Research on Monte Carlo Methods

Mike Giles

mike.giles@maths.ox.ac.uk

Oxford University Mathematical Institute

Mathematical and Computational Finance Group

Nomura, Tokyo, August 13, 2009

Outline

- A little about me and my research
- Monte Carlo sensitivities – computing the Greeks
  - pathwise sensitivities using adjoints (Paul Glasserman)
  - pathwise sensitivities for digital options
- Multilevel Monte Carlo simulation
- Computing on GPUs

Background

- 1981, BA Maths, Cambridge
- 1985, PhD, Aeronautical Engineering, MIT

- 1985 – 92, Professor at MIT in Aeronautics
- 1992 – present, Professor at Oxford University in Computing Laboratory and Mathematical Institute

- 1981 – 2007: research on computational fluid dynamics
- 2005 – present: research on computational finance, in particular Monte Carlo methods
- my research focus is on numerical methods and computing, rather than developing better models

“Smoking Adjoints”

Paper with Paul Glasserman in *Risk* in 2006 on the use of adjoints in computing pathwise sensitivities attracted a lot of interest, and questions:

- what is involved in practice in creating an adjoint code, and can it be simplified?
- do we really have to differentiate the payoff?
- what about non-differentiable payoffs?

Outline

- different approaches to computing Greeks
  - finite differences
  - likelihood ratio method
  - pathwise sensitivity
- using adjoints for pathwise sensitivities
- use of automatic differentiation
- use of conditional expectation for digital options, and “vibrato” extension for multi-dimensional SDEs

Generic Problem

Stochastic differential equation with general drift and volatility terms:

$$\mathrm{d}S_t = a(S_t, t)\,\mathrm{d}t + b(S_t, t)\,\mathrm{d}W_t$$

For a simple European option we want to compute the expected discounted payoff value dependent on the terminal state:

$$V = \mathbb{E}[f(S_T)]$$

Note: the drift and volatility functions are almost always differentiable, but the payoff $f(S)$ is often not.

Generic Problem

Euler discretisation with timestep h:

$$\widehat{S}_{n+1} = \widehat{S}_n + a(\widehat{S}_n, t_n)\, h + b(\widehat{S}_n, t_n)\, \Delta W_n$$

Simplest Monte Carlo estimator for expected payoff is an average of M independent path simulations:

$$M^{-1} \sum_{i=1}^{M} f(\widehat{S}_N^{(i)})$$

Greeks: for hedging and risk management we also want to estimate derivatives of expected payoff V

Finite Differences

Simplest approach is to use a finite difference approximation,

$$\frac{\partial V}{\partial \theta} \approx \frac{V(\theta+\Delta\theta) - V(\theta-\Delta\theta)}{2\,\Delta\theta}$$

$$\frac{\partial^2 V}{\partial \theta^2} \approx \frac{V(\theta+\Delta\theta) - 2\,V(\theta) + V(\theta-\Delta\theta)}{(\Delta\theta)^2}$$

– very simple, but expensive and inaccurate if $\Delta\theta$ is too big, or too small in the case of discontinuous payoffs

Likelihood Ratio Method

For simple cases where we know the terminal probability distribution

$$V \equiv \mathbb{E}\left[f(S_T)\right] = \int f(S)\, p_S(\theta; S)\, \mathrm{d}S$$

we can differentiate this to get

$$\frac{\partial V}{\partial \theta} = \int f \frac{\partial p_S}{\partial \theta}\, \mathrm{d}S = \int f \frac{\partial (\log p_S)}{\partial \theta}\, p_S\, \mathrm{d}S = \mathbb{E}\left[f \frac{\partial (\log p_S)}{\partial \theta}\right]$$

This is the Likelihood Ratio Method (Broadie & Glasserman, 1996) – its great strength is that it can handle discontinuous payoffs

Likelihood Ratio Method

The LRM weakness is in its generalisation to full path simulations for which we get the multi-dimensional integral

$$\widehat{V} = \mathbb{E}[f(\widehat{S})] = \int f(\widehat{S})\, p(\widehat{S})\, \mathrm{d}\widehat{S},$$

where $\mathrm{d}\widehat{S} \equiv \mathrm{d}\widehat{S}_1\ \mathrm{d}\widehat{S}_2\ \mathrm{d}\widehat{S}_3\ \ldots\ \mathrm{d}\widehat{S}_N$

LRM approach stills works, but the variance is $O(h^{-1})$ in general; this blow-up as $h \to 0$ is the weakness of the LRM.

Pathwise sensitivities

Alternatively, for simple Geometric Brownian Motion

$$V \equiv \mathbb{E}\left[f(S_T)\right] = \int f(S_T(\theta; W))\, p_W(W)\, \mathrm{d}W$$

and differentiating this gives

$$\frac{\partial V}{\partial \theta} = \int \frac{\partial f}{\partial S}\, \frac{\partial S_T}{\partial \theta}\, p_W\, \mathrm{d}W = \mathbb{E}\left[\frac{\partial f}{\partial S}\, \frac{\partial S_T}{\partial \theta}\right]$$

with $\partial S_T/\partial\theta$ being evaluated at fixed W.

This is the pathwise sensitivity approach – it can't handle discontinuous payoffs, but generalises well to full path simulations

Pathwise sensitivities

The generalisation involves differentiating the Euler path discretisation,

$$\widehat{S}_{n+1} = \widehat{S}_n + a(\widehat{S}_n, t_n)\, h + b(\widehat{S}_n, t_n)\, \Delta W_n$$

holding fixed the Brownian increments, to get

$$\frac{\partial \widehat{S}_{n+1}}{\partial \theta} = \left(1 + \frac{\partial a}{\partial S}\, h + \frac{\partial b}{\partial S}\, \Delta W_n\right) \frac{\partial \widehat{S}_n}{\partial \theta} + \frac{\partial a}{\partial \theta}\, h + \frac{\partial b}{\partial \theta}\, \Delta W_n$$

leading to

$$\frac{\partial \widehat{V}}{\partial \theta} = \mathbb{E}\left[\frac{\partial f}{\partial S}(\widehat{S}_N)\, \frac{\partial \widehat{S}_N}{\partial \theta}\right]$$

with a variance which remains bounded as $h \to 0$.

