\pi}{n}kl}, \end{aligned} \quad (12)$$

and we write

$$\tilde{a} = \mathcal{F}^{-1}(\tilde{b}).$$

**Facts:**

- The inverse discrete Fourier transform of sequence  $\tilde{b}_0, \tilde{b}_1, \dots, \tilde{b}_{n-1}$  is the same as the forward transform of the same sequence in reversed order

$$\mathcal{F}^{-1}(\tilde{b}) = \mathcal{F}(\text{rev}(\tilde{b})), \quad (13)$$

and vice versa

$$\mathcal{F}^{-1}(\text{rev}(\tilde{b})) = \mathcal{F}(\tilde{b}). \quad (14)$$

This is a direct consequence of (10).

- $\mathcal{F}^{-1}$  is indeed an inverse transformation to  $\mathcal{F}$ , that is

$$\mathcal{F}^{-1}(\mathcal{F}(a)) = \mathcal{F}(\mathcal{F}^{-1}(a)) = a. \quad (15)$$

This result relies on (8) and (9); for a proof see Appendix.

### 1.6 Binomial option pricing

Consider a monthly distribution of FTSE 100 return calibrated to reflect market volatility of 4.4% a month and expected rate of return 0.9% a month:

$$\begin{aligned} pR_u + (1-p)R_d &= 1.009 \\ pR_u^2 + (1-p)R_d^2 &= 0.044^2 + 1.009^2. \end{aligned}$$

Choosing the objective probability to be  $p = \frac{1}{2}$  we solve for  $R_u$  and  $R_d$

$$R_u = 1.053 \text{ with } p_u = \frac{1}{2} \quad (16)$$

$$R_d = 0.965 \text{ with } p_d = \frac{1}{2}. \quad (17)$$

Assuming that the initial value of FTSE Index is 5100.00 points, the evolution of the index in the three months ahead is given by the lattice in Exhibit 7.

Suppose we wish to price a call option struck at  $K = 5355$  (5% out of the money), maturing 3 months from now. The intrinsic value of the option at

number of low returns	$S(0)$	$S(1)$	$S(2)$	$S(3)$
0	5100.00	5370.30	5654.93	5954.64
1		4921.50	5182.34	5457.00
2			4749.25	5000.96
3				4583.02

Table 7  
Binomial stock price lattice.

maturity is

$$C(3) = \begin{bmatrix} 599.64 \\ 102.00 \\ 0.00 \\ 0.00 \end{bmatrix}. \quad (18)$$

Asset pricing theory tells us that the no-arbitrage price of the pay-off

$$\begin{bmatrix} C_u \\ C_d \end{bmatrix}$$

is given as the risk-neutral expectation of the discounted pay-off

$$\text{no-arbitrage value}(C) = \frac{q_u C_u + q_d C_d}{R_f}, \quad (19)$$

where the risk-neutral probabilities  $q_u$  and  $q_d$  are chosen such that the risk-neutrally expected return of all basis assets is equal to the risk-free return

$$\begin{aligned} q_u + q_d &= 1 \\ q_u R_u + q_d R_d &= R_f. \end{aligned}$$

The values  $q_u/R_f$  and  $q_d/R_f$  are known as *state prices*.

Assuming a risk-free rate equivalent to 4% per annum the monthly risk-free return is

$$R_f = 1.04^{1/12} = 1.0033.$$

This gives conditional risk-neutral probabilities of

$$q_u = \frac{R_f - R_d}{R_u - R_d} = \frac{1.0033 - 0.965}{1.053 - 0.965} = 0.43523, \quad (20)$$

$$q_d = \frac{R_u - R_f}{R_u - R_d} = \frac{1.053 - 1.0033}{1.053 - 0.965} = 0.56477, \quad (21)$$

and the valuation formula:

$$\text{no-arbitrage value}(C) = \frac{0.43523C_u + 0.56477C_d}{1.0033}. \quad (22)$$

Recursive application of (22) with terminal value (18) leads to option prices in Exhibit 8.

number of low returns	$C(0)$	$C(1)$	$C(2)$	$C(3)$
0	81.36	162.66	317.54	599.64
1		19.19	44.25	102.00
2			0.00	0.00
3				0.00

Table 8  
Option prices in a binomial lattice.

### 1.7 Option pricing on a circle

For any two  $n$ -dimensional vectors  $a = [a_0, a_1, \dots, a_{n-1}]$ ,  $b = [b_0, b_1, \dots, b_{n-1}]$  we define *circular (cyclic) convolution* of  $a$  and  $b$  to be a new vector  $c$ ,

$$c = a \circledast b,$$

such that

$$c_j = \sum_{k=0}^{n-1} a_{j-k} b_k. \quad (23)$$

One will immediately note that the index  $j - k$  can be negative. If this occurs, we will simply add  $n$  to get the result between 0 and  $n - 1$ ; this practice is consistent with the spoke numbering introduced in Section 1.3, and it merely reflects movement in a circle.

Graphically one can evaluate the circular convolution as follows:

- (1) Set up two concentric circles divided into  $n$  equal segments. Write  $a$  around the *inner circle clockwise* and  $b$  around the *outer circle anticlockwise*. Exhibit 9 shows this for  $n = 4$ .
- (2) Perform a scalar multiplication between the two circles. In Exhibit 9 this would give

$$a_0 b_0 + a_3 b_1 + a_2 b_2 + a_1 b_3.$$

The result is  $c_0$ .

- (3) *Turn the inner circle anticlockwise* by  $\frac{1}{n}$ th of a full circle. Repeat the scalar multiplication between the circles. The result is  $c_1$ . In Exhibit 10

$$c_1 = a_1 b_0 + a_0 b_1 + a_3 b_2 + a_2 b_3.$$

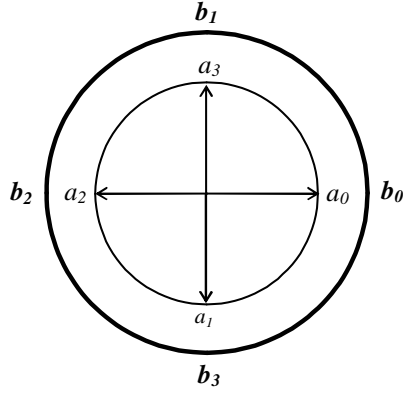


Fig. 9. Computing the first element of circular convolution  $a \circledast b$ .

