ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΕΦΑΡΜΟΣΜΕΝΩΝ ΜΑΘΗΜΑΤΙΚΩΝ ΚΑΙ ΦΥΣΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

Ακαδημαϊκό Έτος 2024-2025

Multilevel Monte Carlo και Εφαρμογές

Αζάς Λεωνίδας (ge20117)
Ζώρζος Αχιλλέας (ge20026)

Επιβλέπων Καθηγητής: κ. Σωτήρης Σαμπάνης

Περιεχόμενα

1 Εισαγωγή

Η μέθοδος Monte Carlo είναι μια στοχαστική τεχνική προσομοίωσης που χρησιμοποιεί τυχαία δείγματα για την προσέγγιση αριθμητικών λύσεων σε προβλήματα που ενδέχεται να είναι αδύνατον ή πολύπλοκο να επιλυθούν αναλυτικά. Η μέθοδος βασίζεται στην ιδέα ότι, μέσω μεγάλης κλίμακας τυχαίων δειγματοληψιών, μπορεί κανείς να εκτιμήσει ιδιότητες κατανομών, ολοκληρώματα και λύσεις διαφορικών εξισώσεων [Metropolis and Ulam, 1949].

Η προέλευση της μεθόδου Monte Carlo συνδέεται με το Manhattan Project τη δεκαετία του 1940, όπου οι John von Neumann, Stanislaw Ulam και Nicholas Metropolis την ανέπτυξαν για τη μελέτη της διάδοσης των νετρονίων στην πυρηνική σχάση. Η ονομασία "Monte Carlo" δόθηκε ως αναφορά στο διάσημο καζίνο του Μονακό, λόγω της σύνδεσης της μεθόδου με την τυχαιότητα και τα τυχερά παιχνίδια [Metropolis, 1987].

Η σύνδεση της αυτή είναι εμφανής και στο άρθρο του Ελληνο-Αμερικανού καθηγητή Νικόλα Μητρόπουλου, όπου αναφέρεται η ασχολία των επιστημόνων του Los Alamos με τυχερά παιχνίδια όπως πόκερ, πασιέντζα και άλλα.

(a) Nicholas Metropolis (b) John von Neumann (c) Stanislaw Ulam

Figure 1: Τρεις σημαντικοί επιστήμονες της υπολογιστικής φυσικής και των Monte Carlo μεθόδων.

Αξίζει να αναφερθεί και η συνεισφορά του πρώτου ηλεκτρονικού υπολογιστή, με ονομασία ENIAC, στο Πανεπιστήμιο της Πενσυλβανίας στην Φιλαδέλφεια. Η ομάδα του Manhattan Project, εντυπωσιασμένη με την ταχύτητα και υπολογιστική δυνατότητα του ENIAC για την εποχή, εμπνεύστηκαν για να τον χρησιμοποιήσουν για τεχνικές στατιστικής δειγματοληψίας που είχαν «αχρηστευτεί», λόγω του μήκους και της μονοτονίας των υπολογισμών.

Από την πρώτη της εφαρμογή, η μέθοδος Monte Carlo έχει εξελιχθεί και έχει βρει εφαρμογές σε πολλούς τομείς, όπως η φυσική, η οικονομία, η μηχανική, και η τεχνητή νοημοσύνη. Σήμερα, η χρήση της έχει επεκταθεί με σύγχρονες παραλλαγές, όπως η Multilevel Monte Carlo (MLMC), που επιτρέπουν τη μείωση της υπολογιστικής πολυπλοκότητας, διατηρώντας παράλληλα την υψηλή ακρίβεια των προσομοιώσεων [Giles, 2018].

Στην εργασία αυτή, θα παρουσιάσουμε το υπόβαθρο και τον τρόπο εφαρμογής της μεθόδου για Option Evaluation μέσα από τα προγραμματιστικά περιβάλλοντα MATLAB και Python, ενώ θα συγκρίνουμε τα αποτελέσματα της με την εξέλιξη της, Multi Level Monte Carlo, και θα παρουσιάσουμε πως μπορεί να αποτελέσει μια αποδοτικότερη λύση στο εκάστοτε πρόβλημα.

2 Δικαιώματα (Options)

Τα options αφορούν συμβόλαια μεταξύ δύο μελών, του κατόχου του δικαιώματος (Option Holder) και του εκδότη του δικαιώματος (Option Issuer), που υπογράφεται σε κάποιο χρόνο t, από το οποίο προκύπτει το δικαίωμα, αλλά όχι η υποχρέωση, (για τον Holder) να αγοραστεί ένα βασικό αγαθό S (underlying asset) με τιμή St σε κάποιο μελλοντικό χρόνο $T > t$. Ανάλογα με το είδος του Option, αλλάζει το πότε μπορεί να ασκηθεί το δικαίωμα αυτό από τον Holder, καθώς και η προκαθορισμένη τιμή που συμφωνείται ανάμεσα στα δύο μέλη. Το πιο απλό δικαίωμα αγοράς αποτελεί το European Call Option [Hull, 2017].

Ένα Ευρωπαϊκό δικαίωμα αγοράς (European Call Option) αποτελεί ένα συμβόλαιο στο οποίο ο Holder έχει δικαίωμα να ασκήσει το δικαίωμα του μόνο σε μια προκαθορισμένη χρονική στιγμή T (expiry date). Η τιμή στην οποία αγοράζεται το underlying asset είναι και αυτή προκαθορισμένη και συνήθως συμβολίζεται με K (strike price).

Ο Holder έχοντας το δικαίωμα αυτό μπορεί να πράξει με δύο τρόπους:

α) Εάν η πραγματική τιμή του Asset St είναι μεγαλύτερη από την προκαθορισμένη τιμή K στον χρόνο T, τότε έχει πλεονέκτημα να ασκήσει το δικαίωμα αυτό και να πουλήσει απευθείας το Asset, βγάζοντας κέρδος $St - K$.

β) Εάν η τιμή του Asset St είναι μικρότερη από την προκαθορισμένη τιμή K, τότε ο Holder δεν έχει λόγο να ασκήσει τον δικαίωμα, αφού δεν θα βγάλει κέρδος από την πώληση του Asset, αλλά αντιθέτως θα βγάλει ζημιά αφού $St–K < 0$.

Ο Holder στις δυο περιπτώσεις είτε βγάζει κέρδος (περίπτωση α), είτε μένει άπραγος και δεν χάνει ή βγάζει χρήματα (περίπτωση β). Αντιθέτως, ο Issuer στον χρόνο T δεν θα βγάλει καθόλου χρήματα στο expiry date, ενώ μπορεί να χάσει ένα απεριόριστο αριθμό χρημάτων εάν αναγκαστεί να πουλήσει ένα αγαθό σε μία τιμή που δεν τον συμφέρει (συγκρίνοντας την με την πραγματική τιμή του κατά την χρονική στιγμή T). Σημειώνεται εδώ, ότι ο Issuer είναι υποχρεωμένος να πουλήσει στην περίπτωση που ο Holder αποφασίσει να ασκήσει το Option. Για να αντισταθμιστεί η ανισορροπία αυτή, όταν υπογράφεται το Option, ο Holder δίνει ένα ποσό χρημάτων στον Issuer, γνωστό και ως τιμή (value) του Option.

Το ζήτημα που μελετούμε σε αυτήν την εργασία είναι η εκτίμηση του ποσού αυτού, χρησιμοποιώντας αριθμητικές μεθόδους και αλγόριθμους, πιο συγκεκριμένα την μέθοδο Monte Carlo.

Ακριβώς με αντίστροφους ρόλους δουλεύουν τα (Ευρωπαϊκά) δικαιώματα πώλησης, ή αλλιώς European Put Options, στα οποία ο Holder έχει το δικαίωμα, αλλά όχι την υποχρέωση, να πουλήσει κάποιο Asset σε μια προκαθορισμένη τιμή σε μια συγκεκριμένη χρονική στιγμή T.

3 Εισαγωγή στο Μοντέλο Black-Scholes

Το μοντέλο Black-Scholes είναι ένα μαθηματικό μοντέλο για την τιμολόγηση ευρωπαϊκών δικαιωμάτων (European-style options). Αναπτύχθηκε από τους Fischer Black και Myron Scholes το 1973, με σημαντικές συνεισφορές από τον Robert Merton. Το μοντέλο παρέχει κλειστή (αναλυτική) λύση για τις τιμές των Ευρωπαϊκών δικαιωμάτων, βασισμένο στην υπόθεση ότι οι τιμές των περιουσιακών στοιχείων ακολουθούν γεωμετρική κίνηση Brown (Geometric Brownian Motion - GBM) με σταθερή μεταβλητότητα και χωρίς δυνατότητες arbitrage στην αγορά [Hull, 2017].

3.1 Υποθέσεις του Μοντέλου Black-Scholes

1. Η τιμή του υποκείμενου περιουσιακού στοιχείου ακολουθεί μια στοχαστική διαδικασία [Highman, 2004]:

$$dS_t = rS_t dt + \sigma S_t dW_t \tag{1}$$

   όπου:

   - S_t είναι η τιμή του περιουσιακού στοιχείου στη χρονική στιγμή t,
   - r είναι ο σταθερός συντελεστής drift (προσδοκώμενη απόδοση),
   - σ είναι η σταθερή μεταβλητότητα,
   - W_t είναι μία τυπική κίνηση Brown (Wiener process).

2. Οι αγορές είναι χωρίς τριβές (δηλαδή δεν υπάρχουν κόστη συναλλαγών ή φόροι).
3. Το επιτόκιο χωρίς κίνδυνο είναι σταθερό και γνωστό.
4. Δεν καταβάλλονται μερίσματα κατά τη διάρκεια ζωής του Option.
5. Το δικαίωμα είναι ευρωπαϊκού τύπου, δηλαδή μπορεί να ασκηθεί μόνο στη λήξη T.
6. Δεν υπάρχουν ευκαιρίες arbitrage.

3.2 Η Μερική Διαφορική Εξίσωση Black-Scholes (PDE)

Κατασκευάζοντας ένα χαρτοφυλάκιο χωρίς κίνδυνο και εξασφαλίζοντας ότι δεν υπάρχει arbitrage, προκύπτει η εξίσωση Black-Scholes συναρτήσει της τιμής του δικαιώματος $f(S,t)$:

$$\frac{\partial f}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 f}{\partial S^2} + rS\frac{\partial f}{\partial S} - rf = 0 \tag{2}$$

Η λύση αυτής της PDE, με τις κατάλληλες οριακές συνθήκες, οδηγεί στην αναλυτική εξίσωση Black-Scholes για την τιμολόγηση ευρωπαϊκών call και put δικαιωμάτων [Hull, 2017].

3.3 Ο Τύπος Black-Scholes για ένα Ευρωπαϊκό Δικαίωμα Πώλησης (Put Option)

Ένα δικαίωμα πώλησης (put option) δίνει στον κάτοχό του το δικαίωμα να πουλήσει ένα περιουσιακό στοιχείο σε προκαθορισμένη τιμή K στη λήξη T. Λύνοντας την εξίσωση Black-Scholes με την οριακή συνθήκη:

$$P(S,T) = \max(K - S_T, 0) \tag{3}$$

προκύπτει η εξίσωση Black-Scholes για ένα ευρωπαϊκό put:

$$P(S_0,T) = Ke^{-rT}N(-d_2) - S_0N(-d_1) \tag{4}$$

όπου:

$$d_1 = \frac{\ln(S_0/K) + (r + \frac{1}{2}\sigma^2)T}{\sigma\sqrt{T}}, \quad d_2 = d_1 - \sigma\sqrt{T} \tag{5}$$

Οι μεταβλητές είναι:

- S_0: Τρέχουσα τιμή μετοχής.
- K: Τιμή εξάσκησης (strike price).
- T: Χρόνος έως τη λήξη.
- r: Επιτόκιο χωρίς κίνδυνο.
- σ: Μεταβλητότητα του υποκείμενου περιουσιακού στοιχείου.
- $N(x)$: Η συσσωρευτική συνάρτηση κατανομής της κανονικής κατανομής (CDF).

Ερμηνεία των Όρων της Εξίσωσης

- $S_0N(-d_1)$: Αντιπροσωπεύει την προσδοκώμενη αξία της πώλησης της μετοχής στη λήξη, προεξοφλημένη με την πιθανότητα ότι το δικαίωμα θα είναι εντός χρημάτων (in the money).
- $Ke^{-rT}N(-d_2)$: Η παρούσα αξία της πληρωμής του strike price K, προεξοφλημένη με την πιθανότητα άσκησης του δικαιώματος πώλησης.

Η εξίσωση αυτή δίνει την ακριβή τιμή ενός ευρωπαϊκού put option, χωρίς την ανάγκη χρήσης Monte Carlo (MC) ή Multilevel Monte Carlo (MLMC).

4 Monte Carlo

Στην πιο άμεση του μορφή, η μέθοδος Monte Carlo αποτελεί μια αρκετά απλή διαδικασία για την εκτίμηση μιας τιμής $a = E[P]$, όπου P έστω τυχαία μεταβλητή [Metropolis and Ulam, 1949]. Η τιμή $E[P]$ προσεγγίζεται υπολογίζοντας τη μέση τιμή των τιμών $P(\omega)$ από ένα σύνολο N ανεξάρτητων και τυχαίων πειραμάτων ω, που προέρχονται από τον χώρο πιθανότητας $(\Omega, \mathcal{F}, P)$. Το πλήθος των πειραμάτων είναι ανάλογο της ακρίβειας που θέλουμε να πετύχουμε στην προσέγγιση. Όσο μεγαλύτερο αριθμό πειραμάτων εκτελέσουμε, τόσο καλύτερα προσεγγίζεται και η πραγματική τιμή που ψάχνουμε [Highman, 2004].