Adjoint sensitivities

The adjoint approach is an efficient implementation of pathwise sensitivities.

Consider a process in which a vector input α leads to a final state vector S which is used to compute a scalar payoff P

$$\alpha \quad \longrightarrow \quad S \quad \longrightarrow \quad P$$

Taking $\dot{\alpha}, \dot{S}, \dot{P}$ to be the derivatives w.r.t. j^{th} component of α, then

$$\dot{S} = \frac{\partial S}{\partial \alpha}\, \dot{\alpha}, \qquad \dot{P} = \frac{\partial P}{\partial S}\, \dot{S},$$

and hence

$$\dot{P} = \frac{\partial P}{\partial S}\, \frac{\partial S}{\partial \alpha}\, \dot{\alpha}.$$

Adjoint sensitivities

Alternatively, defining $\overline{\alpha}, \overline{S}, \overline{P}$ to be the derivatives of P with respect to α, S, P, then

$$\overline{\alpha} \stackrel{\text{def}}{=} \left(\frac{\partial P}{\partial \alpha}\right)^T = \left(\frac{\partial P}{\partial S}\frac{\partial S}{\partial \alpha}\right)^T = \left(\frac{\partial S}{\partial \alpha}\right)^T \overline{S},$$

and similarly

$$\overline{S} = \left(\frac{\partial P}{\partial S}\right)^T \overline{P},$$

giving

$$\overline{\alpha} = \left(\frac{\partial S}{\partial \alpha}\right)^T \left(\frac{\partial P}{\partial S}\right)^T \overline{P}.$$

Adjoint sensitivities

The two are mathematically equivalent, since

$$\dot{P} = \frac{\partial P}{\partial \alpha}\,\dot{\alpha} = \overline{\alpha}^T \dot{\alpha} = \overline{\alpha}_j$$

but the adjoint approach approach is much cheaper because a single calculation gives $\overline{\alpha}$, the sensitivity of P to each one of the elements of α.

- standard approach: cost proportional to number of Greeks
- adjoint approach: cost independent
- crossover point for cost: 4 – 6 Greeks?

Adjoint sensitivities

Note that the standard approach goes forward

$$\dot{\alpha} \quad \longrightarrow \quad \dot{S} \quad \longrightarrow \quad \dot{P}$$

while the adjoint approach does the reverse

$$\overline{\alpha} \quad \longleftarrow \quad \overline{S} \quad \longleftarrow \quad \overline{P}.$$

These correspond to the forward and reverse modes of AD (Automatic Differentiation).

“Smoking Adjoints” paper extended this to multiple timesteps in the path calculation – instead, we’ll extend it to the steps in a whole computer program.

Automatic Differentiation

A computer instruction creates an additional new value:

$$\mathbf{u}^n \;=\; \mathbf{f}^n(\mathbf{u}^{n-1}) \;\equiv\; \begin{pmatrix} \mathbf{u}^{n-1} \\ f_n(\mathbf{u}^{n-1}) \end{pmatrix},$$

and a program is the composition of N such steps.

In forward mode, differentiation w.r.t. one element of the input vector gives

$$\dot{\mathbf{u}}^n = D^n\,\dot{\mathbf{u}}^{n-1}, \quad D^n \equiv \begin{pmatrix} I^{n-1} \\ \partial f_n/\partial \mathbf{u}^{n-1} \end{pmatrix},$$

and hence

$$\dot{\mathbf{u}}^N = D^N\,D^{N-1}\ldots\,D^2\,D^1\,\dot{\mathbf{u}}^0$$

Automatic Differentiation

In reverse mode, we consider the sensitivity of one element of the output vector, to get

$$\left(\overline{\mathbf{u}}^{n-1}\right)^T \equiv \frac{\partial u_i^N}{\partial \mathbf{u}^{n-1}} = \frac{\partial u_i^N}{\partial \mathbf{u}^n} \frac{\partial \mathbf{u}^n}{\partial \mathbf{u}^{n-1}} = \left(\overline{\mathbf{u}}^n\right)^T D^n,$$

$$\Longrightarrow \quad \overline{\mathbf{u}}^{n-1} = \left(D^n\right)^T \overline{\mathbf{u}}^n.$$

and hence

$$\overline{\mathbf{u}}^0 = \left(D^1\right)^T \left(D^2\right)^T \ldots \left(D^{N-1}\right)^T \left(D^N\right)^T \overline{\mathbf{u}}^N.$$

Note: need to go forward through original calculation to compute/store the D^n, then go in reverse to compute $\overline{\mathbf{u}}^n$

Automatic Differentiation

This gives a prescriptive algorithm for reverse mode differentiation.

Again the reverse mode is much more efficient if we want the sensitivity of a single output to multiple inputs.

Key result is that the cost of the reverse mode is at worst a factor 4 greater than the cost of the original calculation, regardless of how many sensitivities are being computed!

The storage of the D^n is minor for SDEs – much more of a concern for PDEs. There are also extra complexities when solving implicit equations through a fixed point iteration.

Automatic Differentiation

Manual implementation of the forward/reverse mode algorithms is possible but tedious.

Fortunately, automated tools have been developed, following one of two approaches:

- operator overloading (ADOL-C, FADBAD++)
- source code transformation (Tapenade, TAF/TAC++, ADIFOR)