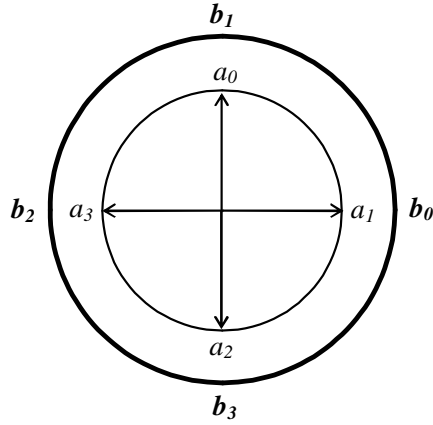


Fig. 10. Computing the second element of the circular convolution  $a \circledast b$ .

- (4) Repeat this procedure to compute  $c_2, \dots, c_{n-1}$ , each time giving the inner circle  $\frac{1}{n}$ th turn anticlockwise.

How can one use the circular convolution for option pricing? If we write both the option pay-off and the pricing kernel in the clockwise direction and then rotate the option pay-off in the anticlockwise direction, we will obtain option prices in the natural order from highest to lowest.

To be specific, let us go back to the binomial option pricing model. At maturity the option can have four different values:

$$C(3) = \begin{bmatrix} 599.64 & 102.00 & 0.00 & 0.00 \end{bmatrix}.$$

Let vector  $q$  contain the conditional one-period risk-neutral probabilities  $q_u = 0.43523$ ,  $q_d = 0.56477$ . Since there are just two states over one period the remaining entries will be padded by zeros:

$$q = \begin{bmatrix} q_u & q_d & 0.00 & 0.00 \end{bmatrix}.$$

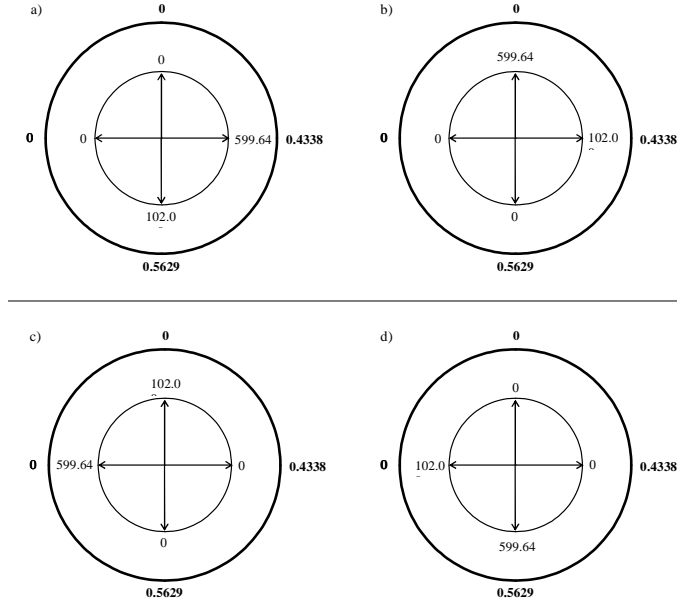


Fig. 11. Option pricing on a circle. Option pay-offs and state prices written in clockwise direction, the inner circle rotates anticlockwise, representing equation (24)  
a)  $c_0 = 599.64 \times 0.4338 + 102.00 \times 0.5629$ ; b)  $c_1 = 102.00 \times 0.4338 + 0 \times 0.5629$ ;  
c)  $c_2 = 0 \times 0.4338 + 0 \times 0.5629$ ; d)  $c_3 = 0 \times 0.4338 + 599.64 \times 0.5629$ ;

Finally, recall that the risk-free return is  $R_f = 1.0033$ . Thus to compute option prices at time  $t = 2$  we need to evaluate

$$c = C(3) \otimes \text{rev}(q/R_f).$$

This operation is depicted graphically in Exhibit 11, where the option pay-offs  $C(3)$  are on the inner circle and the state prices  $q/R_f$  are on the outer circle, both written in clockwise direction. Numerically we obtain

$$C(3) \otimes \text{rev}(q/R_f) = \begin{bmatrix} 317.54 & 44.25 & 0 & 337.54 \end{bmatrix}. \quad (24)$$

Note that we only need the first three prices in (24). The last entry is meaningless – it corresponds to the no-arbitrage price of the pay-off  $[0 \ 599.64]$ .

**Option pricing on a circle.**

Consider a binomial model where  $C(j)$  is the vector of option prices at date  $j = 0, 1, \dots, N$ . Denote by  $q$  the vector containing the risk-neutral probabilities  $q_u$  and  $q_d$ , padded by zeros to have the same dimension as  $C(N)$ . By backward substitution,

$$\begin{aligned}
 C(N-1) &= C(N) \otimes \text{rev}(q) / R_f, \\
 C(N-2) &= C(N) \otimes \text{rev}(q) \otimes \text{rev}(q) / R_f^2, \\
 C(j) &= C(N) \otimes \overbrace{\text{rev}(q) \otimes \text{rev}(q) \otimes \dots \otimes \text{rev}(q)}^{(N-j) \text{ times}} / R_f^j, \tag{25}
 \end{aligned}$$

The vectors  $C(j)$  computed in this manner have more entries than needed, the useful  $j + 1$  entries are at the top end of each vector.

Numerical results are reported in Exhibit 12, the relevant entries are highlighted and should be compared with those in Exhibit 8.

number of low returns	$C(0)$	$C(1)$	$C(2)$	$C(3)$
0	<b>81.36</b>	<b>162.66</b>	<b>317.54</b>	<b>599.64</b>
1	115.28	<b>19.19</b>	<b>44.25</b>	<b>102.00</b>
2	265.47	190.01	<b>0.00</b>	<b>0.00</b>
3	232.62	325.17	337.54	<b>0.00</b>

Table 12

Option prices obtained from circular pricing formula (25). The useful entries are in bold.

*1.8 Circular pricing via discrete Fourier transform*

In this section we will reformulate the circular pricing formula (25) using the discrete Fourier transform. Although we derive the Fourier pricing formula mechanically, in part III we will spell out its more intuitive probabilistic interpretation.