Η εκτίμηση της τιμής a δίνεται από τον τύπο:

$$\hat{P} := \frac{1}{N} \sum_{n=1}^{N} P(\omega^{(n)}) \tag{6}$$

Εύκολα αποδεικνύεται πως η μέση τιμή είναι αμερόληπτη εκτιμήτρια της τιμής a. Συγκεκριμένα:

$$\begin{aligned} E[\hat{P}] &= E\left[\frac{1}{N} \sum_{n=1}^{N} P(\omega^{(n)})\right] \\ &= \frac{1}{N} \sum_{n=1}^{N} E[P(\omega^{(n)})] \\ &= \frac{1}{N} \cdot N \cdot a = a \end{aligned}$$

Ο τύπος της διασποράς $b^2 = Var(P)$ είναι

$$Var[P] = E[(P - E[P])^2] \tag{7}$$

Για την αμερόληπτη εκτίμηση της διασποράς χρησιμοποιείται ο τύπος:

$$\hat{\sigma}^2 := \frac{1}{N-1} \sum_{n=1}^{N} \left(P(\omega^{(n)}) - \hat{P}\right)^2 \tag{8}$$

Από το Κεντρικό Οριακό Θεώρημα, είναι γνωστό ότι για μεγάλο αριθμό πειραμάτων, η κατανομή της εκτίμησης της Monte Carlo προσεγγίζει την κανονική κατανομή:

$$\sum_{n=1}^{N} P(\omega^{(n)}) \sim N(Na, Nb^2) \tag{9}$$

Η τυχαία μεταβλητή $\hat{P} - a$ προσεγγίζει την κατανομή:

$$\hat{P} - a \sim N\left(0, \frac{b^2}{N}\right), \tag{10}$$

που αποτελεί την τυπική κανονική κατανομή N(0,1) μεγεθυμένη κατά $\frac{b}{\sqrt{N}}$.

Από την παραπάνω σχέση, προκύπτει το 95% διάστημα εμπιστοσύνης ως εξής:

$$P\left(\hat{P} - 1.96\frac{b}{\sqrt{N}} \leq a \leq \hat{P} + 1.96\frac{b}{\sqrt{N}}\right) = 0.95 \tag{11}$$

και χρησιμοποιώντας την εκτιμήτρια της διασποράς έχουμε:

$$P\left(\hat{P} - 1.96\frac{\hat{\sigma}}{\sqrt{N}} \leq a \leq \hat{P} + 1.96\frac{\hat{\sigma}}{\sqrt{N}}\right) = 0.95 \tag{12}$$

Από αυτό το διάστημα εμπιστοσύνης φαίνεται πως το εύρος του είναι αντιστρόφως ανάλογο του αριθμού των πειραμάτων N. Επομένως, μεγάλο πλήθος δειγμάτων οδηγεί σε καλύτερη ακρίβεια της προσέγγισης.

Πιο συγκεκριμένα, το εύρος του διαστήματος εμπιστοσύνης είναι:

$$2 \times 1.96\frac{\sigma}{\sqrt{N}}$$

το οποίο είναι αντιστρόφως ανάλογο της ρίζας του αριθμού των δειγμάτων N. Για να μειωθεί το «σφάλμα» κατά έναν παράγοντα 10, απαιτείται εκατονταπλάσια αύξηση του μεγέθους δείγματος. Πρόκειται για έναν σοβαρό περιορισμό που συνήθως καθιστά αδύνατη την επίτευξη πολύ υψηλής ακρίβειας σε μια προσέγγιση Monte Carlo.

4.1 Παράδειγμα *Ζάρια*

Ένα απλό παράδειγμα για να δούμε πώς εφαρμόζεται η μέθοδος Monte Carlo σε τυχαία δείγματα μπορεί να βρεθεί χρησιμοποιώντας δυο κανονικά ζάρια. Έστω το ερώτημα: «Ποια είναι η πιθανότητα το άθροισμα των ζαριών να είναι ίσο με 7;». Είναι προφανές ότι:

- Όλες οι δυνατές ρίψεις δύο ζαριών είναι $6 \times 6 = 36$.
- Οι συνδυασμοί (x, y) που δίνουν άθροισμα 7 είναι: $(1,6), (2,5), (3,4), (4,3), (5,2), (6,1)$. Επομένως, υπάρχουν 6 ενδεχόμενα επιτυχίας.
- Άρα η ζητούμενη πιθανότητα είναι:

$$\frac{6}{36} = \frac{1}{6} = 0.1667$$

Για να υπολογίσουμε, ή πιο σωστά να προσεγγίσουμε, την πιθανότητα αυτή, ακολουθούμε τα εξής βήματα:

1. Ορίζουμε έναν μεγάλο αριθμό προσομοιώσεων/ρίψεων (π.χ. 10.000, 100.000 ή περισσότερες).
2. «Ρίχνουμε» τα δύο ζάρια τυχαία μέσω κάποιας συνάρτησης παραγωγής τυχαίων αριθμών.
3. Μετράμε πόσες φορές προκύπτει το άθροισμα 7.
4. Υπολογίζουμε την εμπειρική πιθανότητα ως:

$$P(S = 7) = \frac{\text{αριθμός ρίψεων με άθροισμα 7}}{\text{συνολικός αριθμός ρίψεων}}$$

Το πρόγραμμα μας επιστρέφει:

- Αριθμός ρίψεων: 1000, Εκτιμώμενη πιθανότητα: 0.1720, Σφάλμα από 1/6: 0.0053
- Αριθμός ρίψεων: 10000, Εκτιμώμενη πιθανότητα: 0.1646, Σφάλμα από 1/6: 0.0021
- Αριθμός ρίψεων: 100000, Εκτιμώμενη πιθανότητα: 0.1652, Σφάλμα από 1/6: 0.0014
- Αριθμός ρίψεων: 1000000, Εκτιμώμενη πιθανότητα: 0.1663, Σφάλμα από 1/6: 0.0004

Σημαντική παρατήρηση είναι η διαφορά μεταξύ της εκτιμώμενης τιμής και του σφάλματος ανάμεσα στα διαφορετικά πλήθη ρίψεων. Είναι εμφανές ότι όσο περισσότερες επαναλήψεις έχει το πείραμα, τόσο καλύτερα προσεγγίζεται η ζητούμενη τιμή.

4.2 Παράδειγμα *European Put Option*

Έχοντας μιλήσει για τα δικαιώματα (options) σε προηγούμενη ενότητα, τώρα θα ασχοληθούμε συγκεκριμένα με ένα παράδειγμα εύρεσης τιμής ενός Ευρωπαϊκού δικαιώματος πώλησης με την χρήση της μεθόδου Monte Carlo.

Όπως αναφέρθηκε και προηγουμένως, το payoff ενός European Put Option βγαίνει από τον τύπο

$$P(S(T)) = \max(E - S(T), 0) \tag{13}$$

Αρχικά θα χρησιμοποιήσουμε την Black-Scholes Formula για να υπολογίσουμε την ακριβή τιμή και έπειτα θα την συγκρίνουμε με τα αποτελέσματα τις μεθόδου Monte Carlo.

Οι παράμετροι για το παράδειγμα μας θα είναι:

$S = 4$	(Αρχική τιμή Asset)
$E = 5$	(Strike price)
$\sigma = 0.3$	(Volatility)
$r = 0.04$	(Risk-free rate)
$T = 1$	(Time to maturity)
$N = 10000$	(Αριθμός προσομοιώσεων)

Η τιμή του Asset στην περίοδο ωρίμανσης T=1 υπολογίζεται από τον τύπο:

$$S_i = S_0 \cdot \exp\left(\left(r - \frac{1}{2}\sigma^2\right)T + \sigma\sqrt{T}Z\right), \quad \text{όπου } Z \sim \mathcal{N}(0,1) \tag{14}$$

Το τελικό Payoff συμπεριλαμβάνει και τον παράγοντα προεξόφλησης και έχουμε:

$$V_i = e^{-rT}\max(E - S_i, 0) \tag{15}$$

Algorithm 1 Monte Carlo προσομοίωση για τιμολόγηση δικαιώματος πώλησης

1: **for** $i = 1$ to N **do**
2: Δείγμα $Z_i \sim \mathcal{N}(0,1)$, Υπολογισμός $S_i = S_0 e^{(r-\frac{1}{2}\sigma^2)T+\sigma\sqrt{T}Z_i}$
3: Υπολογισμός Payoff: $V_i = e^{-rT}\max(E - S_i, 0)$
4: **end for**
5: Εκτίμηση μέσης τιμής: $a_N = \frac{1}{N}\sum_{i=1}^{N} V_i$
6: Εκτίμηση διακύμανσης: $b_N^2 = \frac{1}{N-1}\sum_{i=1}^{N}(V_i - a_N)^2$
7: 95% Διάστημα Εμπιστοσύνης:

$$\text{conf} = \left[a_N - t_{0.95,N-1}\cdot\frac{b_N}{\sqrt{N}}, \quad a_N + t_{0.95,N-1}\cdot\frac{b_N}{\sqrt{N}}\right]$$

Παρακάτω παρατίθενται τα αποτελέσματα σε Matlab και Python:

Μέθοδος	**MATLAB**	**Python**
Ακριβή τιμή Option (Black-Scholes)	1.020686	1.0207
Απλή Monte Carlo Προσομοίωση		
Μέση τιμή	1.024121	1.0332
Τυπική απόκλιση	0.842701	0.8462
95% Διάστημα Εμπιστοσύνης	[1.007604, 1.040638]	(1.0166, 1.0498)
Πλάτος Δ.Ε.	0.0333	0.0332
Μέσος χρόνος εκτέλεσης (sec)	0.000306	0.000330

Table 1: Σύγκριση αποτελεσμάτων Monte Carlo προσομοίωσης σε MATLAB και Python.

Παρατηρούμε πως η μέση τιμή των προσομοιώσεων Monte Carlo προσεγγίζει ικανοποιητικά την ακριβή τιμή του συγκεκριμένου European Put Option.

Τρέχοντας τον ίδιο κώδικα για διαφορετικό αριθμό προσομοιώσεων, είναι εμφανής η σύγκλιση της μεθόδου στην πραγματική τιμή όσο αυξάνεται το N.

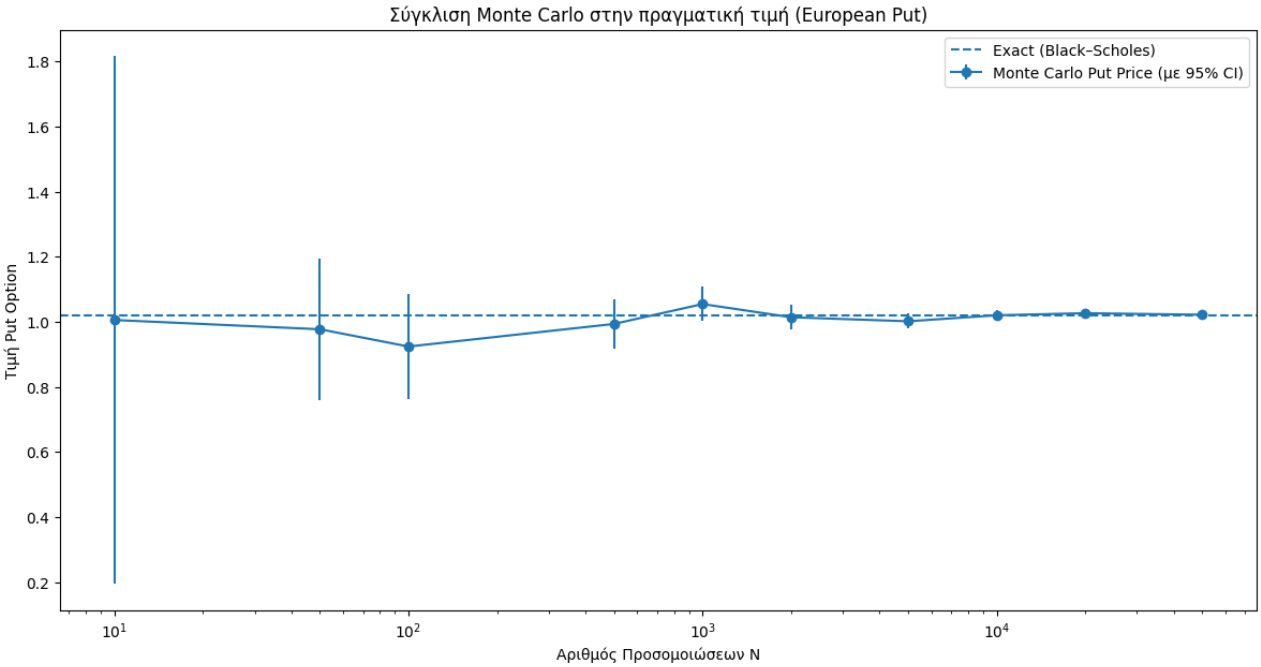

Figure 2: Προσομοίωση Monte Carlo (European Put)

4.3 Αντιθετικές Μεταβλητές

Οι Antithetic Variates (αντιθετικές μεταβλητές) αποτελούν μια διαδεδομένη τεχνική μείωσης διακύμανσης (variance reduction) στις προσομοιώσεις Monte Carlo, που στοχεύει στη βελτίωση της ακρίβειας των αριθμητικών εκτιμήσεων χωρίς να αυξάνει σημαντικά το υπολογιστικό κόστος [Highman, 2004].

Η μέθοδος βασίζεται στη χρήση ζευγαριών τυχαίων μεταβλητών, οι οποίες είναι αρνητικά συσχετισμένες μεταξύ τους. Συγκεκριμένα, αν χρησιμοποιούμε μια τυχαία μεταβλητή

$$Z \sim N(0,1),$$

τότε η αντιθετική της είναι η $-Z$. Επομένως, κατασκευάζονται δύο ποσότητες:

$$Y = f(Z), \quad \widetilde{Y} = f(-Z),$$

όπου f είναι η συνάρτηση του payoff.

Σε εφαρμογές που αφορούν αποτίμηση δικαιωμάτων (*options evaluation*), αυτό συνήθως σημαίνει ότι, για κάθε τροχιά (*sample path*) της στοχαστικής διαδικασίας, προσομοιώνεται ταυτόχρονα η αντίστοιχη *αντιθετική* τροχιά, δημιουργώντας έτσι ένα ζεύγος προσομοιώσεων.