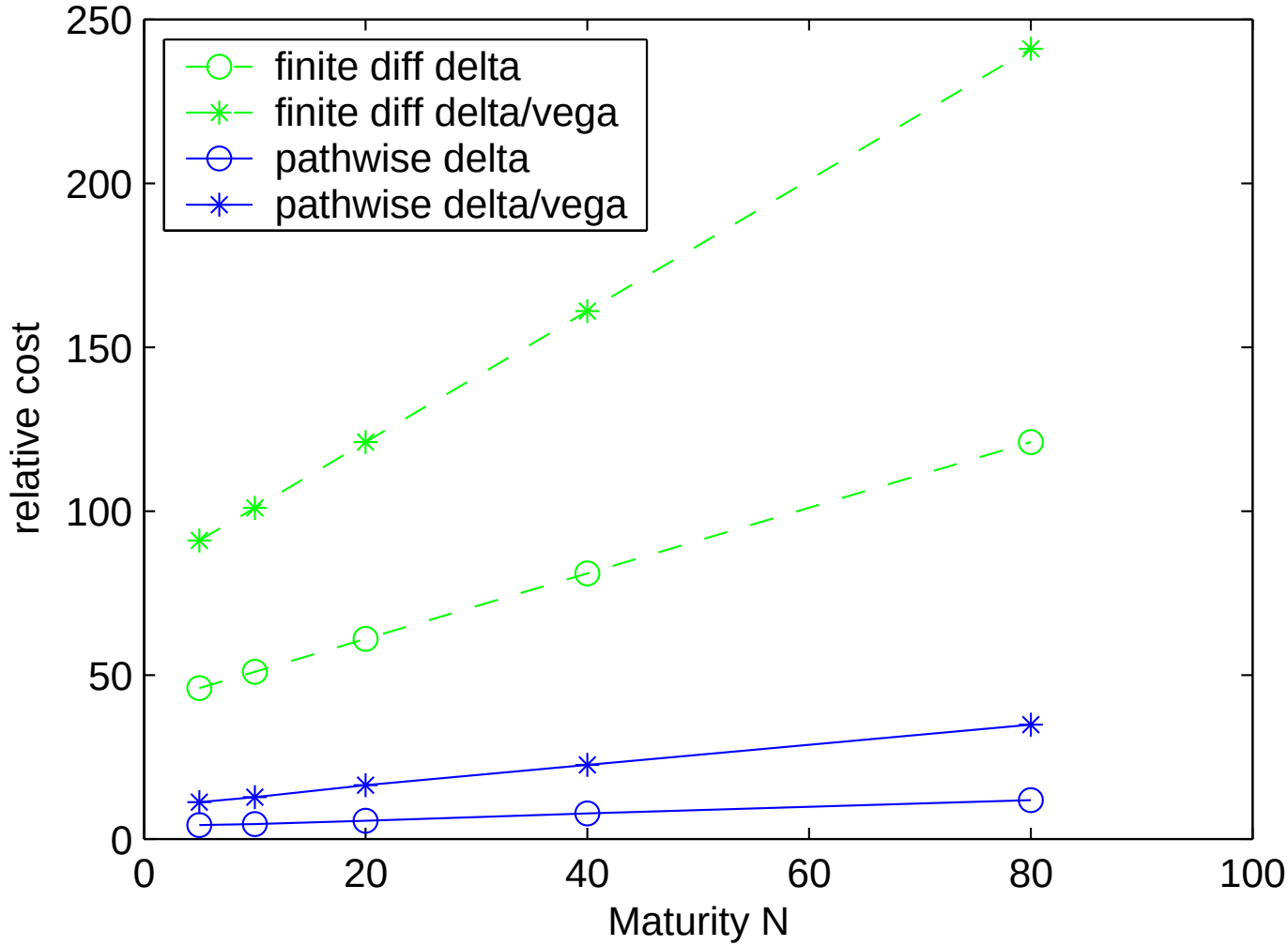
My personal experience is with Tapenade for Fortran, and FADBAD++ for C++. Both are easy to use, Tapenade is as efficient as hand-coded, FADBAD++ less so.

LIBOR Application

- testcase from “Smoking Adjoints” paper
- test problem performs N timesteps with a vector of $N+40$ forward rates, and computes the $N+40$ deltas and vegas for a portfolio of swaptions
- originally hand-coded (using the ideas from AD), now used to test the effectiveness of AD tools