The discrete Fourier transform has one very useful property – it turns circular convolutions into products:

$$\mathcal{F}(a \otimes b) = \sqrt{n} \mathcal{F}(a) \mathcal{F}(b), \tag{26}$$

$$\mathcal{F}^{-1}(a \otimes b) = \sqrt{n} \mathcal{F}^{-1}(a) \mathcal{F}^{-1}(b), \tag{27}$$

$$n = \text{dimension of } a; \tag{28}$$

see Appendix for a proof. This can be used to a great advantage in pricing.

Recall from the preceding section that

$$C_0 = C_N \otimes \overbrace{\text{rev}(q) \otimes \text{rev}(q) \otimes \dots \otimes \text{rev}(q)}^{N \text{ times}} / R_f^N,$$

where  $N$  is the number of time periods to maturity. Now apply the inverse transform  $\mathcal{F}^{-1}$  to both sides, using property (27) on the right hand side

$$\mathcal{F}^{-1}(C_0) = \mathcal{F}^{-1}(C_N) \times \left( \sqrt{\text{dimension of } C_N} \mathcal{F}^{-1}(\text{rev}(q)) / R_f \right)^N. \quad (29)$$

In a binomial model the dimension of  $C_N$  is  $N + 1$ . Furthermore, recall from (14) that  $\mathcal{F}^{-1}(\text{rev}(q)) = \mathcal{F}(q)$  and substitute this into (29)

$$\mathcal{F}^{-1}(C_0) = \mathcal{F}^{-1}(C_N) \times \left( \sqrt{N + 1} \mathcal{F}(q) / R_f \right)^N.$$

Finally, apply the forward transform to both sides again and use (15) on the left hand side:

$$C_0 = \mathcal{F} \left( \mathcal{F}^{-1}(C_N) \times \left( \sqrt{N + 1} \mathcal{F}(q) / R_f \right)^N \right).$$

### Option pricing via discrete Fourier transform.

Consider a model with IID stock returns and constant interest rate, represented by a recombining binomial tree with  $N$  periods and  $N + 1$  trading dates. Let the  $(N + 1)$ -dimensional vector  $C_N$  be the pay-off of the option at expiry. Let  $q$  contain the one-step risk-neutral probabilities as the first two entries, with the remaining  $N - 1$  entries being zeros. Then the first element of  $(N + 1)$ -dimensional vector  $C_0$ ,

$$C_0 = \mathcal{F} \left( \mathcal{F}^{-1}(C_N) \times \left( \sqrt{N + 1} \mathcal{F}(q) / R_f \right)^N \right), \quad (30)$$

is the no-arbitrage price of the option at time 0. The role of the forward and inverse transforms is symmetrical, that is we also have

$$C_0 = \mathcal{F}^{-1} \left( \mathcal{F}(C_N) \times \left( \sqrt{N + 1} \mathcal{F}^{-1}(q) / R_f \right)^N \right). \quad (31)$$

## 2 Fast Fourier Transform (FFT)

This section deals with the implementation of the pricing formula (30) on a computer using fast DFT routines, known as FFTs. It is highly unlikely that the reader will want to write his or her own DFT code, for this would be counterproductive given the wealth and the level of specialization of ready-made algorithms. The use of prepackaged algorithms saves time, but with little documentation at hand implementing otherwise sound mathematical formula

may not prove straightforward. This section provides guidelines that ensure a trouble-free transition between the theoretical pricing formula (30) and a computer code using a DFT routine of reader's choice, with specific examples given in GAUSS and MATLAB.

Two main issues arise in the use of (fast) DFT routines: 1) finding out the mathematical definition of a specific DFT routine, and 2) choosing the right input length to make the computation fast. We now address these two issues in turn.

- (1) Every textbook, and indeed every computer language, defines the forward and inverse transforms slightly differently. Thus the first task of any user is to find out how a given computer routine, call it `dft`, is related to the theoretical transforms  $\mathcal{F}$  and  $\mathcal{F}^{-1}$  defined in (11) and (12). To do so, one proceeds in two simple steps:

- (a) In the first step one determines the normalization factor. Define  $a = [1 \ 0 \ 0 \ 0]$  and compute  $\tilde{a} = \text{dft}(a)$ . If  $\tilde{a}_0 = 0.25$  then

$$\text{either } \text{dft} = \mathcal{F}/\sqrt{n} \text{ or } \text{dft} = \mathcal{F}^{-1}/\sqrt{n};$$

else if  $\tilde{a}_0 = 0.5$  then

$$\text{either } \text{dft} = \mathcal{F} \text{ or } \text{dft} = \mathcal{F}^{-1};$$

and if  $\tilde{a}_0 = 1$  then

$$\text{either } \text{dft} = \sqrt{n}\mathcal{F} \text{ or } \text{dft} = \sqrt{n}\mathcal{F}^{-1}.$$

- (b) To ascertain whether one is dealing with a forward or an inverse transform, one defines  $b = [0 \ 1 \ 0 \ 0]$  and evaluates  $\tilde{b} = \text{dft}(b)$ . If the imaginary part of  $\tilde{b}_1$  is positive then `dft` is proportional to  $\mathcal{F}$ , otherwise it is proportional to  $\mathcal{F}^{-1}$ . In the case of the lattice pricing formulae (30) and (31) one will use two routines, say `dft` and `dfti`, which are inverse to each other. In this instance it does not really matter which of the two transforms is forward and which is inverse. But there are other applications (see Section 3), where it is absolutely crucial to know whether a given routine is proportional to  $\mathcal{F}$  or  $\mathcal{F}^{-1}$ .

**Example 1** *In GAUSS the two DFT transforms are called `dffft` and `dfffti`, respectively, and they are related to  $\mathcal{F}$  and  $\mathcal{F}^{-1}$  as follows:*

$$\begin{aligned} \text{dffft}(a) &\equiv \frac{\mathcal{F}^{-1}(a)}{\sqrt{n}}, \\ \text{dfffti}(a) &\equiv \sqrt{n}\mathcal{F}(a), \end{aligned}$$

where  $n$  is the dimension of vector  $a$ . Equation (30) therefore becomes

$$C_0 = \text{dfffti} \left( \text{dffft}(C_N) \times (\text{dfffti}(b))^N \right). \quad (32)$$

Suppose the vectors  $\mathbf{C}_N$  and  $\mathbf{b}$  have already been defined in GAUSS. To compute the option price at  $t = 0$  we would use the following code:

```

C_0 = dffti( dfft(C_N).*(dfft(b)^N) );           (33)
print 'no-arbitrage price at t=0 is ' C_0[1];

```

The `.*` command stands for element-by-element multiplication.