Για τον υπολογισμό της «αντιθετικής» τροχιάς, χρησιμοποιείται ο τύπος:

$$S_T^{\text{ant}} = S_0 \cdot \exp\Big((r - \tfrac{1}{2}\,\sigma^2)\,T \;-\; \sigma\,\sqrt{T}\,Z \Big) \tag{16}$$

Η εκτίμηση της ζητούμενης ποσότητας (τελικό *Payoff*) προκύπτει τότε ως ο μέσος όρος των δύο εκτιμήσεων:

$$\widehat{Y}_{\text{ant}} \;=\; \frac{Y + \widetilde{Y}}{2}.$$

Μαθηματική απόδειξη της μείωσης διακύμανσης

Η τυπική ανάλυση βασίζεται στον τύπο για τη διακύμανση ενός αθροίσματος:

$$Var(X+Y) = Var(X) + Var(Y) + 2\,Cov(X,Y). \tag{17}$$

Στην περίπτωσή μας, με $X = Y$ και $Y = \widetilde{Y}$, η εκτίμηση $\widehat{Y}_{\text{ant}}$ δίνεται από:

$$\widehat{Y}_{\text{ant}} = \frac{Y + \widetilde{Y}}{2}. \tag{18}$$

Η διακύμανση του $\widehat{Y}_{\text{ant}}$ είναι:

$$Var\big(\widehat{Y}_{\text{ant}}\big) = Var\Big(\frac{Y+\widetilde{Y}}{2}\Big) = \frac{1}{4}\,Var(Y+\widetilde{Y}). \tag{19}$$

Στη συνέχεια, εφαρμόζουμε τον τύπο διακύμανσης αθροίσματος:

$$Var(Y+\widetilde{Y}) = Var(Y) + Var(\widetilde{Y}) + 2\,Cov(Y,\widetilde{Y}). \tag{20}$$

Υποθέτοντας πως Y και $\widetilde{Y}$ έχουν την ίδια διασπορά (κάτι εύλογο, αφού $\widetilde{Y} = f(-Z)$ έχει την ίδια κατανομή με το $f(Z)$ σε συμμετρικές κατανομές όπως η $N(0,1)$), παίρνουμε:

$$Var(Y) = Var(\widetilde{Y}) \;\equiv\; \sigma_Y^2. \tag{21}$$

Άρα:

$$Var(Y+\widetilde{Y}) = \sigma_Y^2 + \sigma_Y^2 + 2\,Cov(Y,\widetilde{Y}) = 2\sigma_Y^2 + 2\,Cov(Y,\widetilde{Y}). \tag{22}$$

Συνεπώς,

$$Var\big(\widehat{Y}_{\text{ant}}\big) = \frac{1}{4}\Big[2\sigma_Y^2 + 2\,Cov(Y,\widetilde{Y})\Big] = \frac{1}{2}\,\sigma_Y^2 + \frac{1}{2}\,Cov(Y,\widetilde{Y}). \tag{23}$$

Εάν $Cov(Y,\widetilde{Y}) < 0$, τότε προφανώς $Var(\widehat{Y}_{\text{ant}})$ είναι μικρότερη από σ_Y^2. Μάλιστα, όσο πιο αρνητική είναι η συσχέτιση $Cov(Y,\widetilde{Y})$, τόσο πιο πολύ κερδίζουμε σε μείωση της διακύμανσης.

Η κεντρική ιδέα στα European Options είναι ότι, καθώς το $\exp(\sigma\sqrt{T}\,Z)$ αυξάνεται με το Z, το $\exp(\sigma\sqrt{T}\,(-Z))$ μειώνεται αντίστοιχα. Εάν το payoff $f(\cdot)$ είναι μονότονα αύξουσα συνάρτηση του S_T (που με τη σειρά του μεγαλώνει με το Z), τότε:

- Για μεγάλες τιμές του Z, το $Y = f(S_T)$ τείνει να είναι «μεγάλο», ενώ το $\widetilde{Y} = f(S_T^{(\text{ant})})$ τείνει να είναι «μικρό».
- Για μικρές ή αρνητικές τιμές του Z, συμβαίνει το ανάποδο.

Όταν λοιπόν προσπαθούμε να υπολογίσουμε τη συνδιακύμανση $Cov(Y,\widetilde{Y})$, αυτή η «αντίστροφη κίνηση» οδηγεί σε αρνητικό πρόσημο.

Αυτό έχει ως αποτέλεσμα την αύξηση της στατιστικής ακρίβειας, μείωνοντας το εύρος του διαστήματος εμπιστοσύνης, και επομένως την καλύτερη σύγκλιση στην πραγματική τιμή της εκτιμώμενης ποσότητας, με δεδομένο αριθμό προσομοιώσεων.

Algorithm 2 Monte Carlo προσομοίωση με Antithetic Variate για τιμολόγηση δικαιώματος πώλησης

1: **for** $i = 1$ to N **do**
2: Δείγμα $Z_i \sim \mathcal{N}(0,1)$
3: Υπολογισμός δύο τιμών του υποκείμενου περιουσιακού στοιχείου:

$$S_i^+ = S_0 e^{(r-\frac{1}{2}\sigma^2)T+\sigma\sqrt{T}Z_i}, \quad S_i^- = S_0 e^{(r-\frac{1}{2}\sigma^2)T-\sigma\sqrt{T}Z_i}$$

4: Υπολογισμός δύο payoffs:

$$V_i^+ = e^{-rT}\max(E - S_i^+, 0), \quad V_i^- = e^{-rT}\max(E - S_i^-, 0)$$

5: Υπολογισμός μέσου όρου των δύο payoffs:

$$V_i = \frac{V_i^+ + V_i^-}{2}$$

6: **end for**
7: Εκτίμηση μέσης τιμής:

$$a_N = \frac{\sum_{i=1}^{N/2} V_i}{N}$$

8: Εκτίμηση διακύμανσης:

$$b_N^2 = \frac{1}{N-1}\sum_{i=1}^{N/2}(V_i - a_N)^2$$

9: 95% Διάστημα Εμπιστοσύνης:

$$\text{conf} = \left[a_N - t_{0.95,N-1}\cdot\frac{b_N}{\sqrt{N}}, \quad a_N + t_{0.95,N-1}\cdot\frac{b_N}{\sqrt{N}}\right]$$

Εφαρμόζοντας τα παραπάνω στο ίδιο παράδειγμα με την ενότητα 3.2 καταλήγουμε στα εξής αποτελέσματα:

Υπενθυμίζουμε ότι:

$S = 4$	(Αρχική τιμή Asset)
$E = 5$	(Strike price)
$\sigma = 0.3$	(Volatility)
$r = 0.04$	(Risk-free rate)
$T = 1$	(Time to maturity)
$N = 10000$	(Αριθμός προσομοιώσεων)

Τα αποτελέσματα σε MATLAB και Python είναι τα εξής:

Μέθοδος	**MATLAB**	**Python**
Exact Option Value (Black-Scholes)	1.0207	1.0207
Monte Carlo με Antithetic Variates		
Μέση τιμή	1.020188	1.0201
Τυπική απόκλιση	0.137903	0.1419
95% Διάστημα Εμπιστοσύνης	[1.017485, 1.022891]	(1.0173, 1.0229)
Πλάτος Δ.Ε.	0.0054	0.0056
Μέσος χρόνος εκτέλεσης (sec)	0.000420	0.000396

Table 2: Σύγκριση αποτελεσμάτων Monte Carlo προσομοίωσης με Antithetic Variates σε MATLAB και Python.

Η διαφορά της απλής Monte Carlo με την Antithetic Variates γίνεται εμφανής τόσο στην τυπική απόκλιση (≈ 0.1 σε σχέση με ≈ 0.8 στην απλή Monte Carlo), όσο και στο εύρος του διαστήματος εμπιστοσύνης (≈ 0.0004 σε σχέση με ≈ 0.03), με σχετικά παρόμοιο χρόνο εκτέλεσης.

Η αποδοτικότητα της Antithetic μεθόδου, σε σχέση με την απλή Monte Carlo φαίνεται και στο παρακάτω γράφημα:

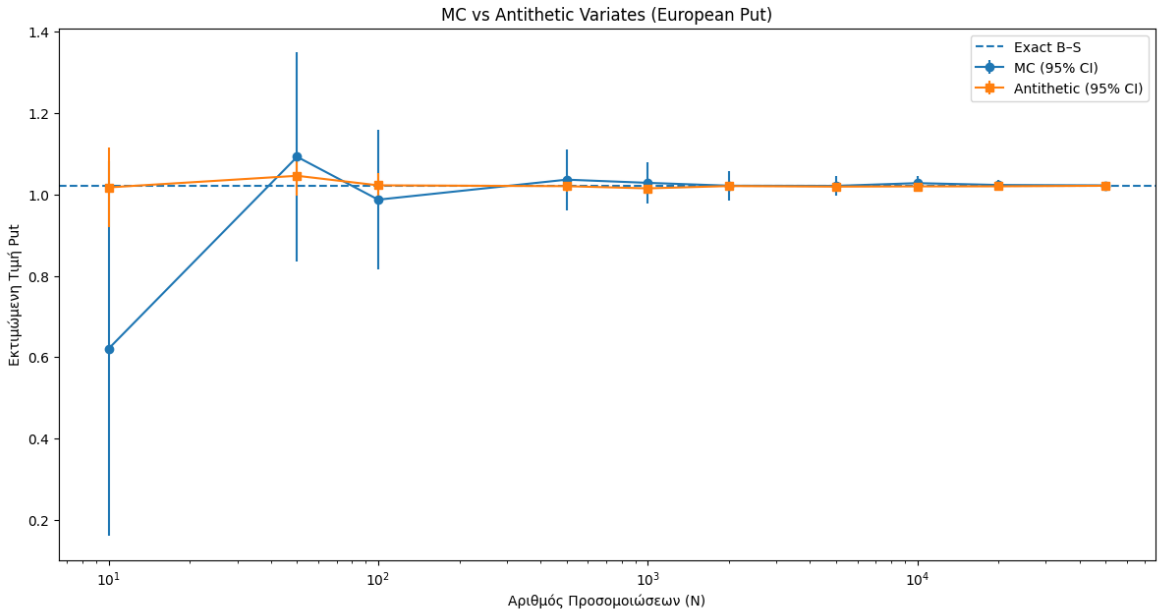

Figure 3: Σύγκριση Monte Carlo με και χωρίς Antithetic Variates για European Put

5 Multilevel Monte Carlo

5.1 Προέλευση της μεθόδου

Η μέθοδος Πολυεπίπεδου Monte Carlo (Multilevel Monte Carlo, MLMC) αποτελεί μια σημαντική εξέλιξη στις αριθμητικές μεθόδους για την εκτίμηση μαθηματικών προσδοκιών σε στοχαστικά συστήματα. Η βασική ιδέα της MLMC είναι η χρήση προσομοιώσεων σε πολλαπλά επίπεδα ακρίβειας, με τις περισσότερες προσομοιώσεις να εκτελούνται σε χαμηλότερη ακρίβεια και χαμηλότερο κόστος, ενώ λιγότερες προσομοιώσεις εκτελούνται σε υψηλότερη ακρίβεια και υψηλότερο κόστος. Αυτή η προσέγγιση μπορεί να μειώσει σημαντικά το συνολικό υπολογιστικό κόστος σε σχέση με τις παραδοσιακές μεθόδους Monte Carlo.

Η πρώτη εφαρμογή της MLMC αποδίδεται στον Mike Giles, ο οποίος το 2008 παρουσίασε τη μέθοδο στο πλαίσιο της προσομοίωσης διαδρομών στοχαστικών διαφορικών εξισώσεων (SDEs) για την τιμολόγηση χρηματοοικονομικών παραγώγων. Στην εκτενή του εργασία "Multilevel Monte Carlo Methods" (2018), ο Giles παρουσιάζει τη θεωρία και τις εφαρμογές της πολυεπίπεδης μεθόδου Monte Carlo, αναλύοντας τη θεμελίωση της μεθόδου στην προσομοίωση διαδρομών στοχαστικών διαφορικών εξισώσεων (SDEs), καθώς και την αποδοτικότητά της ως προς το υπολογιστικό κόστος για δεδομένη ακρίβεια [Giles, 2018].

Ωστόσο, προηγούμενες εργασίες είχαν θέσει τα θεμέλια για την ανάπτυξη της MLMC. Συγκεκριμένα, ο S. Heinrich το 2001 εφάρμοσε μια πολυεπίπεδη μέθοδο Monte Carlo στην παραμετρική ολοκλήρωση, εισάγοντας μια προσέγγιση που μείωσε το υπολογιστικό κόστος μέσω της χρήσης μιας γεωμετρικής ακολουθίας επιπέδων ακρίβειας [Heinrich, 2001].

Η MLMC έχει βρει εφαρμογές σε διάφορους τομείς, όπως η ποσοτικοποίηση αβεβαιότητας σε μερικές διαφορικές εξισώσεις με τυχαίους συντελεστές και η τιμολόγηση χρηματοοικονομικών παραγώγων. Η ευελιξία και η αποδοτικότητα της μεθόδου την καθιστούν ένα ισχυρό εργαλείο για την επίλυση πολύπλοκων στοχαστικών προβλημάτων.

5.2 Monte Carlo με Δύο Επίπεδα

Έστω ότι επιθυμούμε να εκτιμήσουμε την προσδοκώμενη τιμή της τυχαίας μεταβλητής $E[P_1]$, αλλά είναι πιο υπολογιστικά αποδοτικό να εκτιμήσουμε την $E[P_0]$, όπου $P_0 \approx P_1$. Σύμφωνα με την ιδιότητα της γραμμικότητας της προσδοκώμενης τιμής:

$$E[P_1] \;=\; E[P_0] \;+\; E\big[P_1 - P_0\big]. \tag{24}$$

Αυτό επιτρέπει τη χρήση μιας αμερόληπτης εκτιμήτριας δύο επιπέδων:

$$E[P_1] \;\approx\; \frac{1}{N_0}\sum_{n=1}^{N_0} P_0^{(n)} \;+\; \frac{1}{N_1}\sum_{n=1}^{N_1}\big(P_1^{(n)} - P_0^{(n)}\big). \tag{25}$$

Το σημαντικό πλεονέκτημα αυτής της προσέγγισης είναι ότι αν η διαφορά $P_1 - P_0$ είναι μικρή, τότε και η διασπορά της θα είναι μικρή, γεγονός που σημαίνει ότι δεν απαιτείται μεγάλος αριθμός δειγμάτων για να εκτιμηθεί $E[P_1 - P_0]$. Αυτό οδηγεί σε σημαντική μείωση του συνολικού υπολογιστικού κόστους.