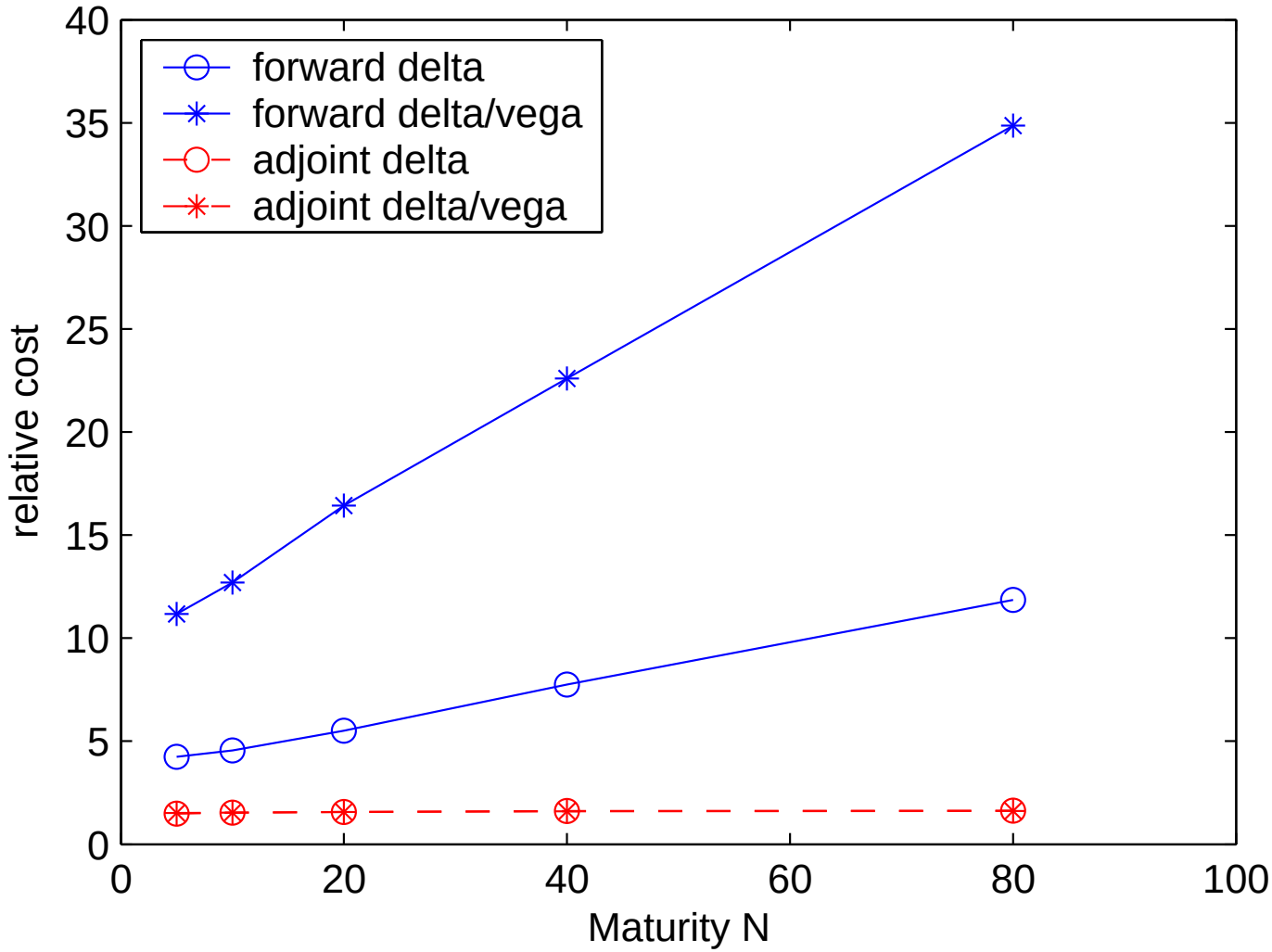
LIBOR Application

Finite differences versus forward pathwise sensitivities:



LIBOR Application

Hand-coded forward versus adjoint pathwise sensitivities:



LIBOR Application

Timings per path for $N = 40$; the hybrid version uses hand-coded for the path and FADBAD++ for the payoff

milliseconds/path	Gnu g++	Intel `icc`
original	0.37	0.10
hand-coded forward	0.97	0.52
hand-coded reverse	0.47	0.19
FADBAD++ forward	4.30	5.00
FADBAD++ reverse	6.20	4.86
hybrid forward	1.02	0.63
hybrid reverse	0.65	0.35
TAC++ forward	1.36	0.85
TAC++ reverse	1.28	0.45

Automatic Differentiation

Conclusions?

- hand-coded is clearly most efficient
  I optimised the implementation (tradeoff between storage versus recomputation) in a way the AD tools cannot yet.
- however, AD code is very useful for debugging/validating hand-coded version, and can be used for bits which are not computationally intensive
- AD is likely to be useful for bigger applications (vital in computational science and engineering)

Vibrato Monte Carlo

One remaining problem – what if payoff is not differentiable?

- LRM
  - estimator variance $O(h^{-1})$
- Malliavin calculus
  - estimator variance $O(1)$
  - recent paper by Glasserman & Chen shows it can be viewed as a pathwise/LRM hybrid
  - might be good choice when few Greeks needed
- new “vibrato” Monte Carlo idea
  - also a pathwise/LRM hybrid
  - estimator variance $O(h^{-1/2})$
  - efficient adjoint implementation

Vibrato Monte Carlo

- new idea is based on use of conditional expectation for a simple digital option in Paul Glasserman's book
- output of each SDE path calculation becomes a narrow (multivariate) Normal distribution
- combine pathwise sensitivity for the differentiable SDE, with LRM for the discontinuous payoff
- avoiding the differentiation of the payoff also simplifies the implementation in real-world setting

Vibrato Monte Carlo

Final timestep of Euler path discretisation is

$$\widehat{S}_N = \widehat{S}_{N-1} + a(\widehat{S}_{N-1}, t_{N-1})\, h + b(\widehat{S}_{N-1}, t_{N-1})\, \Delta W_{N-1}$$

Instead of using random number generator to get a value for ΔW_{N-1}, consider the whole distribution of possible values, so $\widehat{S}_N$ has a Normal distribution with mean

$$\mu_W = \widehat{S}_{N-1} + a(\widehat{S}_{N-1}, t_{N-1})\, h$$

and standard deviation

$$\sigma_W = b(\widehat{S}_{N-1}, t_{N-1})\, \sqrt{h}$$

where $W \equiv (\Delta W_0, \Delta W_1, \ldots \Delta W_{N-2})$.

Vibrato Monte Carlo

For a particular path given by a particular vector W, the expected payoff is

$$\mathbb{E}_Z[f(\mu_W + \sigma_W Z)]$$

where Z is a unit Normal random variable.

Averaging over all W then gives the same overall expectation as before.

Note also that, for given W, $\widehat{S}_N$ has a Normal distribution

$$p_S(\widehat{S}) = \frac{1}{\sqrt{2\pi}\,\sigma_W} \exp\left(-\frac{(\widehat{S}-\mu_W)^2}{2\,{\sigma_W}^2}\right)$$

Vibrato Monte Carlo

In the case of a simple digital call with strike K, the analytic solution is

$$\mathbb{E}_Z[f(\mu_W + \sigma_W Z)] = \exp(-rT)\ \Phi\left(\frac{\mu_W - K}{\sigma_W}\right).$$

- for each W, the payoff is now smooth, differentiable
- derivative is $O(h^{-1/2})$ near strike, near zero elsewhere
  $\Longrightarrow$ variance is $O(h^{-1/2})$
- analytic evaluation of conditional expectation not possible in general for multivariate cases
  $\Longrightarrow$ use Monte Carlo estimation!

Vibrato Monte Carlo

Main novelty comes in calculating the sensitivity.

For a particular W, we have a Normal probability distribution for $\widehat{S}_N$ and can apply the Likelihood Ratio method to get

$$\frac{\partial}{\partial\theta}\,\mathbb{E}_Z\left[f(\widehat{S}_N)\right] \;=\; \mathbb{E}_Z\left[f(\widehat{S}_N)\,\frac{\partial(\log p_S)}{\partial\theta}\right],$$

where

$$\begin{aligned}\frac{\partial(\log p_S)}{\partial\theta} &= \frac{\partial(\log p_S)}{\partial\mu_W}\,\frac{\partial\mu_W}{\partial\theta} + \frac{\partial(\log p_S)}{\partial\sigma_W}\,\frac{\partial\sigma_W}{\partial\theta}\\ &= \frac{Z}{\sigma_W}\,\frac{\partial\mu_W}{\partial\theta} + \frac{Z^2-1}{\sigma_W}\,\frac{\partial\sigma_W}{\partial\theta}.\end{aligned}$$

Averaging over all W then gives the expected sensitivity.