The DFT algorithm is approximately three times faster than the backward recursion in binomial model; the computational time for both algorithms grows quadratically with the number of periods<sup>1</sup>, see Exhibit 13.

trading interval in minutes	number of periods	execution time in seconds	
		DFT	backward recursion
60	504	0.15	0.4
30	1008	0.6	1.6
15	2016	2.3	6.4
5	6048	20.8	61.6

Pentium III 750MHz, 128Mb RAM, GAUSS

Table 13

Comparison of pricing speed in a binomial lattice between backward recursion and discrete Fourier transform formula (30).

- (2) A naive implementation of DFT algorithm with  $n$ -dimensional input requires  $n^2$  complex multiplications (see example above). An efficient implementation of DFT, known as the fast Fourier transform (FFT), will only require  $Kn \ln n$  operations<sup>2</sup>, but one still has to choose  $n$  carefully because the constant  $K$  can be very large for some choices of  $n$ . Some FFT implementations automatically restrict the transform length to the most suitable values of  $n$  (typically  $n = 2^p$  or  $n = 2^p 3^q 5^r$ ), which is the case in GAUSS. Others, such as MATLAB, will compute FFT of any length; here it is particularly important for the user to choose  $n$  sensibly, otherwise the FFT algorithm may turn out to be very slow indeed.

**Example 2** *The forward and inverse FFT in MATLAB are called `fft` and `ifft`, respectively:*

$$\text{fft}(a) \equiv \sqrt{n} \mathcal{F}^{-1}(a) \quad (34)$$

$$\text{ifft}(a) \equiv \frac{\mathcal{F}(a)}{\sqrt{n}}, \quad (35)$$

<sup>1</sup> GAUSS programmes *Binomial.gss* and *DFT.gss* available from author's website.

<sup>2</sup> The fast Fourier transform does not appear in undergraduate textbooks on numerical mathematics and the most useful references on the introductory level are web based, see <http://www.fftw.org/links.html>, and in particular the online manual Hey [1999]. An efficient implementation of FFT for all transform lengths is suggested in Frigo and Johnson [1998]; it is used in Matlab. Efficient implementation of mixed 2, 3, 5-radix algorithm is due to Temperton [1992]; it is used in GAUSS. Duhamel and Vetterli [1990] is an excellent survey of FFT algorithms.

where  $n$  is the dimension of vector  $a$ . The option pricing equation (30) therefore becomes

$$C_0 = \text{ifft} \left( \text{fft}(C_N) \times ((N+1) \times \text{ifft}(b))^N \right),$$

which in terms of MATLAB code reads

```
C_0 = ifft( fft(C_N).*((N+1)*ifft(b)).^N );      (36)
sprintf 'no-arbitrage price at t=0 is %0.2f' C_0(1);
```

The commands `.*` and `.^` stand for element-by-element multiplication and exponentiation, respectively.

There are many instances when FFT of length  $n_1$  is faster than FFT of length  $n_2$  even though  $n_1 > n_2$ . This somewhat counterintuitive phenomenon is illustrated in Exhibit 14.

$n$	factorization	execution time in seconds
499 979	499 979	27.2
1048 575	$3 \times 5^2 \times 11 \times 31 \times 41$	5.2
1048 576	$2^{20}$	0.93
1080 000	$2^6 3^3 5^4$	0.11

Pentium III 750MHz, 128Mb RAM, MATLAB

Table 14

Execution time of FFT algorithm for different input lengths  $n$ .

To understand why some transform lengths are more suitable than others we need one piece of terminology and one fact: i) FFT algorithm for length  $n = 2^p$  is called radix-2 algorithm; ii) the higher the  $b$  the slower the radix- $b$  algorithm per output length. There is one notable exception: radix-4 is faster than radix-2 by about 25%.

In practice, one uses transforms of size  $n = 2^p 3^q 5^r$ . If the original vector size is *not* of this form, then a sufficient number of zeros is added. Ideally,  $q$  and  $r$  should be small compared to  $p$  because of the fact ii) above. The advantage of using mixed-radix algorithms is twofold: a) more transform lengths are available, which means one need not pad the input with too many zeros; b) one can use the operation-saving prime factor algorithm<sup>3</sup>.

To illustrate the item a), with vector size  $2^{10} + 1 = 1025$  the next available size for radix-2 algorithm is  $n = 2048 = 2^{11}$  but with mixed 2,3,5-radix algorithm one could use length  $n = 1080 = 2^3 3^3 5$  which is nearly twice as small and consequently the Fourier transform evaluation is twice as fast compared to radix-2 algorithm. To illustrate property b), one should notice that highly composite lengths such as  $1080000 = 2^6 3^3 5^4$  evaluate faster than simple powers of similar length such as  $2^{20} = 1048 576$ , see Exhibit 14. Transforms which are not of the length  $n =$

<sup>3</sup> The prime factor algorithm (PFA) works faster because the factors 2, 3 and 5 have no common divisors, see Temperton [1992].

$2^p 3^q 5^r$  can take very long to compute, especially if  $n$  is a large prime, again see Exhibit 14.

**Example 3** *MATLAB* will allow the user to perform FFT of any length; this is done using commands (34) and (35). However, as we have noted above, it is eminently sensible to restrict transform lengths to  $n = 2^p 3^q 5^r$  with  $q$  and  $r$  small relative to  $p$  to obtain the best performance. *MATLAB* provides function `nextpow2` giving the next bigger power of 2. In addition, *MATLAB* allows the user to specify the transform length by including it as a second optional argument of `fft` and `ifft`. Hence a fast implementation of (36) in *MATLAB* would read:

```
length = 2^nextpow2(N+1);
C_0 = ifft( fft(C_N,length).*((length*ifft(b,length)).^N) );
```

The padding of the original input `C_N` by zeros to the dimension `length` is done automatically.