5.3 Monte Carlo με Πολλαπλά Επίπεδα

Μία φυσική γενίκευση της παραπάνω προσέγγισης είναι η χρήση πολλαπλών επιπέδων προσεγγίσεων $P_0, P_1, \ldots, P_L$. Η γραμμικότητα της προσδοκώμενης τιμής δίνει:

$$E[P_L] \;=\; E[P_0] \;+\; \sum_{l=1}^{L} E\big[P_l - P_{l-1}\big]. \tag{26}$$

Αυτό οδηγεί στην εξής πολυεπίπεδη εκτιμήτρια:

$$E[P_L] \;\approx\; \frac{1}{N_0}\sum_{n=1}^{N_0} P_0^{(0,n)} \;+\; \sum_{l=1}^{L}\frac{1}{N_l}\sum_{n=1}^{N_l}\big(P_l^{(l,n)} - P_{l-1}^{(l,n)}\big), \tag{27}$$

όπου κάθε επίπεδο εκτιμάται ανεξάρτητα, αξιοποιώντας τη μείωση της διασποράς στα υψηλότερα επίπεδα.

5.4 Διακριτοποίηση Euler-Maruyama

Η διακριτοποίηση Euler-Maruyama αποτελεί μια αριθμητική μέθοδο για την προσέγγιση λύσεων στοχαστικών διαφορικών εξισώσεων (SDEs). Είναι ιδιαίτερα χρήσιμη στις μεθόδους Multilevel Monte Carlo (MLMC), επιτρέποντας την αποδοτική εκτίμηση προσδοκώμενων τιμών σε πολλαπλά επίπεδα.

Χρησιμοποιούμε το μοντέλο Geometric Brownian Motion (GBM) για την περιγραφή της δυναμικής τιμής ενός περιουσιακού στοιχείου:

$$dS_t = rS_t dt + \sigma S_t dW_t \tag{28}$$

Η μέθοδος Euler-Maruyama (EM) διακριτοποιεί αυτήν την εξίσωση πάνω σε ένα χρονικό πλέγμα $\{t_n = nh\}_{n=0}^{N}$ με βήμα χρόνου $h = T/N$. Η αριθμητική προσέγγιση του $S_n \approx S_{t_n}$ δίνεται από:

$$S_{n+1} = S_n + rS_n h + \sigma S_n \Delta W_n \tag{29}$$

όπου:

$$\Delta W_n = W_{t_{n+1}} - W_{t_n} \sim \mathcal{N}(0, h). \tag{30}$$

Στη μέθοδο Multilevel Monte Carlo (MLMC), ορίζουμε διαφορετικά μεγέθη χρονικών βημάτων h_ℓ σε διαφορετικά επίπεδα:

$$h_\ell = 2^{-\ell} T \tag{31}$$

όπου ℓ είναι ο δείκτης επιπέδου. Τα fine επίπεδα (μεγάλο ℓ) έχουν μικρότερα χρονικά βήματα h_ℓ, οδηγώντας σε ακριβέστερες προσεγγίσεις.

5.5 Ανάλυση Υπολογιστικού Κόστους MLMC

Έστω ότι C_0 και C_l είναι τα υπολογιστικά κόστη παραγωγής ενός δείγματος για τα P_0 και $P_l - P_{l-1}$ αντίστοιχα. Το συνολικό υπολογιστικό κόστος είναι:

$$C_{\text{total}} = \sum_{l=0}^{L} N_l C_l. \tag{32}$$

Παράλληλα, έστω ότι V_0 και V_l είναι οι διασπορές για τα P_0 και $P_l - P_{l-1}$ αντίστοιχα. Τότε η συνολική διασπορά δίνεται από:

$$V_{\text{total}} = \sum_{l=0}^{L} \frac{V_l}{N_l}. \tag{33}$$

Σύμφωνα με τον Giles, για βελτιστοποίηση του κόστους με σταθερή διασπορά, εφαρμόζουμε τη μέθοδο των πολλαπλασιαστών Lagrange, χρησιμοποιώντας μ^2 ως πολλαπλασιαστή για την ελαχιστοποίηση του κόστους:

$$\frac{\partial}{\partial N_l} \sum_{k=0}^{L} \left(N_k C_k + \mu^2 \frac{V_k}{N_k} \right) = 0. \tag{34}$$

Λύνοντας ως προς N_l, προκύπτει:

$$N_l = \mu \sqrt{\frac{V_l}{C_l}}, \tag{35}$$

και άρα:

$$N_l C_l = \mu \sqrt{V_l C_l}. \tag{36}$$

Για συνολική διασπορά ε^2, ο συντελεστής μ ικανοποιεί:

$$\mu = \varepsilon^{-2} \sum_{l=0}^{L} \sqrt{V_l C_l}, \tag{37}$$

οπότε το συνολικό κόστος γίνεται:

$$C_{\text{total}} = \varepsilon^{-2} \left(\sum_{l=0}^{L} \sqrt{V_l C_l} \right)^2. \tag{38}$$

5.6 Σύγκριση με Standard Monte Carlo

Το κόστος του κλασικού Monte Carlo είναι:

$$C_{\text{MC}} = \varepsilon^{-2} V_0 C_L. \tag{39}$$

Η εξοικονόμηση κόστους μέσω του MLMC δίνεται από δύο περιπτώσεις:
- Όταν $\sqrt{V_l C_l}$ αυξάνεται με το επίπεδο:

$$\frac{MLMC}{MC} = \frac{V_L}{V_0}. \tag{40}$$

- Όταν $\sqrt{V_l C_l}$ μειώνεται με το επίπεδο:

$$\frac{MLMC}{MC} = \frac{C_0}{C_L}. \tag{41}$$

Αυτές οι δύο περιπτώσεις δείχνουν πως γίνεται η σύγκριση του υπολογιστικού κόστους μεταξύ των δύο μεθόδων αναλόγως την αύξηση ή μείωση του $\sqrt{V_l C_l}$.

5.7 Θεώρημα Πολυπλοκότητας MLMC

Προτού μιλήσουμε για το θεώρημα πολυπλοκότητας του M. Giles, ας θυμηθούμε ότι στην πράξη είναι αδύνατο να προσεγγίσουμε με απόλυτη ακρίβεια την τυχαία μεταβλητή P. Στη γενική μέθοδο MLMC παίρνουμε ως αποτέλεσμα τη μεταβλητή P_L του υψηλότερου επιπέδου που αντιστοιχεί στη μεταβλητή ενδιαφέροντος P.

Έστω Y προσέγγιση του $E[P]$ και έχουμε το μέσο τετραγωνικό σφάλμα (MSE) ως:

$$MSE = E[(Y - E[P])^2] = V[Y] + (E[Y] - E[P])^2. \tag{42}$$

Έστω επίσης ότι $\hat{Y}$ είναι ο εκτιμητής της πολυεπίπεδης μεθόδου:

$$\hat{Y} = \sum_{l=0}^{L} Y_l, \tag{43}$$

με

$$Y_l = \frac{1}{N_l} \sum_{n=1}^{N_l} (P_l^{(l,n)} - P_{l-1}^{(l,n)}). \tag{44}$$

Τότε,

$$E[\hat{Y}] = E[P_L], \tag{45}$$

$$Var[\hat{Y}] = \sum_{l=0}^{L} \frac{V_l}{N_l}, \tag{46}$$

$$V_l = Var[P_l - P_{l-1}]. \tag{47}$$

Από τα παραπάνω βλέπουμε ότι το MSE μπορεί να γραφτεί ως:

$$MSE = E[(\hat{Y} - E[P])^2] = Var[\hat{Y}] + (E[P_L - P])^2. \tag{48}$$

Για να διασφαλίσουμε ότι το MSE είναι μικρότερο από ε^2, αρκεί να διασφαλίσουμε ότι $Var[\hat{Y}]$ και $(E[P_L - P])^2$ είναι και τα δύο μικρότερα από $\frac{1}{2}\varepsilon^2$.

Συνδυάζοντας αυτήν την ιδέα με μια γεωμετρική ακολουθία επιπέδων, στην οποία το κόστος αυξάνεται εκθετικά με το επίπεδο, ενώ τόσο το weak error $E[P_l - P]$ όσο και το multilevel correction variance V_l μειώνονται εκθετικά, οδηγούμαστε στο παρακάτω θεώρημα:

Theorem 1. Έστω P ένα τυχαίο μέγεθος, και P_ℓ η αριθμητική προσέγγιση του επιπέδου ℓ. Αν υπάρχουν ανεξάρτητοι εκτιμητές Y_ℓ βασισμένοι σε N_l δείγματα Monte Carlo, καθένα από τα οποία έχει αναμενόμενο κόστος C_ℓ και διασπορά V_ℓ, και αν υπάρχουν θετικές σταθερές $\alpha, \beta, \gamma, c_1, c_2, c_3$ τέτοιες ώστε:

$$E[|P_\ell - P|] \leq c_1 2^{-\alpha \ell}, \tag{49}$$

$$V_\ell \leq c_2 2^{-\beta \ell}, \tag{50}$$

$$C_\ell \leq c_3 2^{\gamma \ell}, \tag{51}$$

τότε υπάρχει μια σταθερά c_4 ώστε:

$$C \leq \begin{cases} c_4\varepsilon^{-2}, & \text{αν } \beta > \gamma, \\ c_4\varepsilon^{-2}(\log\varepsilon)^2, & \text{αν } \beta = \gamma, \\ c_4\varepsilon^{-2-(\gamma-\beta)/\alpha}, & \text{αν } \beta < \gamma. \end{cases}$$

Σύγκλιση και Υπολογιστικό Κόστος στην MLMC

Βλέπουμε ότι η παράμετρος α είναι η τάξη ασθενούς σύγκλισης (weak convergence order). Λόγω της υπάρχουσας βιβλιογραφίας για την ασθενή σύγκλιση (weak convergence), συνήθως μπορούμε να προσδιορίσουμε τη σωστή τάξη όταν εφαρμόζουμε το θεώρημα σε διαφορετικά πλαίσια.

Η κατασκευή εκτιμητών (estimators) που ικανοποιούν τις δύο συνθήκες ii. και iv. είναι επίσης σχετικά απλή. Η παράμετρος β στη συνθήκη iii. είναι ο ρυθμός ισχυρής σύγκλισης (strong convergence order). Αυτό δείχνει ότι, σε αντίθεση με την κλασική μέθοδο Monte Carlo, όπου χρησιμοποιείται μόνο η ασθενής σύγκλιση, η MLMC χρησιμοποιεί τόσο ασθενή όσο και ισχυρή σύγκλιση.

Αυτό συνεπάγεται ότι τα αριθμητικά σχήματα με υψηλότερο strong convergence order β θα δώσουν καλύτερα αποτελέσματα όταν εφαρμόζουμε την MLMC. Στην πράξη, η κύρια πρόκληση βρίσκεται στο να βρεθεί και να αποδειχθεί η κατάλληλη τιμή του εκθέτη β για χρήση στη συνθήκη iii.

Εκτίμηση Σταθερών και Χρήση Εναλλακτικών Εκτιμητών

Αξίζει να σημειωθεί ότι στην πράξη δεν γνωρίζουμε τις σταθερές c_1, c_2, c_3 που υποτίθεται ότι είναι γνωστές στο θεώρημα. Στην πραγματικότητα, σχεδόν ποτέ δεν γνωρίζουμε τα c_1, c_2, πράγμα που σημαίνει ότι πρέπει να εκτιμηθούν εμπειρικά από τις εκτιμήσεις του ασθενούς σφάλματος (weak error) και της διασποράς της multilevel διόρθωσης (multilevel correction variance).

Αυτό που δεν είναι αμέσως προφανές όταν κοιτάμε το complexity theorem είναι ότι επιτρέπει και άλλους εκτιμητές πέρα από τον «φυσικό» multilevel estimator, όπου χρησιμοποιούμε το ίδιο αριθμητικό σχήμα διακριτοποίησης (numerical discretization scheme) τόσο για τα coarse paths όσο και για τα fine paths.

Μπορούν να χρησιμοποιηθούν άλλοι εκτιμητές, αρκεί να ικανοποιούν τη συνθήκη ii, αφού αυτό διασφαλίζει ότι $E[\hat{Y}] = E[P_L]$. Για παράδειγμα, μπορούμε να χρησιμοποιήσουμε τον εκτιμητή:

$$\hat{Y}_\ell = \frac{1}{N_\ell}\sum_{i=1}^{N_\ell}(P_\ell^f - P_{\ell-1}^c), \tag{52}$$

όπου τα superscripts f και c χρησιμοποιούνται για να τονίσουν ότι εφαρμόσαμε διαφορετικές προσεγγίσεις για τα fine και τα coarse paths, αντίστοιχα.

Το βασικό σημείο είναι πως, εφόσον διατηρείται η ταυτότητα

$$E[P_\ell^f] = E[P_\ell^c], \tag{53}$$

μπορούμε να χρησιμοποιούμε δύο διαφορετικές προσεγγίσεις για τα coarse και τα fine paths στην προσομοίωση του SDE και παρόλα αυτά να διατηρούμε τη θεμελιώδη ταυτότητα $E[\hat{Y}] = E[P_L]$.

Ανάλυση Πολυπλοκότητας

Από το θεώρημα έχουμε ότι

$$V_l = O(2^{-\beta l}), \quad C_l = O(2^{\gamma l}), \tag{54}$$

το οποίο μας δίνει

$$N_l \sim \left(\frac{2^{-\beta l}}{2^{\gamma l}}\right)^{1/2} = 2^{-(\beta+\gamma)l/2}. \tag{55}$$

Το υπολογιστικό κόστος για το επίπεδο l δίνεται ως

$$N_l \cdot C_l \sim \left(2^{-(\beta+\gamma)l/2} \cdot 2^{\gamma l}\right) = 2^{(\gamma-\beta)l/2}. \tag{56}$$

Ακολουθώντας τον Giles μπορούμε να διατυπώσουμε τα εξής:

- Από τα παραπάνω είναι σαφές ότι, εάν $\beta > \gamma$, το κυρίαρχο υπολογιστικό κόστος θα βρίσκεται στα πιο χαμηλά επίπεδα. Εδώ το κόστος είναι $C_l = O(1)$ και απαιτούνται $O(\varepsilon^{-2})$ δείγματα για την επίτευξη της επιθυμητής ακρίβειας.