Vibrato Monte Carlo

To improve the variance, we note that

$$\begin{aligned}\mathbb{E}_Z\left[f(\mu_W+\sigma_W Z)\ Z\right] &= \mathbb{E}_Z\left[-f(\mu_W-\sigma_W Z)\ Z\right] \\ &= \tfrac{1}{2}\,\mathbb{E}_Z\left[\Big(f(\mu_W+\sigma_W Z)-f(\mu_W-\sigma_W Z)\Big)\ Z\right]\end{aligned}$$

and similarly

$$\begin{aligned}&\mathbb{E}_Z\left[f(\mu_W+\sigma_W Z)\ (Z^2-1)\right] \\ &= \tfrac{1}{2}\,\mathbb{E}_Z\left[\Big(f(\mu_W+\sigma_W Z)-2f(\mu_W)+f(\mu_W-\sigma_W Z)\Big)\ (Z^2-1)\right]\end{aligned}$$

This gives an estimator with $O(1)$ variance when $f(S)$ is Lipschitz, and $O(h^{-1/2})$ variance when it is discontinuous.

Multivariate extension

In general we have

$$\widehat{S}(W, Z) = \mu_W + C_W Z$$

where $\Sigma_W = C_W {C_W}^T$ is the covariance matrix, and Z is a vector of uncorrelated Normals. The joint p.d.f. is

$$\log p_S = -\tfrac{1}{2} \log |\Sigma_W| - \tfrac{1}{2} (\widehat{S} - \mu_W)^T {\Sigma_W}^{-1} (\widehat{S} - \mu_W) - \tfrac{1}{2} d \log(2\pi)$$

and so

$$\frac{\partial \log p_S}{\partial \mu_W} = {C_W}^{-T} Z,$$

$$\frac{\partial \log p_S}{\partial \Sigma_W} = \tfrac{1}{2} {C_W}^{-T} \left(Z Z^T - I \right) {C_W}^{-1}$$

Multivariate extension

This leads to

$$\frac{\partial}{\partial\theta}\,\mathbb{E}_Z\left[f(\widehat{S})\right] \;=\; \mathbb{E}_Z\left[f(\widehat{S})\;\frac{\partial(\log p_S)}{\partial\theta}\right]$$

where

$$\frac{\partial(\log p_S)}{\partial\theta} = \left(\frac{\partial \log p_S}{\partial \mu_W}\right)^T \frac{\partial \mu_W}{\partial\theta} + \mathrm{tr}\left(\frac{\partial \log p_S}{\partial \Sigma_W}\;\frac{\partial \Sigma_W}{\partial\theta}\right)$$

and $\frac{\partial \mu_W}{\partial\theta}$, $\frac{\partial \Sigma_W}{\partial\theta}$ come from pathwise sensitivity analysis.

A more efficient estimator can be obtained by similar reasoning to the scalar case.

Vibrato Monte Carlo

Test case: Geometric Brownian motion

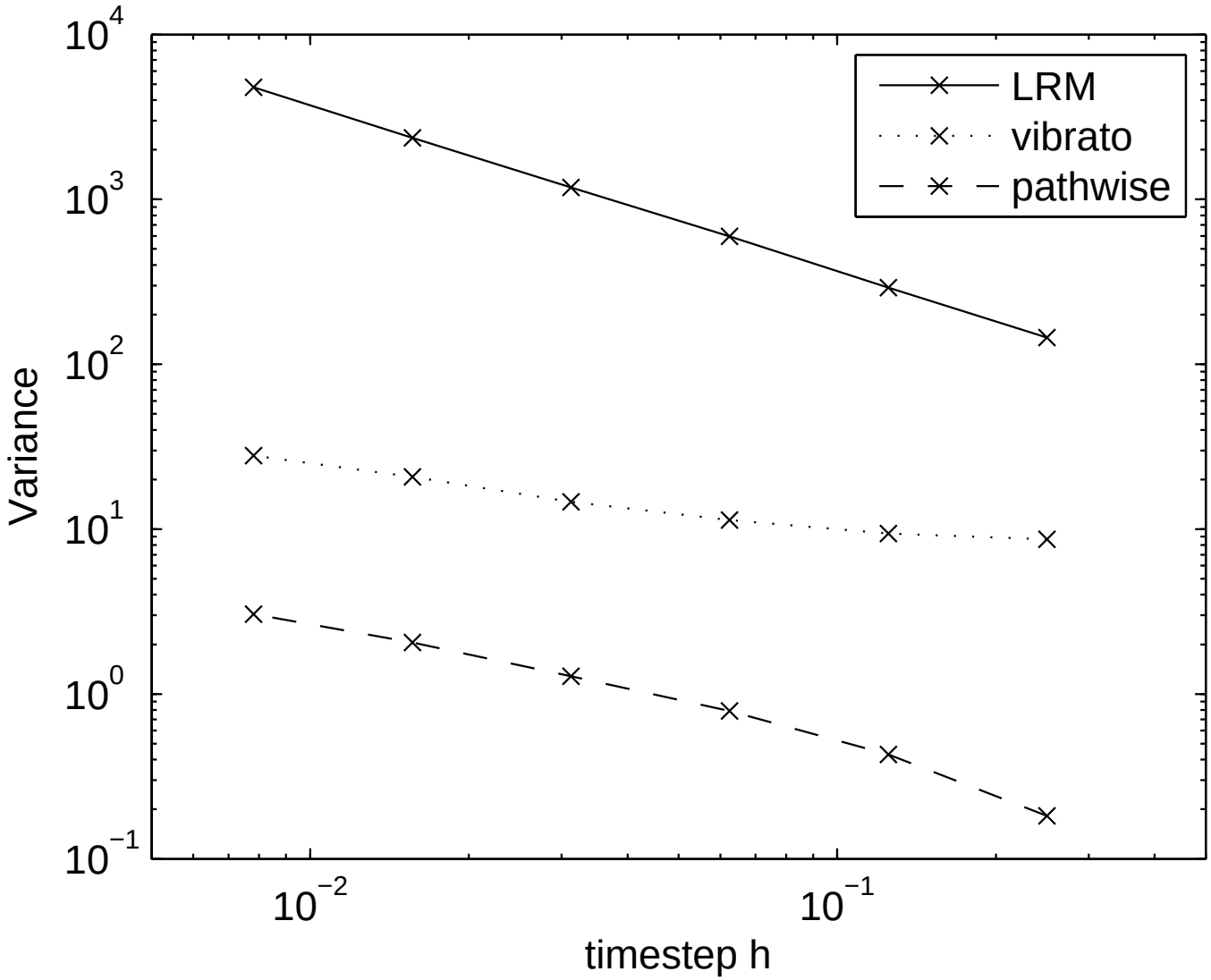
$$\begin{aligned} \mathrm{d}S_t^{(1)} &= r\, S_t^{(1)}\, \mathrm{d}t + \sigma^{(1)}\, S_t^{(1)}\, \mathrm{d}W_t^{(1)} \\ \mathrm{d}S_t^{(1)} &= r\, S_t^{(2)}\, \mathrm{d}t + \sigma^{(2)}\, S_t^{(2)}\, \mathrm{d}W_t^{(2)} \end{aligned}$$