To find the nearest transform length of the form  $n = 2^p 3^q 5^r$  one can use the following code:

```
length = N+1;
while max(factor(length)) > 5;
    length = length+1;
end;
```

**Example 4** In *GAUSS* the fast Fourier forward and inverse transforms are performed by functions `fftn` and `ffti`. These functions use Temper-ton's [1992] mixed 2,3,5-radix algorithm, and the padding of input vector by zeros to the nearest available length  $n = 2^p 3^q 5^r$  is done automatically. If  $n$  is the input dimension the output dimension from `fftn` and `ffti` will be `nextn(n)`. In terms of *GAUSS* code one writes similarly as in (33):

```
C_0 = ffti( fftn(C_N).*(ffti(b)^N) );
```

One can increase the speed further by choosing a composite length  $n = 2^p 3^q 5^r$  where  $q$  and  $r$  are non-zero but small relative to  $p$ . The optimal length is given by *GAUSS* function `optn(N+1)`, and the padding by zeros to this dimension must be performed by the user.

The FFT implementation of binomial pricing algorithm<sup>4</sup> has a blistering speed compared to the DFT, see Exhibit 15.

Because it is so fast one can explore higher trading frequencies and see that the Black–Scholes formula really does describe the limiting value, see Exhibit 16. Note that the Black–Scholes formula itself is still about 10 000 times faster than the FFT algorithm.

<sup>4</sup> *GAUSS* code *FFT.gss* available from author's website.

trading interval in minutes	number of periods	execution time in seconds	
		<b>DFT</b>	<b>FFT</b>
30	1008	0.6	0.003
15	2016	2.3	0.006
5	6048	20.8	0.022
1	30240	510	0.27

Pentium III 750MHz, 128Mb RAM, GAUSS

Table 15  
Speed of binomial pricing using DFT and FFT algorithms.

$\Delta t$ (seconds)	Black-Scholes			
	60	10	1	0
<b>Option price</b>	75.93398	75.93284	75.93286	75.93288
<b>Option delta</b>	0.31668534	0.31668346	0.31668334	0.31668331

Table 16  
Option price and option delta in continuous-time limit and its binomial approximation.

### 3 Further applications of FFT in finance

Practical applications of DFT (FFT) in modern finance go beyond the binomial model, but the essential structure of the pricing formulae is that of equation (30). To motivate the passage to continuous time, let us rewrite the DFT pricing equation (30) to take explicit account of the maturity date  $T$  and the rebalancing frequency  $\Delta t$ , with  $N_{\Delta t} = T/\Delta t$  trading periods and instantaneous risk-free rate  $r$ :

$$\begin{aligned}
C_0 &= \mathcal{F} \left( \mathcal{F}^{-1} (C_{T,\Delta t}) \times \left( \sqrt{N_{\Delta t} + 1} \mathcal{F} (q_{\Delta t}) e^{-r\Delta t} \right)^{N_{\Delta t}} \right) \\
&= e^{-rT} \mathcal{F} \left( \mathcal{F}^{-1} (C_{T,\Delta t}) \times \left( \sqrt{N_{\Delta t} + 1} \mathcal{F} (q_{\Delta t}) \right)^{N_{\Delta t}} \right). \tag{37}
\end{aligned}$$

The quantity  $\left( \sqrt{N_{\Delta t} + 1} \mathcal{F} (q_{\Delta t}) \right)^{N_{\Delta t}}$  is known as the (risk-neutral) *characteristic function* of log stock price, and in practice one is mainly interested in models where the *continuous-time limit* of (37) is available in closed form. This is the case in the class of exponential Lévy models with affine stochastic volatility process, discussed in Carr and Wu [2004]. This class contains a large number of popular models allowing for excess kurtosis, stochastic volatility and leverage effects. It includes, among others, the stochastic volatility models of Heston [1993], Duffie et al. [2000] and all exponential Lévy models (see, for example, Madan and Seneta [1990] and Eberlein et al. [1998]). For an exhaustive characterization of affine processes see Duffie et al. [2003].

In the continuous-time limit the discrete Fourier transform is replaced by the (continuous) Fourier transform: that is for a contingent claim with payoff  $C_T = f(\ln S_T)$  we wish to find coefficients  $\psi(v)$  such that

$$f(x) = \int_{\beta-i\infty}^{\beta+i\infty} \psi(v)e^{vx} dv \quad (38)$$

for some real constant  $\beta$ <sup>5</sup>. The recipe for obtaining the coefficients  $\psi(v)$  is known – it is given by the inverse Fourier transform<sup>6</sup>:

$$\psi(v) = \frac{1}{2\pi} \int_{-\infty}^{+\infty} f(x)e^{-vx} dx. \quad (39)$$

For example, the payoff of a European call option with strike price  $e^k$  corresponds to  $f(x) = (e^x - e^k)^+$  whereby simple integration yields

$$\psi(v) = \frac{e^{(1-v)k}}{2\pi v(v-1)} \text{ for } \operatorname{Re} v > 1.$$

Substituting for  $C_T$  from (38) the risk-neutral pricing formula reads

$$\begin{aligned} C_0(\ln S_0) &= e^{-rT} \mathbf{E}^Q [C_T(\ln S_T)] \\ &= e^{-rT} \mathbf{E}^Q \left[ \int_{\beta-i\infty}^{\beta+i\infty} \psi(v)e^{v \ln S_T} dv \right] \\ &= e^{-rT} \int_{\beta-i\infty}^{\beta+i\infty} \psi(v)e^{v \ln S_0} \mathbf{E}^Q [e^{v \ln S_T/S_0}] dv, \end{aligned} \quad (40)$$

where  $\phi^Q(-iv) := \mathbf{E}^Q [e^{v \ln S_T/S_0}]$  is the risk-neutral characteristic function of log stock return. It is now clear that the continuous-time pricing formula (40) is a direct analogy of its discrete-time counterpart (37), whereby instead of the discrete characteristic function  $\left(\sqrt{N_{\Delta t} + 1} \mathcal{F}(q_{\Delta t})\right)^{N_{\Delta t}}$  we use the continuous characteristic function  $\mathbf{E}^Q [e^{iv \ln S_T}]$ ; instead of discrete Fourier coefficients  $\mathcal{F}^{-1}(C_{T,\Delta t})$  we use the continuous coefficients  $\psi$ , and instead of summation we use integration.

There is, nevertheless, one major difference between (37) and (40): whereas the former spends significant amount of time computing the characteristic function of log returns and Fourier coefficients of the option, the latter has both quantities available in closed form. This makes the continuous-time pricing formula (40) even faster than the accelerated binomial formula (30).