- Εάν $\beta < \gamma$, το κυρίαρχο κόστος θα βρίσκεται στα πιο υψηλά επίπεδα, όπου, λόγω της συνθήκης (i), όπου $2^{-\alpha L} = O(\varepsilon)$, έχουμε συνεπώς:

$$C_L = O(\varepsilon^{-\gamma/\alpha}). \tag{57}$$

- Εάν $\beta = 2\alpha$, το συνολικό κόστος είναι $O(C_L)$, που αντιστοιχεί σε $O(1)$ δείγματα στο πιο υψηλό επίπεδο, το οποίο είναι το καλύτερο που μπορεί να επιτευχθεί.

- Τέλος, εάν $\beta = \gamma$, οι συνεισφορές τόσο στη διακύμανση όσο και στην υπολογιστική προσπάθεια κατανέμονται περίπου ομοιόμορφα σε όλα τα επίπεδα l.

6 Υλοποίηση της MLMC σε MATLAB και Python

Για την υλοποίηση της MLMC χρησιμοποιούμε του κώδικες του Giles (*mlmc, mlmc_fn, mlmc_test*). Υπενθυμίζουμε ότι το παράδειγμα μας αποτελεί ένα *European Put Option* με τις εξής παραμέτρους:

$S = 4$	(Αρχική τιμή Asset)
$E = 5$	(Strike price)
$\sigma = 0.3$	(Volatility)
$r = 0.04$	(Risk-free rate)
$T = 1$	(Time to maturity)

Επιπλέον, για την MLMC χρειάζεται να ορίσουμε:

$N_0 = 1000$	(αρχικός αριθμός δειγμάτων ανά επίπεδο)
$L_{\min} = 2$	(ελάχιστο επίπεδο)
$L_{\max} = 6$	(μέγιστο επίπεδο)
$N = 20000$	(αριθμός δειγμάτων για τα convergence tests)
$L = 5$	(επίπεδο για τα convergence tests)
$\text{Eps} = [0.005, 0.01, 0.02, 0.05, 0.1]$	(διάνυσμα τιμών ε για τις δοκιμές)
$M = 4$	(Time step refinement factor)
$n_f = M^L$	(Fine level: αριθμός χρονικών βημάτων)
$n_c = \frac{n_f}{M}$	(Coarse level: αριθμός χρονικών βημάτων)
$h_f = \frac{T}{n_f}$	(Fine level time step)
$h_c = \frac{T}{n_c}$	(Coarse level time step)

Algorithm 3 MLMC Level Function for European Put Option (`European_put_l`)

Είσοδος: επίπεδο l, πλήθος δειγμάτων N
Ορισμός σταθερών: $M = 4$, $T = 1$, $r = 0.04$, $\sigma = 0.3$, $E = 5$, $S = 4$
Υπολογισμός $n_f = M^l$, $n_c = n_f/M$, $h_f = T/n_f$, $h_c = T/n_c$
Αρχικοποίηση: $sums[1:6] = 0$
for κάθε παρτίδα N_2 μέχρι N (batch size 10000) **do**
    Δημιουργία X_f, X_c με αρχική τιμή S
    if $l = 0$ **then**
        $X_f \leftarrow X_f + rX_f h_f + \sigma X_f \Delta W_f$ με τυχαία dW_f
    else
        for κάθε χρονικό βήμα $n = 1$ έως n_c **do**
            Διάσπαση του βήματος σε $M(=4)$ βήματα για το λεπτό επίπεδο
            Ενημέρωση X_f και X_c με αντίστοιχα ΔW
        end for
    end if
    Υπολογισμός πληρωμών:
    $P_f = e^{-rT}\max(E - X_f, 0)$
    $P_c = e^{-rT}\max(E - X_c, 0)$ αν $l > 0$, αλλιώς $P_c = 0$
    Ενημέρωση αθροισμάτων:
    $sums[1] \mathrel{+}= \sum(P_f - P_c)$
    $sums[2] \mathrel{+}= \sum(P_f - P_c)^2$
    $sums[5] \mathrel{+}= \sum P_f$
    $sums[6] \mathrel{+}= \sum P_f^2$
end for
Έξοδος: $(sums, \text{cost} = N \cdot (n_f + n_c))$

Algorithm 4 European Put Option MLMC Main Routine

1: **Ορισμός Παραμέτρων:** $S = 4$, $E = 5$, $\sigma = 0.3$, $r = 0.04$, $T = 1$
2: **Αρχικοποίηση:** $N_0 = 1000$, $L_{\min} = 2$, $L_{\max} = 6$
3: Ορισμός δειγμάτων σύγκλισης $N = 20000$, επιπέδων $L = 5$
4: Ορισμός ακριβείας: $Eps = [0.005, 0.01, 0.02, 0.05, 0.1]$
5: Κλήση `mlmc_test` με ορίσματα:
`mlmc_test(@European_put_l, N, L, N0, Eps, Lmin, Lmax, fp)`
6: Υπολογισμός ακριβούς τιμής με τύπο Black-Scholes:
$d_1 = \frac{\log(S/E)+(r+\sigma^2/2)T}{\sigma\sqrt{T}}$
$d_2 = d_1 - \sigma\sqrt{T}$
$P_{\text{BS}} = Ee^{-rT}\Phi(-d_2) - S\Phi(-d_1)$
7: Κλήση `mlmc_plot` για την απεικόνιση αποτελεσμάτων

Σχόλια

Ο αλγόριθμος δέχεται δύο βασικές παραμέτρους: το επίπεδο διακριτοποίησης l και τον αριθμό δειγμάτων N. Στην αρχή ορίζονται οι βασικές σταθερές του προβλήματος, όπως η μεταβλητότητα σ, το επιτόκιο χωρίς ρίσκο r, ο χρονικός ορίζοντας T, η τιμή εξάσκησης E, η αρχική τιμή της μετοχής S, καθώς και ο παράγοντας refinement M, δηλαδή πόσες χρονικές υποδιαιρέσεις περιλαμβάνει το fine επίπεδο ανά βήμα του coarse επιπέδου. Από αυτές τις τιμές προκύπτει ο αριθμός χρονικών βημάτων του κάθε επιπέδου και τα αντίστοιχα χρονικά βήματα h_f και h_c.

Στη συνέχεια, ο αλγόριθμος εκτελείται σε παρτίδες των 10.000 δειγμάτων για λόγους αποδοτικής διαχείρισης μνήμης. Για κάθε παρτίδα, αρχικοποιούνται τα διανύσματα των τιμών X_f και X_c, τα οποία αντιπροσωπεύουν την τιμή της μετοχής στο τέλος της προσομοίωσης για το fine και το coarse επίπεδο αντίστοιχα. Αν βρισκόμαστε στο επίπεδο $l = 0$, δεν υπάρχει coarse προσομοίωση και γίνεται μόνο ένα χρονικό βήμα για το fine επίπεδο. Αντίθετα, για επίπεδα $l > 0$, η εξέλιξη του X_f γίνεται μέσω $n_c \cdot M$ fine βημάτων, και το X_c ενημερώνεται σε κάθε βήμα με τη χρήση του αθροίσματος των αντίστοιχων fine τυχαίων διακυμάνσεων. Με αυτόν τον τρόπο εξασφαλίζεται ότι τα μονοπάτια fine και coarse είναι συσχετισμένα, γεγονός απαραίτητο για τη μείωση της διασποράς των διαφορών $P_f - P_c$, που αποτελεί τον βασικό στόχο του MLMC.

Αφού υπολογιστούν οι τελικές τιμές X_f και X_c, υπολογίζονται οι αποπληρωμές (payoffs) του ευρωπαϊκού put ως

$$P_f = e^{-rT}\max(E - X_f, 0), \quad P_c = e^{-rT}\max(E - X_c, 0).$$

Για το επίπεδο $l = 0$, όπου δεν υπάρχει coarse προσομοίωση, τίθεται $P_c = 0$.

Στο τέλος κάθε παρτίδας, ενημερώνονται τέσσερα βασικά στατιστικά αθροίσματα:

- $\sum(P_f - P_c)$, που χρησιμοποιείται για την εκτίμηση του μέσου όρου των διαφορών,
- $\sum(P_f - P_c)^2$, για τον υπολογισμό της διασποράς των διαφορών,
- $\sum P_f$, για έλεγχο του πραγματικού μέσου όρου των payoffs,
- $\sum P_f^2$, για υπολογισμό της διασποράς των payoffs.

Τέλος, ο αλγόριθμος επιστρέφει το συνολικό υπολογιστικό κόστος, το οποίο ισούται με το πλήθος των χρονικών βημάτων που απαιτούνται για κάθε δείγμα σε κάθε επίπεδο: $N \cdot (n_f + n_c)$.

6.1 Εκτίμηση παραμέτρων α, β, γ

Παράμετρος	**Python**	**MATLAB**
α	2.070274	3.435871
β	2.019037	1.925267
γ	2.000000	2.000000

Table 3: Σύγκριση των εκτιμώμενων παραμέτρων α, β, γ μεταξύ Python και MATLAB.

Από αυτόν τον πίνακα φαίνεται ότι υπάρχει μια διαφορά κυρίως στην παράμετρο a. Είναι πιθανό αυτό να οφείλεται σε διαφορετικές υλοποιήσεις, ελαφρώς διαφορετικά seed τυχαίων δειγμάτων για τις τυχαίες μεταβλητές, ή σε διαφορετικό χειρισμό της στοχαστικής μεθόδου στα δύο προγραμματιστικά περιβάλλοντα (π.χ. μικροδιαφορές στη μέθοδο Euler-Maruyama, στα rounding errors κλπ.). Η παράμετρος β είναι κοντά μεταξύ των δύο εκδόσεων (2.02 vs 1.93), ενώ η γ (που σχετίζεται με το κόστος ανά επίπεδο) είναι σταθερά 2 και στις δύο.

Είναι σημαντικό να αναφέρουμε πως οι παράμετροι α, β, γ προσεγγίζονται χρησιμοποιώντας γραμμική παλινδρόμηση στον κώδικα του Giles.

6.2 Έλεγχοι Σύγκλισης (Convergence Tests)

Table 4: Αποτελέσματα MLMC για την Python

l	**ave**$(P_f - P_c)$	**ave**(P_f)	**var**$(P_f - P_c)$	**var**(P_f)	**Cost**
0	0.963 45	0.963 45	0.8417	0.8417	1.00
1	0.039 88	1.0010	0.021 94	0.7425	5.00
2	0.007 42	1.0217	0.007 66	0.7243	20.00
3	0.001 55	1.0191	0.001 88	0.7158	80.00
4	0.000 43	1.0242	0.000 47	0.7102	320.00
5	0.000 095	1.0173	0.000 11	0.7109	1280.00

Table 5: Αποτελέσματα MLMC για το MATLAB

l	$\mathbf{ave}(P_f - P_c)$	$\mathbf{ave}(P_f)$	$\mathbf{var}(P_f - P_c)$	$\mathbf{var}(P_f)$	**Cost**
0	0.967 25	0.967 25	0.8530	0.8530	1.25
1	0.037 44	1.0135	0.023 23	0.7653	5.00
2	0.007 19	1.0119	0.007 37	0.7287	20.00
3	0.000 94	1.0103	0.001 85	0.7214	80.00
4	0.000 32	1.0223	0.000 45	0.7205	320.00
5	−0.000 001	1.0116	0.000 12	0.7134	1280.00

Ανάλυση των Αποτελεσμάτων MLMC

Τα αποτελέσματα της πολυεπίπεδης μεθόδου Monte Carlo (MLMC) παρουσιάζουν ενδιαφέροντα χαρακτηριστικά καθώς προχωράμε σε υψηλότερα επίπεδα l. Παρακάτω αναλύουμε τα βασικά ευρήματα που προκύπτουν από τους πίνακες αποτελεσμάτων.

Α) Μείωση της διαφοράς $\mathbf{ave}(P_f - P_c)$ στα υψηλότερα επίπεδα

Παρατηρούμε ότι όσο αυξάνεται το επίπεδο l, η διαφορά μεταξύ των εκτιμήσεων P_f (fine level) και P_c (coarse level) μειώνεται σημαντικά. Στα πρώτα επίπεδα, η διαφορά είναι αρκετά μεγάλη, αλλά από το επίπεδο $l = 3$ και μετά μειώνεται ραγδαία.

Αυτό το φαινόμενο είναι αναμενόμενο, καθώς η MLMC στηρίζεται στην ιδέα ότι οι διαδοχικές εκτιμήσεις γίνονται όλο και πιο ακριβείς, ώστε να συγκλίνουν σταδιακά στην πραγματική τιμή της προσδοκίας. Η σταδιακή μείωση της διαφοράς μεταξύ των επιπέδων αποτελεί έναν από τους βασικούς δείκτες ότι η μέθοδος συγκλίνει σωστά.

Β) Μείωση της διακύμανσης $\mathbf{var}(P_f - P_c)$

Ένα ακόμα σημαντικό εύρημα είναι ότι η διακύμανση της διαφοράς $P_f - P_c$, δηλαδή η ποσότητα $\mathrm{var}(P_f - P_c)$, μειώνεται καθώς προχωράμε σε υψηλότερα επίπεδα. Αυτό δείχνει ότι οι εκτιμήσεις γίνονται σταδιακά πιο σταθερές και ότι η πρόσθετη πληροφορία που προκύπτει από ένα νέο επίπεδο είναι όλο και λιγότερο σημαντική.

Η συμπεριφορά αυτή συμφωνεί πλήρως με τη θεωρία της MLMC, η οποία υποδεικνύει ότι η κύρια συνεισφορά στη συνολική ακρίβεια προέρχεται από τα χαμηλότερα επίπεδα.

Γ) Σταθεροποίηση της μέσης τιμής $\mathbf{ave}(P_f)$

Η μέση τιμή P_f (η εκτίμηση της προσδοκίας της τυχαίας μεταβλητής που προσομοιώνουμε) παραμένει σχετικά σταθερή καθώς αυξάνεται το επίπεδο l. Παρατηρούνται μικρές διακυμάνσεις, αλλά γενικά η τιμή συγκλίνει σε μια σταθερή εκτίμηση.