with a simple digital call option based solely on $S_T^{(1)}$.

Parameters: $r = 0.05,\ \sigma^{(1)} = 0.2,\ \sigma^{(2)} = 0.3,\ T = 1,\ S_0^{(1)} = S_0^{(2)} = 100,\ K = 100,\ \rho = 0.5$

Numerical results compare LRM, vibrato with one Z per W, and pathwise with conditional expectation.

Vibrato Monte Carlo



Multivariate extension

Can also treat payoffs dependent on $S(\tau)$ at intermediate times, by taking

$$t_n < \tau < t_{n+1}$$

and using simple Brownian motion interpolation between $\widehat{S}_n$ and $\widehat{S}_{n+1}$ to get a Normal distribution for $\widehat{S}(\tau)$, with

mean: $$\widehat{S}_n + \frac{\tau - t_n}{t_{n+1} - t_n} \left(\widehat{S}_{n+1} - \widehat{S}_n \right)$$

variance: $$\frac{(\tau - t_n)(t_{n+1} - \tau)}{t_{n+1} - t_n} \, b^2(\widehat{S}_n, t_n)$$

Conclusions

Monte Carlo estimation of sensitivities is an important problem in computational finance

Improved methods need ideas from both mathematics

- adjoint technique
- vibrato Monte Carlo
- (multilevel Monte Carlo)

. . . and computer science

- automatic differentiation

Multilevel Monte Carlo

The objective is to

- achieve a user-specified accuracy
- at a reduced computational cost
- through combining Monte Carlo simulations with different numbers of timesteps

The idea came from experience with multigrid in the iterative solution of finite difference equations, but the details are completely different.

Generic Problem

Stochastic differential equation with general drift and volatility terms:

$$\mathrm{d}S(t) = a(S,t)\,\mathrm{d}t + b(S,t)\,\mathrm{d}W(t)$$

For simple European options, we want to estimate the expected value of an option dependent on the terminal state

$$P = f(S(T))$$

with a uniform Lipschitz bound,

$$|f(U) - f(V)| \ \leq c\,\|U - V\|\,, \quad \forall\, U, V.$$

Standard MC Approach

Euler discretisation with timestep h:

$$\widehat{S}_{n+1} = \widehat{S}_n + a(\widehat{S}_n, t_n)\, h + b(\widehat{S}_n, t_n)\, \Delta W_n$$

Simplest estimator for expected payoff is an average of N independent path simulations:

$$\widehat{Y} = N^{-1} \sum_{i=1}^{N} f(\widehat{S}^{(i)}_{T/h})$$

- weak convergence – $O(h)$ error in expected payoff
- strong convergence – $O(h^{1/2})$ error in individual paths

Standard MC Approach

Mean Square Error is $O\left(N^{-1} + h^2\right)$

- first term comes from variance of estimator
- second term comes from bias due to weak convergence

To make this $O(\varepsilon^2)$ requires

$$N = O(\varepsilon^{-2}), \quad h = O(\varepsilon) \quad \Longrightarrow \quad \text{cost} = O(N\,h^{-1}) = O(\varepsilon^{-3})$$

Aim is to improve this cost to $O\left(\varepsilon^{-2}(\log\varepsilon)^2\right)$, by combining simulations with different numbers of timesteps – same accuracy as finest calculations, but at a much lower computational cost.

Other work

- Many variance reduction techniques to greatly reduce the cost, but without changing the order
- Richardson extrapolation improves the weak convergence and hence the order
- Multilevel method is a generalisation of two-level control variate method of Kebaier (2005), and similar to ideas of Speight (2009)
- Also related to multilevel parametric integration by Heinrich (2001)

Multilevel MC Approach

Consider multiple sets of simulations with different timesteps $h_l = 2^{-l}\,T,\ l = 0, 1, \ldots, L$, and payoff $\widehat{P}_l$

$$\mathbb{E}[\widehat{P}_L] = \mathbb{E}[\widehat{P}_0] + \sum_{l=1}^{L} \mathbb{E}[\widehat{P}_l - \widehat{P}_{l-1}]$$