<sup>5</sup> For the Fourier transform to work  $C_T(\ln S_T)S_T^{-\beta}$  must be integrable as a function of  $\ln S_T$ . There are derivative securities, such as call and put options, where one needs to take  $\beta \neq 0$  to insure integrability ( $\beta > 1$  for the call,  $\beta > 0$  for the put).

<sup>6</sup> Some unrestrictive technical conditions must hold to make sure that for  $\psi$  given in (39) equation (38) holds for all values of  $\ln S_T$ , see Chandrasekharan [1989].

**Example 5** In the celebrated Heston [1993] model,

$$\begin{aligned} d\sigma_t^2 &= (a - b\sigma_t^2)dt + \tilde{\sigma}\sigma_t dB_1^Q \\ d\ln S_t &= \left(r - \frac{\sigma_t^2}{2}\right)dt + \sigma_t dB_2^Q, \end{aligned}$$

we have

$$\begin{aligned} \phi^Q(-iv) &= \mathbb{E}^Q \left[ e^{v\ln(S_T/S_0)} \right] = e^{\gamma(v) + \delta(v)\sigma_0^2}, \\ \gamma(v) &= rTv + \frac{a}{\tilde{\sigma}^2} \left( b - \rho\tilde{\sigma}v - 2\ln \left( \frac{1 - c_2(v)e^{c_1(v)T}}{1 - c_2(v)} \right) \right), \\ \delta(v) &= \frac{(b - \rho\tilde{\sigma}v + c_1(v))(1 - e^{c_1(v)T})}{\tilde{\sigma}^2(1 - c_2(v)e^{c_1(v)T})}, \\ c_1(v) &= \sqrt{(b - \rho\tilde{\sigma}v)^2 - \tilde{\sigma}^2(v + v^2)}, \\ c_2(v) &= \frac{b - \rho\tilde{\sigma}v + c_1(v)}{b - \rho\tilde{\sigma}v - c_1(v)}, \end{aligned}$$

where  $\rho = \text{Corr}(dB_1^Q, dB_2^Q)$ .

Option pricing therefore boils down to evaluation of integrals of the type

$$\begin{aligned} C_0(\ln S_0) &= S_0 e^{-rT} \int_{\beta-i\infty}^{\beta+i\infty} \frac{e^{(v-1)(\ln S_0 - k)}}{2\pi v(v-1)} \phi^Q(-iv) dv \\ &= 2S_0 e^{-rT} \int_{\beta+i0}^{\beta+i\infty} \text{Re} \left( \frac{e^{(v-1)(\ln S_0 - k)}}{2\pi v(v-1)} \phi^Q(-iv) \right) dv, \end{aligned} \quad (41)$$

where both  $\psi(v)$  and  $\phi^Q(v)$  are known. To evaluate (41) one truncates the integral at a high value of  $\text{Im } v$  and then uses a numerical quadrature to approximate it by a sum, see Lee [2004] for a detailed exposition. This yields an expression of the type

$$C_0(\ln S_0) \approx 2 \text{Re} \left( S_0 e^{-rT} \sum_{j=0}^{n-1} w_j \frac{e^{(v_j-1)(\ln S_0 - k)}}{2\pi v_j(v_j - 1)} \phi^Q(-iv_j) \right), \quad (42)$$

where the integration weights  $w_j$  and abscissas  $v_j$  depend on the quadrature rule. It is particularly convenient to use Newton-Cotes rules, which employ equidistantly spaced abscissas. For example, a trapezoidal rule yields

$$\begin{aligned} v_j &= \beta + ij\Delta v, \\ \text{Im } v_{\max} &= (n-1)\Delta v, \\ w_0 = w_n &= \frac{1}{2}\Delta v, \\ w_1 = w_2 = \dots &= w_{n-1} = \Delta v. \end{aligned} \quad (43)$$

In conclusion, if the characteristic function of log returns is known, one needs to evaluate a single sum (42) to find the option price. Consequently, there is *no need* to use FFT if one wishes to evaluate the option price for *one fixed* log strike  $k$ .

### 3.1 FFT option pricing with multiple strikes

The situation is very different if we want to evaluate the option price (42) for many different strikes simultaneously. Let us consider  $m = 121$  values of moneyness  $\kappa_l = \ln S_0 - k_l$  ranging from  $-30\%$  to  $30\%$  with increment  $\Delta\kappa = 0.5\%$

$$\kappa_l = \kappa_{\max} - l\Delta\kappa, \quad (44)$$

$$\kappa_{\max} = 0.30, \quad l = 0, \dots, m - 1. \quad (45)$$

The idea of using FFT in this context is due to Carr and Madan [1999], and it has recently been improved upon by using so-called  $z$ -transform, see Chourdakis [2004]:

**Definition 6** *The number*

$$a_0z^0 + a_1z^{-1} + \dots + a_{n-1}z^{-(n-1)}$$

*is called the  $z$ -transform of sequence  $a$ . The discrete Fourier transform of sequence  $a$  is obtained as a special case of  $z$ -transform with  $n$  specific values of  $z$ :*

$$z_l = e^{-i\frac{2\pi}{n}l}, \quad l = 0, 1, \dots, n - 1.$$

Carr and Madan have noted that with equidistantly spaced abscissas (43) one can write the option pricing equation (42) for different strike values (44, 45) as a  $z$ -transform with  $z_l = e^{-i\Delta v \Delta \kappa l}$ :

$$C_{0l} = 2S_0 e^{(\beta-1)\kappa_l - rT} \operatorname{Re} \sum_{k=0}^{n-1} e^{i\Delta v \Delta \kappa kl} a_j, \quad (46)$$

$$a_j = w_j \frac{e^{ij\Delta v \kappa_{\max}} \phi^Q(-iv_j)}{2\pi v_j (v_j - 1)}.$$

Setting

$$\Delta v \Delta \kappa = \frac{2\pi}{n} \quad (47)$$

Carr and Madan obtain a discrete Fourier transform in (46). Chourdakis [2004] points out that there is a fast algorithm for the  $z$ -transform which works even when  $\Delta v \Delta \kappa \neq \frac{2\pi}{n}$  and  $m \neq n$ :

### Chirp- $z$ transform

Chirp- $z$  transform is an efficient algorithm for evaluating the  $z$ -transform for  $m$  different points  $z$  of the form

$$z_k = Aw^k, \quad k = 0, 1, \dots, m-1,$$

where  $A$  and  $w$  are arbitrary complex numbers. The chirp- $z$  transform works by rephrasing the original  $z$ -transform as a circular convolution and then computing this convolution by means of three FFTs as shown in Part I, Section “Circular pricing via DFT”. For more details see Bluestein [1968], Rabiner et al. [1969], and Bailey and Swartztrauber [1991]. Compared to the standard  $n$ -long FFT the chirp- $z$  algorithm is approximately  $6(\ln m + 1)/\ln n$  times slower, for  $m \leq n$ .