Αυτό αποτελεί έναν σημαντικό δείκτη ότι η μέθοδος MLMC λειτουργεί σωστά, καθώς υποδηλώνει ότι τα χαμηλά επίπεδα παρέχουν ήδη μια καλή προσέγγιση της πραγματικής προσδοκίας και τα υψηλότερα επίπεδα προσθέτουν μόνο μικρές διορθώσεις.

Δ) Εκθετική αύξηση του κόστους υπολογισμού

Ένα βασικό χαρακτηριστικό της MLMC είναι ότι το υπολογιστικό κόστος αυξάνεται εκθετικά με το επίπεδο l. Από τα αποτελέσματα βλέπουμε ότι το κόστος ανά επίπεδο ακολουθεί την ακολουθία:

$$1, 5, 20, 80, 320, 1280$$

που είναι συμβατή με τη θεωρητική εκτίμηση $\mathcal{O}(M^l)$, όπου M είναι ο παράγοντας υποδιαίρεσης των χρονικών βημάτων (στην περίπτωσή μας $M = 4$).

Η MLMC εκμεταλλεύεται αυτή την ιδιότητα ώστε να επιτυγχάνει υψηλή ακρίβεια με ελεγχόμενο κόστος. Τα χαμηλά επίπεδα απαιτούν μεγάλο αριθμό προσομοιώσεων, ενώ τα υψηλά επίπεδα έχουν λιγότερες προσομοιώσεις, μειώνοντας δραματικά το συνολικό κόστος σε σύγκριση με μια απλή προσομοίωση Monte Carlo.

6.3 Διαφοροποιήσεις στα Επίπεδα Ανοχής ε

Table 6: Αποτελέσματα MLMC Complexity Tests για την Python.

ε	**Value**	**MLMC Cost**	**Std Cost**	**Savings**	N_l **ανά επίπεδο**			
					N_{l1}	N_{l2}	N_{l3}	N_{l4}
0.005	1.0183	218 700	3 033 000	13.87	99 369	7178	2044	532
0.010	1.0175	48 340	189 600	3.92	20 155	1637	1000	–
0.020	1.0318	30 030	47 390	1.58	5027	1000	1000	–
0.050	1.0275	26 000	7583	0.29	1000	1000	1000	–
0.100	0.9735	26 000	1896	0.07	1000	1000	1000	–

Table 7: Αποτελέσματα MLMC Complexity Tests για το MATLAB.

ε	**Value**	**MLMC Cost**	**Std Cost**	**Savings**	N_l **ανά επίπεδο**		
					N_{l1}	N_{l2}	N_{l3}
0.005	1.0210	162 500	777 300	4.78	76 516	6192	1793
0.010	1.0170	52 200	194 300	3.72	19 304	1613	1000
0.020	1.0210	31 160	48 580	1.56	4924	1000	1000
0.050	1.0700	26 250	7773	0.30	1000	1000	1000
0.100	0.9569	26 250	1943	0.07	1000	1000	1000

Ανάλυση των Αποτελεσμάτων των MLMC Complexity Tests

Στους πίνακες παρουσιάζονται τα αποτελέσματα των MLMC Complexity Tests για διαφορετικές ανοχές σφάλματος ε. Τα αποτελέσματα συγκρίνουν το κόστος της πολυεπίπεδης μεθόδου Monte Carlo (MLMC) με το κόστος μιας κλασικής Monte Carlo (Standard MC) προσέγγισης. Παρακάτω αναλύουμε τις βασικές παρατηρήσεις από τους πίνακες.

Α) Μείωση του κόστους με τη χρήση MLMC

Μια από τις σημαντικότερες παρατηρήσεις είναι η μεγάλη διαφορά μεταξύ του κόστους της MLMC (**mlmc_cost**) και του κόστους της απλής Monte Carlo (**std_cost**). Όπως φαίνεται στα αποτελέσματα:

- Για μικρές τιμές ε, όπως $\varepsilon = 0.005$, το κόστος της MLMC είναι πολλαπλάσια μικρότερο σε σύγκριση με την απλή Monte Carlo.
- Ο παράγοντας εξοικονόμησης (**savings**), ο οποίος ορίζεται ως ο λόγος $\frac{\text{std\_cost}}{\text{mlmc\_cost}}$, μειώνεται καθώς αυξάνεται η τιμή της ε.
- Για μικρότερα σφάλματα ($\varepsilon = 0.005$), η εξοικονόμηση είναι ≈ 13.87 (Python) και ≈ 4.78 (MATLAB), δείχνοντας ότι η MLMC είναι σημαντικά πιο αποδοτική.
- Όταν το σφάλμα είναι μεγαλύτερο ($\varepsilon = 0.1$), η εξοικονόμηση μειώνεται σε ≈ 0.07, δείχνοντας ότι η MLMC δεν έχει το ίδιο πλεονέκτημα όταν δεν απαιτείται υψηλή ακρίβεια.

Αυτή η συμπεριφορά είναι απόλυτα συμβατή με τη θεωρία της MLMC, η οποία είναι σχεδιασμένη ώστε να βελτιώνει την ακρίβεια με χαμηλότερο κόστος, ειδικά για πολύ μικρά ε.

Β) Κατανομή των δειγμάτων ανά επίπεδο

Οι πίνακες περιλαμβάνουν επίσης τη κατανομή των αριθμών δειγμάτων N_l στα διάφορα επίπεδα προσομοίωσης της MLMC. Παρατηρούμε ότι:

- Για μικρές τιμές ε (π.χ. $\varepsilon = 0.005$), τα χαμηλά επίπεδα (l_1, l_2) έχουν μεγάλο αριθμό δειγμάτων, ενώ τα υψηλότερα επίπεδα έχουν σαφώς λιγότερα. Αυτό είναι χαρακτηριστικό της MLMC, όπου τα χονδροειδή επίπεδα έχουν περισσότερες προσομοιώσεις επειδή είναι υπολογιστικά φθηνότερα.

- Για μεγαλύτερες τιμές ε (π.χ. $\varepsilon = 0.1$), ο αριθμός δειγμάτων στα διάφορα επίπεδα εξισώνεται, καθώς η απαίτηση ακρίβειας μειώνεται και η MLMC λειτουργεί πιο κοντά σε μια παραδοσιακή Monte Carlo προσέγγιση.

- Ο συνολικός αριθμός δειγμάτων N_l μειώνεται απότομα καθώς προχωράμε σε υψηλότερα επίπεδα, επιβεβαιώνοντας ότι τα υψηλά επίπεδα χρησιμοποιούνται μόνο για μικρές διορθώσεις στην προσδοκία της εκτίμησης.

Αυτό υποδεικνύει ότι η MLMC καταφέρνει να καταμερίσει αποδοτικά τους πόρους της, δίνοντας έμφαση σε επίπεδα όπου το σφάλμα είναι σημαντικότερο και χρησιμοποιώντας λιγότερα δείγματα όπου το σφάλμα είναι ήδη μικρό.

Σύγκριση των αποτελεσμάτων Python και MATLAB

Η σύγκριση των αποτελεσμάτων μεταξύ των δύο υλοποιήσεων (Python και MATLAB) δείχνει ότι:

- Οι εκτιμήσεις της MLMC για την προσδοκία (**value**) είναι παρόμοιες και στις δύο υλοποιήσεις, με μικρές αποκλίσεις λόγω των διαφορών στις μεθόδους υλοποίησης και των αριθμητικών σφαλμάτων.

- Οι τιμές του κόστους διαφέρουν μεταξύ Python και MATLAB, αλλά η γενική τάση (η μείωση του κόστους καθώς αυξάνεται ε) παραμένει η ίδια.

- Οι αριθμοί δειγμάτων N_l είναι επίσης παρόμοιοι, αλλά μικρές διαφορές μπορεί να οφείλονται σε διαφορετική διαχείριση των τυχαίων αριθμών ή στη διαφορετική υπολογιστική απόδοση των δύο γλωσσών.

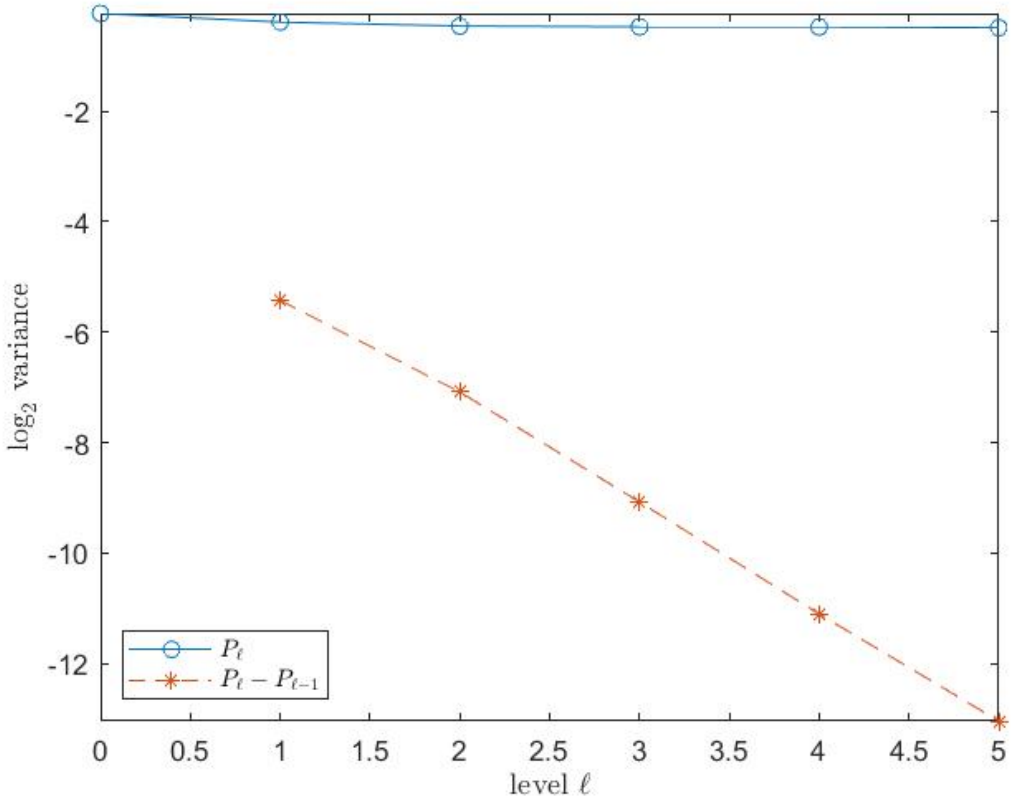

Figure 4: Variance per Level

1) Διάγραμμα $\log_2$(variance) έναντι ℓ

Εδώ παρατίθενται δύο καμπύλες: (α) η $\log_2$(variance) του εκτιμώμενου μεγέθους P_ℓ και (β) η $\log_2$(variance) του "διορθωτικού" όρου $P_\ell - P_{\ell-1}$.

- Το P_ℓ διατηρεί παρόμοια ή σχετικά σταθερή διακύμανση σε όλα τα επίπεδα, υποδηλώνοντας ότι οι coarse (χαμηλό ℓ) και οι fine προσομοιώσεις (υψηλό ℓ) δίνουν παρόμοια "τάξη μεγέθους" στην τυχαιότητα του payoff.
- Η διακύμανση της διαφοράς $P_\ell - P_{\ell-1}$ μειώνεται αισθητά καθώς αυξάνει το ℓ. Αυτό είναι βασικός μηχανισμός της MLMC: η "διορθωτική" ποσότητα γίνεται όλο και πιο μικρή και με μικρότερη διακύμανση, επιτρέποντας τη μείωση του συνολικού κόστους (λιγότερα δείγματα στα ακριβά επίπεδα).

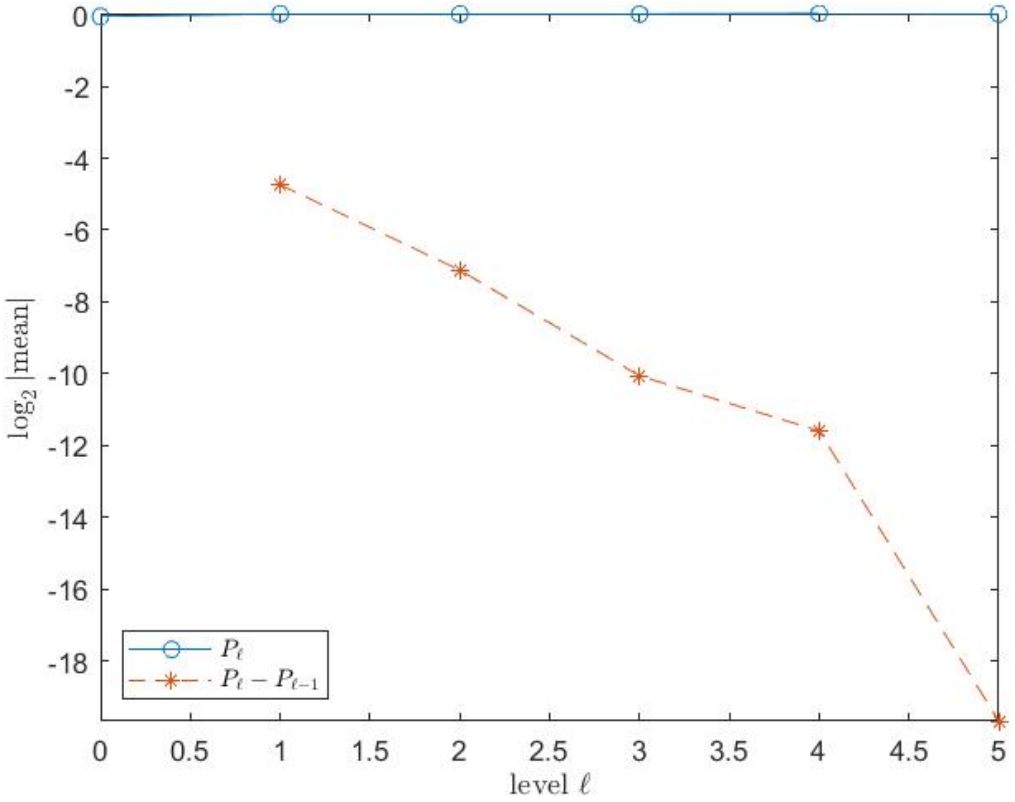

Figure 5: Mean per Level

2) Διάγραμμα $\log_2(|\text{mean}|)$ έναντι ℓ

Στο διάγραμμα απεικονίζεται το μέγεθος της μέσης τιμής (payoff) P_ℓ και της διαφοράς $P_\ell - P_{\ell-1}$ σε λογαριθμική κλίμακα βάσης 2:

- Το P_ℓ συχνά διατηρεί σταθερά χαμηλό (ή μέτριο) μέγεθος, ενίοτε μεταβαλλόμενο λίγο με ℓ.
- Η διαφορά $P_\ell - P_{\ell-1}$ μειώνεται ραγδαία, επιβεβαιώνοντας ότι τα υψηλά επίπεδα λειτουργούν όντως ως μια μικρή βελτίωση των ήδη υπολογισμένων χαμηλών επιπέδων.