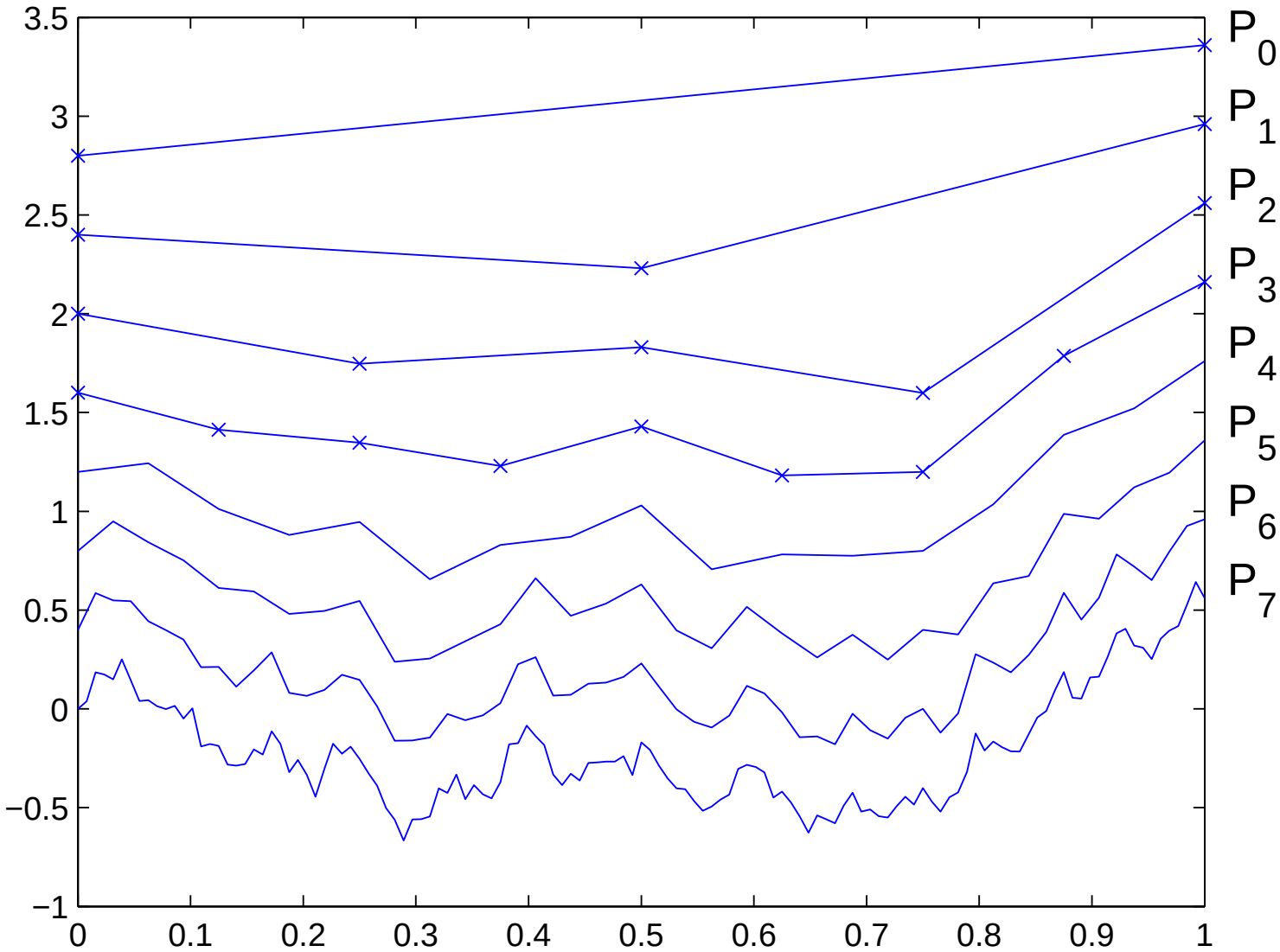
Expected value is same – aim is to reduce variance of estimator for a fixed computational cost.

Key point: approximate $\mathbb{E}[\widehat{P}_l - \widehat{P}_{l-1}]$ using N_l simulations with $\widehat{P}_l$ and $\widehat{P}_{l-1}$ obtained using same Brownian path.

$$\widehat{Y}_l = N_l^{-1} \sum_{i=1}^{N_l} \left(\widehat{P}_l^{(i)} - \widehat{P}_{l-1}^{(i)} \right)$$

Multilevel MC Approach

Discrete Brownian path at different levels



Multilevel MC Approach

- each level adds more detail to Brownian path and reduces the error in the numerical integration
- $\mathbb{E}[\widehat{P}_l - \widehat{P}_{l-1}]$ reflects impact of that extra detail on the payoff
- different timescales handled by different levels – similar to different wavelengths being handled by different grids in multigrid solvers for iterative solution of PDEs

Multilevel MC Approach

Using independent paths for each level, the variance of the combined estimator is

$$\mathbb{V}\left[\sum_{l=0}^{L}\widehat{Y}_l\right]=\sum_{l=0}^{L}N_l^{-1}V_l,\qquad V_l\equiv\mathbb{V}[\widehat{P}_l-\widehat{P}_{l-1}],$$

and the computational cost is proportional to $\displaystyle\sum_{l=0}^{L}N_l\,h_l^{-1}$.

Hence, the variance is minimised for a fixed computational cost by choosing N_l to be proportional to $\sqrt{V_l\,h_l}$.

The constant of proportionality can be chosen so that the combined variance is $O(\varepsilon^2)$.

Multilevel MC Approach

For the Euler discretisation and the Lipschitz payoff function

$$\mathbb{V}[\widehat{P}_l - P] = O(h_l) \quad \Longrightarrow \quad \mathbb{V}[\widehat{P}_l - \widehat{P}_{l-1}] = O(h_l)$$

and the optimal N_l is asymptotically proportional to h_l.

To make the combined variance $O(\varepsilon^2)$ requires

$$N_l = O(\varepsilon^{-2} L\, h_l).$$

To make the bias $O(\varepsilon)$ requires

$$L = \log_2 \varepsilon^{-1} + O(1) \quad \Longrightarrow \quad h_L = O(\varepsilon).$$

Hence, we obtain an $O(\varepsilon^2)$ MSE for a computational cost which is $O(\varepsilon^{-2} L^2) = O(\varepsilon^{-2} (\log \varepsilon)^2)$.