The MATLAB command for chirp- $z$  transform of  $n$ -long input sequence  $a$  reads

$$\text{czt}(a, m, w, A).$$

A GAUSS procedure *czt.gss* is available from author’s website.

The decision whether to use the simple summation (42)  $m$  times, or whether to apply the chirp- $z$  transform (46) depends on the desired number of strikes  $m$ . The speed of the former relative to the latter is roughly  $m/6/(\log_2 m + 1)$  times higher. As a rough guide, for  $m \approx 36$  the simple summation (42) is as fast as the chirp- $z$  formula (46), for  $m = 8$  it is three times faster, and for  $m = 150$  it is three times slower.

One also has to decide whether to force the FFT spacing of strike values (47); this is done by boosting  $n$  while keeping  $\Delta v$  fixed. Suppose that  $v_{\max}$  is chosen sufficiently high to achieve desired accuracy for a single strike. As a rule of thumb, if the initial spacing  $\frac{2\pi}{\text{Im } v_{\max}}$  is six times coarser than the desired spacing of log strikes  $\Delta \kappa$  one should use the chirp- $z$  transform, otherwise it will be faster to increase  $n$  to satisfy (47) and use the short FFT algorithm described in Bailey and Schwarztrauber [2004, pp. 392-393].

The value of  $\text{Im } v_{\max}$  tends to be higher for short maturities, and for parametric distributions with heavy tails, such as variance gamma or generalized hyperbolic. In such circumstances FFT formula (46)-(47) is preferable. Non-parametric empirical equity return distributions have characteristic functions that decay faster, leading to lower values of  $\text{Im } v_{\max}$ , leaving the chirp- $z$  transform as the best option.

## 4 Conclusions

The present paper makes three contributions. It explains the working of the discrete Fourier transform in a non-technical language in the familiar bino-

mial option pricing model. Secondly, it highlights the common perils in the computer implementation of fast DFT algorithms. Thirdly, it explains how the binomial pricing formula relates to more complex continuous-time models which allow for excess kurtosis, stochastic volatility and leverage effects and which are used routinely in the finance industry.

The present paper does not give an exhaustive account of DFT in Finance. One can quite easily extend the Fourier pricing formula from binomial lattice of Part I to multinomial lattices, see Černý [2004, Chapter 12]. Further applications of FFT appear in Albanese et al [2004], Andreas et al. [2002], Benhamou [2002], Chiarella and El-Hassan [1997], Dempster and Hong [2002], and Rebonato and Cooper [1998]. For the most up-to-date developments in option pricing using (continuous) Fourier transform see Carr and Wu [2004], and for evaluation of hedging errors refer to Černý [2003] and Hubalek et al [2004].

## Notes

I would like to thank David Miles and Jonathan Wainwright for suggesting important clarifications in an early draft. I am grateful to Peter Carr, Sanjiv Das and Stephen Figlewski, who provided helpful comments and pointers to references. This is an abridged and adapted version of of Černý [2004, Chapter 7]. GAUSS is a trademark of Aptech Systems, Inc.; MATLAB is a registered trademark of The MathWorks, Inc.

## 5 Appendix

### 5.1 Inverse Discrete Fourier Transform

To show  $\mathcal{F}^{-1}(\mathcal{F}(a)) = a$  we need to prove that for  $b = \mathcal{F}(a)$  defined in (11) we have  $\mathcal{F}^{-1}(b) = a$ . Denote  $\tilde{a} = \mathcal{F}^{-1}(b)$  and express  $\tilde{a}$  from definition (12)

$$\tilde{a}_l = \frac{1}{\sqrt{n}} \sum_{k=0}^{n-1} b_k z_n^{-kl}.$$

Now substitute for  $b_k$  from (11)

$$\tilde{a}_l = \frac{1}{n} \sum_{k=0}^{n-1} \left( \sum_{j=0}^{n-1} a_j z_n^{jk} \right) z_n^{-kl},$$

move  $z_n^{-kl}$  inside the inner summation

$$\tilde{a}_l = \frac{1}{n} \sum_{k=0}^{n-1} \left( \sum_{j=0}^{n-1} a_j z_n^{k(j-l)} \right),$$

change the order of summation

$$\tilde{a}_l = \frac{1}{n} \sum_{j=0}^{n-1} \left( \sum_{k=0}^{n-1} a_j z_n^{k(j-l)} \right),$$

and take  $a_j$  in front of the inner sum (it does not depend on  $k$ )

$$\tilde{a}_l = \frac{1}{n} \sum_{j=0}^{n-1} a_j \left( \sum_{k=0}^{n-1} \left( z_n^{j-l} \right)^k \right).$$

By virtue of (8)-(9) the inner sum  $\sum_{k=0}^{n-1} \left( z_n^{j-l} \right)^k$  equals 0 for  $j \neq l$  and for  $j = l$  it equals  $n$ . Consequently

$$\tilde{a}_l = \frac{1}{n} \sum_{j=0}^{n-1} a_j \left( \sum_{k=0}^{n-1} \left( z_n^{j-l} \right)^k \right) = a_l$$

for all  $l$  which proves that  $\mathcal{F}^{-1}(\mathcal{F}(a)) = a$ .