Η δραστική μείωση του μέσου μεγέθους της διαφοράς δείχνει γιατί χρειάζονται ελάχιστα δείγματα στα fine επίπεδα (η συνεισφορά τους στην τελική εκτίμηση είναι πολύ μικρή).

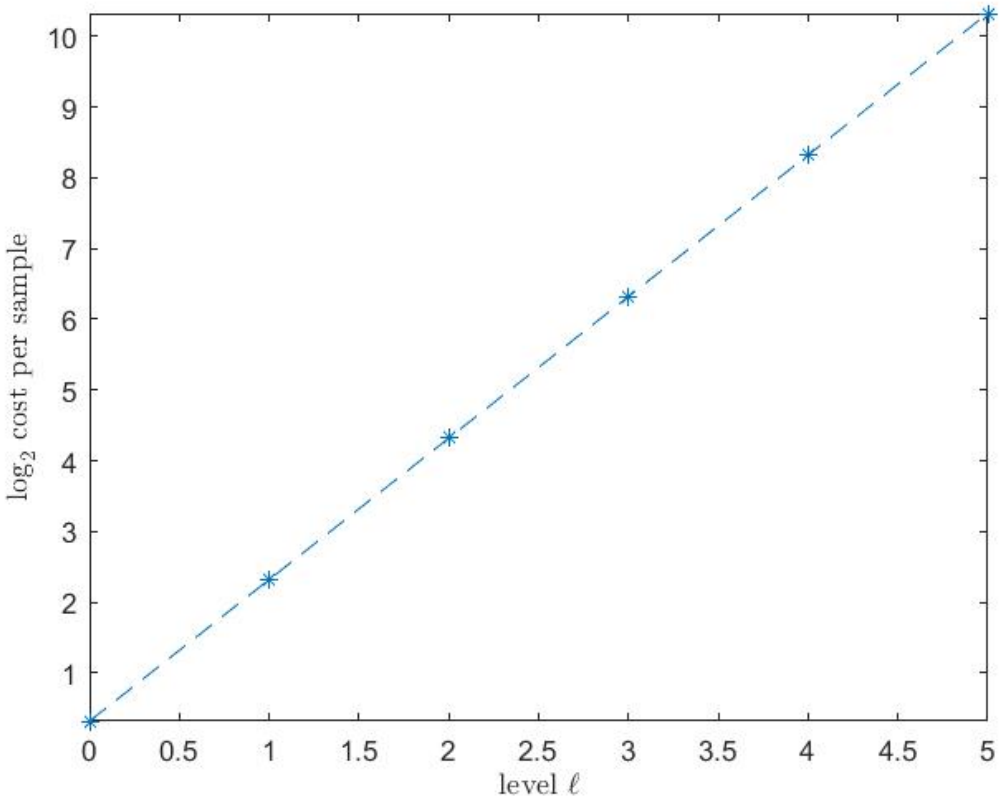

Figure 6: Cost per Sample per Level

3) Διάγραμμα $\log_2$(cost per sample) έναντι ℓ

Εδώ βλέπουμε πώς κλιμακώνεται το κόστος ανά δείγμα με το επίπεδο ℓ.

- Καθώς το ℓ αυξάνεται, ο αριθμός βημάτων στη χρονοβηματική μέθοδο μεγαλώνει, με αποτέλεσμα το κόστος υπολογισμού της προσομοίωσης να αυξάνεται εκθετικά.

Αυτό δικαιολογεί γιατί, στην MLMC, δεν θέλουμε να διαθέσουμε πολλά δείγματα στα υψηλά επίπεδα (το κόστος τους είναι αρκετά μεγάλο σε σχέση με τα χαμηλά επίπεδα).

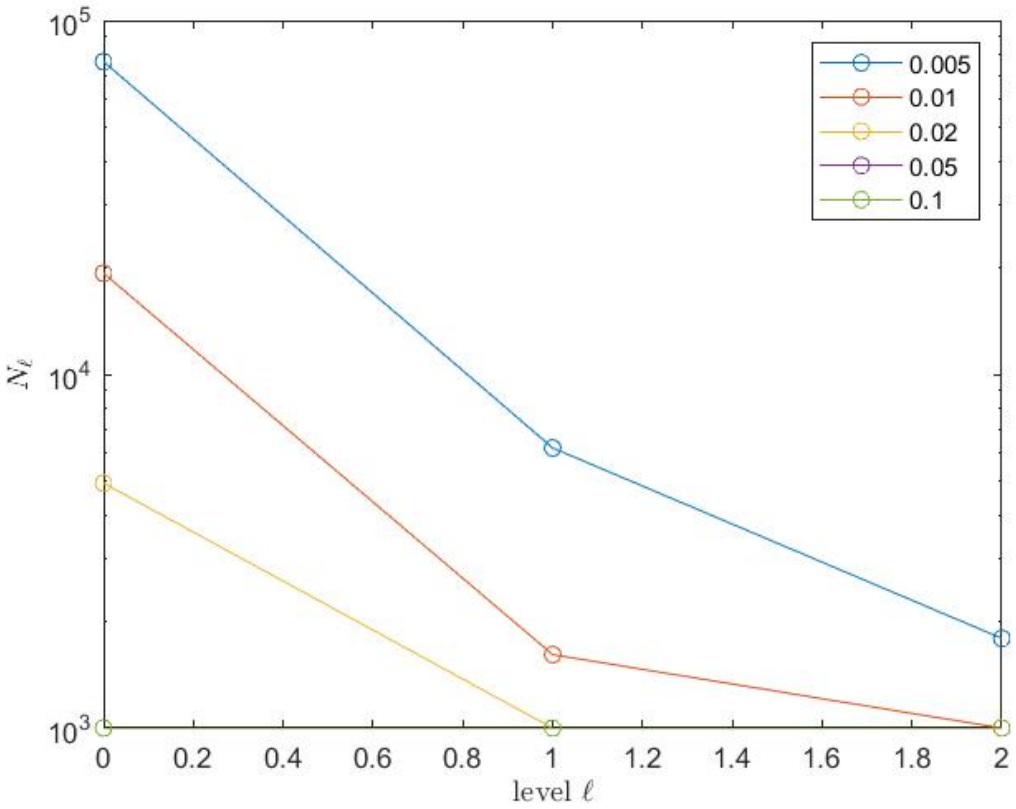

Figure 7: Number of samples per estimation per Level

4) Διάγραμμα N_ℓ έναντι του επιπέδου ℓ

Στο διάγραμμα αυτό, βλέπουμε την κατανομή των αριθμών των δειγμάτων N_ℓ στα διάφορα επίπεδα ℓ για διάφορες τιμές ε. Οι κυριότερες παρατηρήσεις είναι:

- Για μικρές τιμές του ε, παρατηρείται έντονη συγκέντρωση δειγμάτων στα coarser επίπεδα ($\ell = 0$ ή $\ell = 1$). Αυτό συμβαίνει διότι οι προσομοιώσεις στα χαμηλά επίπεδα είναι φθηνότερες και το MLMC εκμεταλλεύεται αυτήν τη διαφορά κόστους.

- Όταν το ε αυξάνεται (π.χ. $\varepsilon = 0.1$), η κατανομή τείνει να "εξομαλύνεται" και οι διαφορές ανάμεσα στα επίπεδα μειώνονται, καθώς δεν απαιτείται πλέον τόσο λεπτομερής ανάλυση στα υψηλά επίπεδα.

Αυτή η προσαρμοστική κατανομή δειγμάτων επιβεβαιώνει το κεντρικό πλεονέκτημα της MLMC: αφιερώνει περισσότερους πόρους στα φθηνά και σημαντικά επίπεδα, ενώ περιορίζει δειγματοληψία στα ακριβά επίπεδα μόνο όσο χρειάζεται.

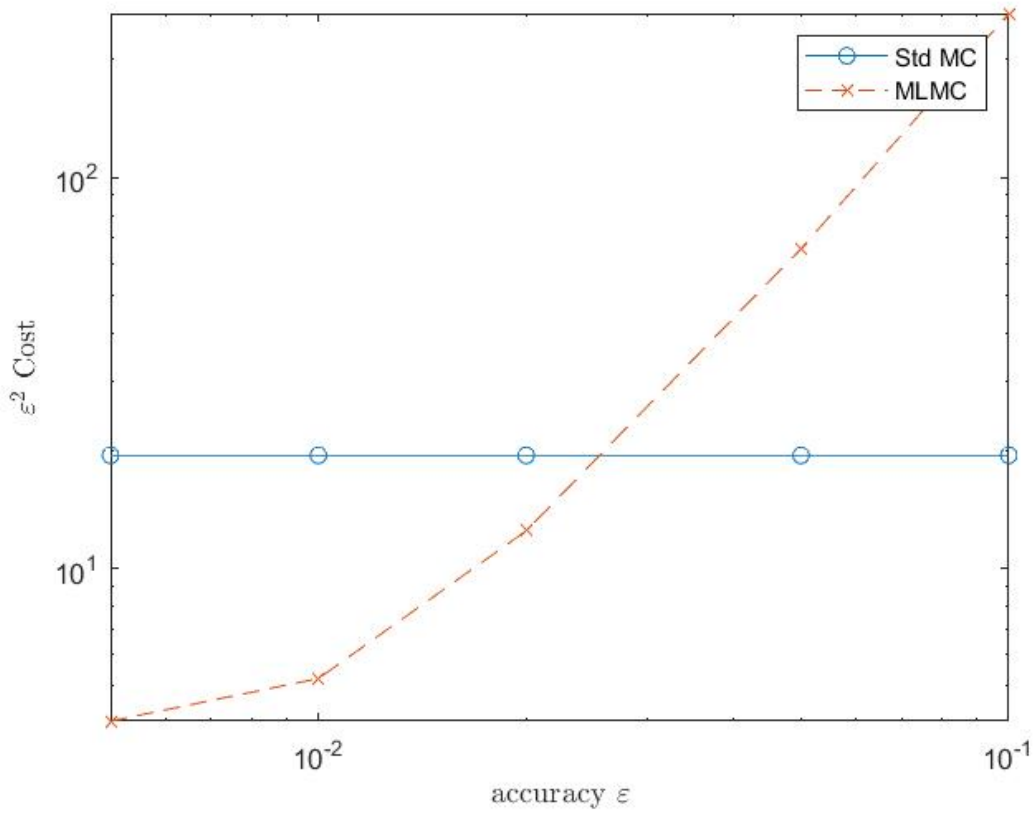

Figure 8: MLMC vs MC per Level

5) Διάγραμμα $\varepsilon^2 \times$ Cost έναντι της ακρίβειας ε

Στο διάγραμμα αυτό συγκρίνεται η τροποποιημένη συνάρτηση κόστους $\varepsilon^2 \times$ Cost για την κλασική Monte Carlo (Std MC) και τη Multilevel (MLMC).

- Η καμπύλη της Std MC παρουσιάζεται σχεδόν ως σταθερή, κάτι που είναι αναμενόμενο για μια κλασική μονοεπίπεδη προσέγγιση, όπου το κόστος τυπικά μεγαλώνει απότομα για μικρότερα ε.
- Η καμπύλη της MLMC παραμένει αρκετά χαμηλή σε χαμηλές τιμές ε, αλλά ανεβαίνει, όταν η ακρίβεια αυξάνεται.
- Η απόσταση ανάμεσα στις δύο καμπύλες δείχνει το όφελος της MLMC ως προς το κόστος για μια δεδομένη απαιτούμενη ακρίβεια. Στις πολύ μικρές ε (υψηλή ακρίβεια), η MLMC υπερτερεί ξεκάθαρα σε απόδοση.

7 Appendix

7.1 Monte Carlo Estimation - Standard + Antithetic Variates (MATLAB)

```
%%%%%%%%%%%%%%%% Problem and method parameters %%%%%%%%%%%%%%%
S = 4; E = 5; sigma = 0.3; r = 0.04; T = 1;
Dt = 1e-3; N = T/Dt; M = 1e4;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Compute the exact Black-Scholes price for comparison
d1 = (log(S/E) + (r + 0.5*sigma^2) * T) / (sigma * sqrt(T));
d2 = d1 - sigma * sqrt(T);
val = E * exp(-r*T) * normcdf(-d2) - S * normcdf(-d1);
rng(100, 'twister'); % random number generator initialization

V = zeros(M,1);

for i = 1:M
    % Option Paths
    z = randn();
    Sfinal = S * exp((r - 0.5 * sigma^2) * T + sigma * sqrt(T) * z);
    Sfinal_anti = S * exp((r - 0.5 * sigma^2) * T - sigma * sqrt(T) * z);
    % Payoff
    V(i) = exp(-r * T) * max(E - Sfinal, 0);
    Vanti(i) = exp(-r * T) * max(E - Sfinal_anti, 0);
    Vfinal(i) = 0.5 * (V(i) + Vanti(i));
end

% Original Path
aM = mean(V);
bM = std(V);
conf = [aM - 1.96 * bM / sqrt(M), aM + 1.96 * bM / sqrt(M)];

% Antithetic Path
aM_final = mean(Vfinal);
bM_final = std(Vfinal);
conf_final = [aM_final - 1.96 * bM_final / sqrt(M), aM_final + 1.96 *
    bM_final / sqrt(M)];

Cost_per_sample = N;
MC_cost = M * Cost_per_sample;

% Display results
fprintf('Exact Option Value (Black-Scholes)  : %f\n', val)
fprintf('Mean (option value) MC      : %f\n', aM);
fprintf('Std. deviation of payoffs MC: %f\n', bM);
fprintf('95%% confidence interval MC: [%f, %f]\n', conf(1), conf(2));
fprintf('Mean (option value) Antithetic: %f\n', aM_final);
fprintf('Std. deviation of payoffs Antithetic: %f\n', bM_final);
fprintf('95%% confidence interval Antithetic: [%f, %f]\n', conf_final(1),
    conf_final(2));
fprintf('Monte Carlo method cost   : %f\n', MC_cost);
```

Listing 1: Monte Carlo estimation of a European put option - standard & with antithetic variates (MATLAB)

7.2 Monte Carlo Estimation for European Put Option (Python)

```python
import numpy as np
import scipy.stats as st
import timeit

def european_put_option_mc(S0, E, r, sigma, T, N):
    Z = np.random.standard_normal(N)
    ST = S0 * np.exp((r - 0.5 * sigma**2) * T + sigma * np.sqrt(T) * Z)
    V = np.exp(-r * T) * np.maximum(E - ST, 0)

    aM = np.mean(V)
    bM = np.std(V)

    conf = st.t.interval(0.95, N - 1, loc=aM, scale=bM / np.sqrt(N))
    ci_width = conf[1] - conf[0]

    return aM, bM, conf, ci_width

if __name__ == "__main__":
    S = 4
    E = 5
    sigma = 0.3
    r = 0.04
    T = 1
    N = 10000

    d1 = (np.log(S / E) + (r + 0.5 * sigma**2) * T) / (sigma * np.sqrt(T))
    d2 = d1 - sigma * np.sqrt(T)
    val = E * np.exp(-r * T) * st.norm.cdf(-d2) - S * st.norm.cdf(-d1)

    print(f"Exact Option Value through B-S Formula: {val:.4f}")

    num_runs = 1000
    total_time = timeit.timeit(lambda: european_put_option_mc(S, E, r, sigma
        , T, N), number=num_runs)
    average_execution_time = total_time / num_runs

    print(f"Average execution time: {average_execution_time:.6f} seconds")

    put_price, put_std, put_ci, put_ci_width = european_put_option_mc(S, E,
        r, sigma, T, N)

    print("=== Monte Carlo Simulation ===")
    print(f"MC Put Price             : {put_price:.4f}")
    print(f"Std Dev                  : {put_std:.4f}")
    print(f"95% CI                   : ({put_ci[0]:.4f}, {put_ci[1]:.4f})")
    print(f"CI Width                 : {put_ci_width:.4f}")
    print(f"Average Execution Time   : {average_execution_time:.6f} seconds"
        )
```

Listing 2: Monte Carlo Simulation for European Put Option in Python

7.3 Monte Carlo Estimation for European Put Option using Antithetic Variates (Python)

```python
import numpy as np
import scipy.stats as st
import timeit

def european_put_option_mc_ant(S0, E, r, sigma, T, N):
    Z = np.random.standard_normal(N)
    Z_ant = -Z
    ST_1 = S0 * np.exp((r - 0.5*sigma**2)*T + sigma*np.sqrt(T)*Z)
    ST_2 = S0 * np.exp((r - 0.5*sigma**2)*T + sigma*np.sqrt(T)*Z_ant)

    payoff_1 = np.maximum(E - ST_1, 0)
    payoff_2 = np.maximum(E - ST_2, 0)

    V = 0.5 * (payoff_1 + payoff_2)
    V = np.exp(-r*T) * V

    aM = np.mean(V)
    bM = np.std(V, ddof=1)
    conf = st.t.interval(0.95, df=N-1, loc=aM, scale=bM/np.sqrt(N))
    ci_width = conf[1] - conf[0]

    return aM, bM, conf, ci_width

if __name__ == "__main__":
    S = 4
    E = 5
    sigma = 0.3
    r = 0.04
    T = 1
    N = 10000

    d1 = (np.log(S/E) + (r + 0.5*sigma**2)*T) / (sigma*np.sqrt(T))
    d2 = d1 - sigma*np.sqrt(T)
    exact_bs = E*np.exp(-r*T)*st.norm.cdf(-d2) - S*st.norm.cdf(-d1)

    num_runs = 1000
    total_time = timeit.timeit(lambda: european_put_option_mc_ant(S, E, r,
        sigma, T, N), number=num_runs)
    average_execution_time = total_time / num_runs

    put_price_ant, put_std_ant, put_ci_ant, put_ci_width_ant =
        european_put_option_mc_ant(S, E, r, sigma, T, N)

    print(f"Exact B-S Put Value       : {exact_bs:.4f}")
    print("=== Monte Carlo with Antithetic Variates ===")
    print(f"MC Put Price              : {put_price_ant:.4f}")
    print(f"Std Dev                   : {put_std_ant:.4f}")
    print(f"95% CI                    : ({put_ci_ant[0]:.4f}, {put_ci_ant
        [1]:.4f})")
    print(f"CI Width                  : {put_ci_width_ant:.4f}")
    print(f"Average Execution Time    : {average_execution_time:.6f} seconds"
        )
```

Listing 3: Monte Carlo with Antithetic Variates for European Put Option (Python)

7.4 MLMC Method for European Put Option (MATLAB)

Main Script: `European_put_MLMC.m`

```
function European_put_MLMC
close all; clear all;
addpath('..');

% Parameters for European Put Option
S = 4;    % Initial Stock Price
E = 5;    % Strike Price
sigma = 0.3; % Volatility
r = 0.04;  % Risk-Free Interest Rate
T = 1;    % Time to Maturity
N0 = 1000; % Initial samples on coarse levels
Lmin = 2; % Minimum refinement level
Lmax = 6; % Maximum refinement level

fprintf(1,'\n  European Put Option  \n');
N = 20000; % Samples for convergence tests
L = 5; % Levels for convergence tests
Eps = [ 0.005 0.01 0.02 0.05 0.1];

filename = 'eurMLMC__put';
fp = fopen([filename '.txt'],'w');
mlmc_test(@European_put_l, N, L, N0, Eps, Lmin, Lmax, fp);
fclose(fp);

% Compute the exact Black-Scholes price for comparison
d1 = (log(S/E) + (r + 0.5*sigma^2) * T) / (sigma * sqrt(T));
d2 = d1 - sigma * sqrt(T);
val = E * exp(-r*T) * normcdf(-d2) - S * normcdf(-d1);
fprintf(1,'\n Exact value: %f \n', val);

% Plot results
nvert = 3;
mlmc_plot(filename, nvert);
if(nvert==1)
    figure(1)
    print('-deps2',[filename 'a.eps'])
    figure(2)
    print('-deps2',[filename 'b.eps'])
else
    print('-deps2',[filename '.eps'])
end
end
```

Listing 4: Main MLMC script for pricing a European put option

Level Simulation Function (MATLAB): `European_put_l.m`

```
function [sums, cost] = European_put_l(l, N)
M = 4; % Time step refinement factor
T = 1; % Time to maturity
r = 0.04; % Risk-free interest rate
sigma = 0.3; % Volatility
E = 5; % Strike Price
S = 4; % Initial stock price
nf = M^l;
nc = nf/M;
hf = T/nf; % Fine level time step
hc = T/nc; % Coarse level time step
sums(1:6) = 0;

for N1 = 1:10000:N
    N2 = min(10000, N-N1+1);
    X0 = S;
    Xf = X0 * ones(1, N2);
    Xc = Xf;
    if l == 0
        dWf = sqrt(hf) * randn(1, N2);
        Xf = Xf + r*Xf*hf + sigma*Xf.*dWf;
    else
        for n = 1:nc
            dWc = zeros(1, N2);
            for m = 1:M
                dWf = sqrt(hf) * randn(1, N2);
                dWc = dWc + dWf;
                Xf = Xf + r*Xf*hf + sigma*Xf.*dWf;
            end
            Xc = Xc + r*Xc*hc + sigma*Xc.*dWc;
        end
    end

    % European Put Payoff Calculation
    Pf = max(E - Xf, 0); % Fine level payoff
    Pc = max(E - Xc, 0); % Coarse level payoff

    Pf = exp(-r*T) * Pf;
    Pc = exp(-r*T) * Pc;

    if l == 0
        Pc = 0;
    end

    sums(1) = sums(1) + sum(Pf - Pc);
    sums(2) = sums(2) + sum((Pf - Pc).^2);
    sums(3) = sums(3) + sum((Pf - Pc).^3);
    sums(4) = sums(4) + sum((Pf - Pc).^4);
    sums(5) = sums(5) + sum(Pf);
    sums(6) = sums(6) + sum(Pf.^2);
end

cost = N * (nf + nc); % Cost defined as total number of time steps
end
```

Listing 5: Function defining simulation at level `l` for MLMC

7.5 MLMC Method for European Put Option (Python)

Euler-Maruyama Paths for European Put Option

```python
import numpy as np
import math

class EmgbmLevel:
    def __init__(self, level, S0, E, sigma, r, T):
        self.l = level
        self.S0 = S0
        self.E = E
        self.sigma = sigma
        self.r = r
        self.T = T
        self.M = 4

        self.nf = self.M**level if level > 0 else 1
        self.nc = self.nf // self.M if level > 0 else 0

        self.hf = T / self.nf if self.nf > 0 else 0
        self.hc = T / self.nc if self.nc > 0 else 0

        self.cost = self.nf + self.nc

    def evaluate(self, increments_fine):
        N = increments_fine.shape[0]
        Xf = np.full(N, self.S0)
        Xc = np.full(N, self.S0) if self.l > 0 else np.zeros(N)

        if self.l == 0:
            dWf = increments_fine[:, 0] * math.sqrt(self.hf)
            Xf += self.r * Xf * self.hf + self.sigma * Xf * dWf
        else:
            idx = 0
            for _ in range(self.nc):
                dWc_sum = np.zeros(N)
                for _ in range(self.M):
                    dWf = increments_fine[:, idx] * math.sqrt(self.hf)
                    idx += 1
                    Xf += self.r * Xf * self.hf + self.sigma * Xf * dWf
                    dWc_sum += dWf
                Xc += self.r * Xc * self.hc + self.sigma * Xc * dWc_sum

        Pf = np.exp(-self.r * self.T) * np.fmax(self.E - Xf, 0.0)
        if self.l == 0:
            Pc = np.zeros(N)
        else:
            Pc = np.exp(-self.r * self.T) * np.fmax(self.E - Xc, 0.0)
        return Pf, Pc

    def em_gbm_euler_sampler(N, l):
    nfine = 4**l if l > 0 else 1
    samplef = np.random.randn(N, nfine)
    samplec = np.zeros((N, 1))
    return samplef, samplec
```

Listing 6: Class EmgbmLevel: Fine and Coarse Euler–Maruyama paths

```
S = 4.0
E = 5.0
sigma = 0.3
r = 0.04
T = 1.0
Lmax = 6

problems = []
for lev in range(Lmax + 1):
    prob = EmgbmLevel(lev, S, E, sigma, r, T)
    prob.cost = prob.nf + prob.nc
    problems.append(prob)
```

Listing 7: Problem setup: Create one EmgbmLevel instance per level

Main Function

```
import numpy as np

def main():
    np.random.seed(1)

    S, E, sigma, r, T = 4.0, 5.0, 0.3, 0.04, 1.0
    N0, Lmin, Lmax, N, L = 1000, 2, 6, 20000, 5

    Eps = [0.005, 0.01, 0.02, 0.05, 0.1]

    problems = []
    for lev in range(Lmax + 1):
        prob = EmgbmLevel(lev, S, E, sigma, r, T)
        prob.cost = prob.nf + prob.nc
        problems.append(prob)

    mlmc_test(
        mlmc_fn=lambda lev, Ns: mlmc_fn(
            l=lev,
            N=Ns,
            problems=problems,
            sampler=em_gbm_euler_sampler,
            coupled_problem=True
        ),
        N=N,
        L=L,
        N0=N0,
        Eps=Eps,
        Lmin=Lmin,
        Lmax=Lmax,
        logfile=None)

if __name__ == "__main__":
    main()
```

Listing 8: Main function: Calls mlmc_test with custom components

Οι βασικές συναρτήσεις `mlmc`, `mlmc_fn`, και `mlmc_test` βασίζονται στον κώδικα του Giles[1].

[1] M.B. Giles, Multilevel Monte Carlo methods, https://people.maths.ox.ac.uk/gilesm/mlmc/

7.6 Code for $Sum of Dice = 7$ Example

```python
import random

def monte_carlo_sum_7(num_trials):
    count_sum_7 = 0
    for _ in range(num_trials):
        die1 = random.randint(1, 6)
        die2 = random.randint(1, 6)
        if die1 + die2 == 7:
            count_sum_7 += 1
    probability = count_sum_7 / num_trials
    return probability

for trials in [1000, 10000, 100000, 1000000]:
    est_prob = monte_carlo_sum_7(trials)
    print(f"Number of Throws: {trials}, "
          f"Calculated Probability: {est_prob:.4f}, "
          f"Error from 1/6: {abs(est_prob - 1/6):.4f}")
```

Listing 9: Monte Carlo estimation of probability that two dice sum to 7

Βιβλιογραφία

[Giles, 2018] Giles, M. B. (2018). Multilevel monte carlo methods. *Acta Numerica*, 24.

[Heinrich, 2001] Heinrich, S. (2001). Multilevel monte carlo methods. In *Large-Scale Scientific Computing*, pages 58–67. Springer.

[Highman, 2004] Highman, D. J. (2004). *An Introduction to Financial Option Valuation: Mathematics, Stochastics and Computation*. Cambridge University Press.

[Hull, 2017] Hull, J. C. (2017). *Options, Futures, and Other Derivatives*. Pearson, 9th edition.

[Metropolis, 1987] Metropolis, N. (1987). The beginning of the monte carlo method. *Los Alamos Science*, 15:125–130. Available at: `https://mcnp.lanl.gov/pdf_files/Article_1987_LAS_Metropolis_125--130.pdf`.

[Metropolis and Ulam, 1949] Metropolis, N. and Ulam, S. (1949). The monte carlo method. *Journal of the American Statistical Association*, 44(247):335–341. Available at: `https://web.williams.edu/Mathematics/sjmiller/public_html/105Sp10/handouts/MetropolisUlam_TheMonteCarloMethod.pdf`.