## 5.2 Discrete Fourier Transform of Convolutions

We wish to show  $\mathcal{F}(a \otimes b) = \sqrt{n} \mathcal{F}(a) \mathcal{F}(b)$ . Let us begin by computing  $c = a \otimes b$ . From the definition (23)

$$c_j = \sum_{k=0}^{n-1} a_{j-k} b_k. \quad (48)$$

By  $d$  denote the Fourier transform of  $c$ ,  $d = \mathcal{F}(a \otimes b)$  and use the definition (11) to evaluate  $d_l$

$$d_l = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} c_j z_n^{jl}.$$

Now substitute for  $c_j$  from (48),

$$d_l = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} \left( \sum_{k=0}^{n-1} a_{j-k} b_k \right) z_n^{jl},$$

move  $z^{jl}$  inside the inner bracket, writing it as a product  $z^{jl} = z^{(j-k)l} z^{kl}$ ,

$$d_l = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} \sum_{k=0}^{n-1} a_{j-k} z^{(j-k)l} b_k z^{kl},$$

change the order of summation,

$$d_l = \frac{1}{\sqrt{n}} \sum_{k=0}^{n-1} \sum_{j=0}^{n-1} a_{j-k} z^{(j-k)l} b_k z^{kl},$$

and take  $b_k z^{kl}$  in front of the inner summation (it does not depend on  $j$ ),

$$d_l = \frac{1}{\sqrt{n}} \sum_{k=0}^{n-1} b_k z^{kl} \left( \sum_{j=0}^{n-1} a_{j-k} z^{(j-k)l} \right). \quad (49)$$

It is easy to realize that the inner sum does not depend on  $k$ , because it always adds the same  $n$  elements; only the order in which these elements are added depends on  $k$  (we are completing one full turn around the circle, starting at  $k$ th spoke). Hence we have:

$$\sum_{j=0}^{n-1} a_{j-k} z^{(j-k)l} = \sum_{j=0}^{n-1} a_j z^{jl} \text{ for all } k,$$

and substituting this into (49) we finally obtain

$$d_l = \sqrt{n} \underbrace{\left( \frac{1}{\sqrt{n}} \sum_{k=0}^{n-1} b_k z^{kl} \right)}_{\tilde{b}_l} \underbrace{\left( \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} a_j z^{jl} \right)}_{\tilde{a}_l}.$$

From the definition of the forward transform (11)  $\tilde{a} = \mathcal{F}(a)$  and  $\tilde{b} = \mathcal{F}(b)$ , which completes the proof.

## References

- Albanese, C., Jackson, K., and Wiberg, P. A new Fourier transform algorithm for value at risk. *Quantitative Finance* 4 (2004), 328–338.
- Andreas, A., Engelmann, B., Schwendner, P., and Wystup, U. Fast Fourier method for the valuation of options on several correlated currencies. In *Foreign Exchange Risk*, J. Hakala and U. Wystup, Eds. Risk Publications, 2002.
- Bailey, D. H., and Swartztrauber, P. N. The fractional Fourier transform and applications. *SIAM Review* 33, 3 (1991), 389–404.
- Benhamou, E. Fast Fourier transform for discrete Asian options. *Journal of Computational Finance* 6, 1 (2002).
- Bluestein, L. I. A linear filtering approach to the computation of the discrete Fourier transform. *IEEE Northeast Electronics Research and Engineering Meeting 10* (1968), 218–219.
- Carr, P., and Madan, D. B. Option valuation using the fast Fourier transform. *Journal of Computational Finance* 2 (1999), 61–73.
- Carr, P., and Wu, L. Time-changed Lévy processes and option pricing. *Journal of Financial Economics* 71, 1 (2004), 113–141.
- Černý, A. The risk of optimal, continuously rebalanced hedging strategies and its efficient evaluation via Fourier transform. Tech. rep., The Business School, Imperial College London, 2003.
- Černý, A. *Mathematical Techniques in Finance: Tools for Incomplete Markets*. Princeton University Press, 2004.
- Chandrasekharan, K. *Classical Fourier Transforms*. Springer, 1989.
- Chiarella, C., and El-Hassan, N. Evaluation of derivative security prices in the heath-jarrow-morton framework as path integrals using fast Fourier transform techniques. *Journal of Financial Engineering* 6, 2 (1997), 121–147.
- Chourdakis, K. Option pricing using the fractional FFT. Working paper on [www.theponytail.net](http://www.theponytail.net), 2004.
- Dempster, M. A. H., and Hong, S. S. G. Spread option valuation and the fast Fourier transform. In *Mathematical Finance – Bachelier Congress 2000*, H. Geman, D. Madan, S. R. Pliska, and T. Vorst, Eds. Springer, 2002, pp. 203–220.
- Duffie, D., Filipović, D., and Schachermayer, W. Affine processes and applications in finance. *The Annals of Applied Probability* 13, 3 (2003), 984–1053.
- Duffie, D., Pan, J., and Singleton, K. Transform analysis and asset pricing for affine jump diffusions. *Econometrica* 68 (2000), 1343–1376.

- Duhamel, P., and Vetterli, M. Fast Fourier transforms: A tutorial review and a state of the art. *Signal Processing* 19 (1990), 259–299.
- Eberlein, E., Keller, U., and Prause, K. New insights into smile, mispricing and value at risk: The hyperbolic model. *Journal of Business* 71, 3 (1998), 371–405.
- Frigo, M., and Johnson, S. G. FFTW: An adaptive software architecture for the FFT. In *Proc. IEEE Intl. Conf. On Acoustics, Speech, and Signal Processing* (Seattle, WA, 1998), vol. 3, pp. 1381–1384.
- Heston, S. A closed-form solution for options with stochastic volatility with applications to bond and currency options. *Review of Financial Studies* 6 (1993), 327–344.
- Hey, A. *FFT Demystified*. <http://www.eptools.com/tn/T0001/INDEX.HTM>, 1999.
- Hubalek, F., Kallsen, J., and Krawczyk, L. Variance-optimal hedging and Markowitz-efficient portfolios for processes with stationary independent increments. Working paper, 2004.
- Lee, R. W. Option pricing by transform methods: Extensions, unification and error control. *Journal of Computational Finance* 7, 3 (2004).
- Madan, D., and Seneta, E. The variance gamma model for stock market returns. *Journal of Business* 63, 4 (1990), 511–524.
- Rabiner, L. R., Schafer, R. W., and Rader, C. M. The chirp z-transform algorithm and its application. *Bell Systems Technical Journal* 48, 5 (1969), 1249–1292.
- Rebonato, R., and Cooper, I. Coupling backward induction with monte carlo simulations: A fast Fourier transform (FFT) approach. *Applied Mathematical Finance* 5, 2 (1998), 131–141.
- Temperton, C. A generalized prime factor FFT algorithm for any  $n = 2^p 3^q 5^s$ . *SIAM Journal on Scientific and Statistical Computing* 13 (1992), 676–686.