MLMC Results

Geometric Brownian motion:

$$\mathrm{d}S = r\,S\,\mathrm{d}t + \sigma\,S\,\mathrm{d}W, \qquad 0 < t < T,$$

$$T = 1, \quad S(0) = 100, \quad r = 0.05, \quad \sigma = 0.2$$

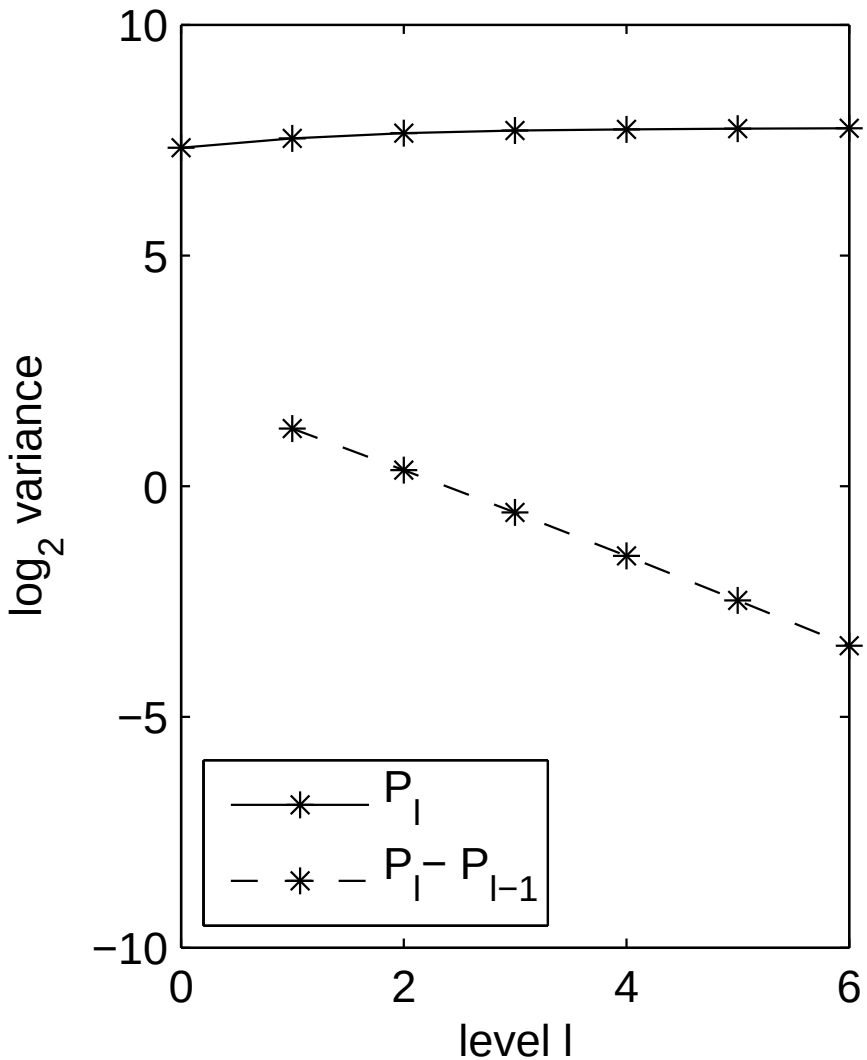
European call option with discounted payoff

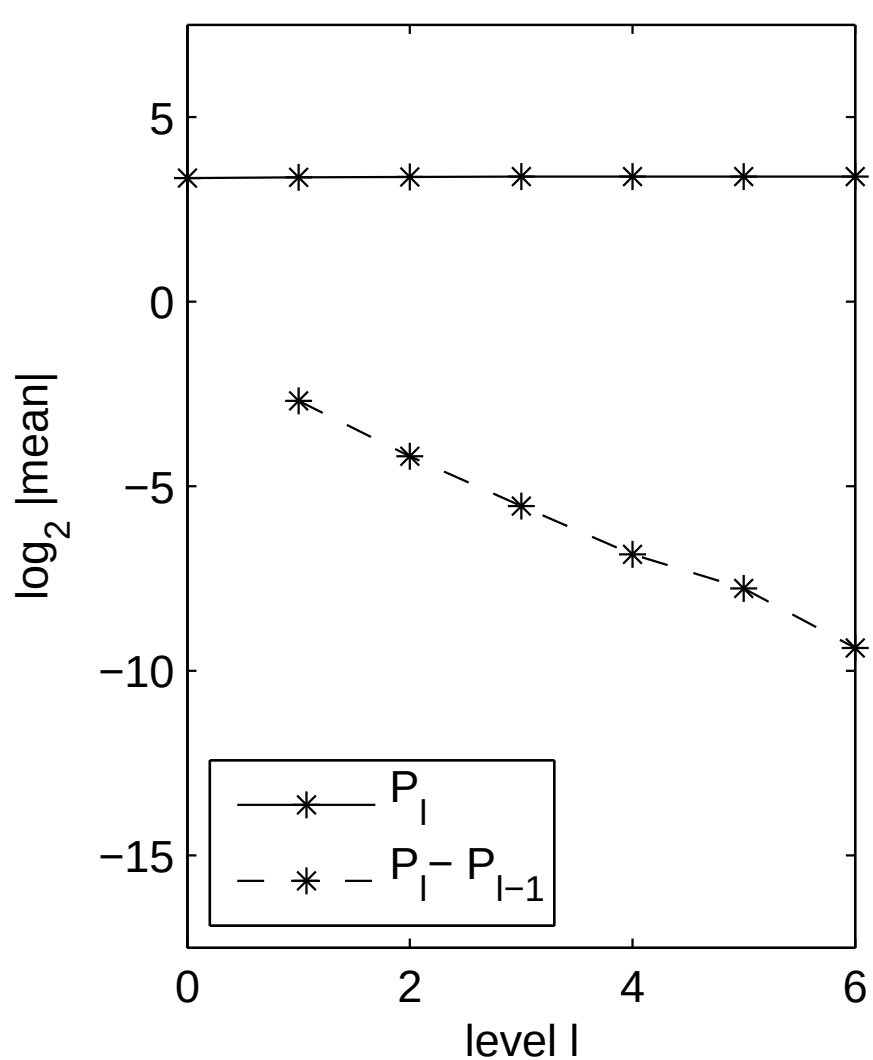
$$\exp(-rT)\ \max(S(T) - K, 0)$$

with strike $K = 100$.

MLMC Results

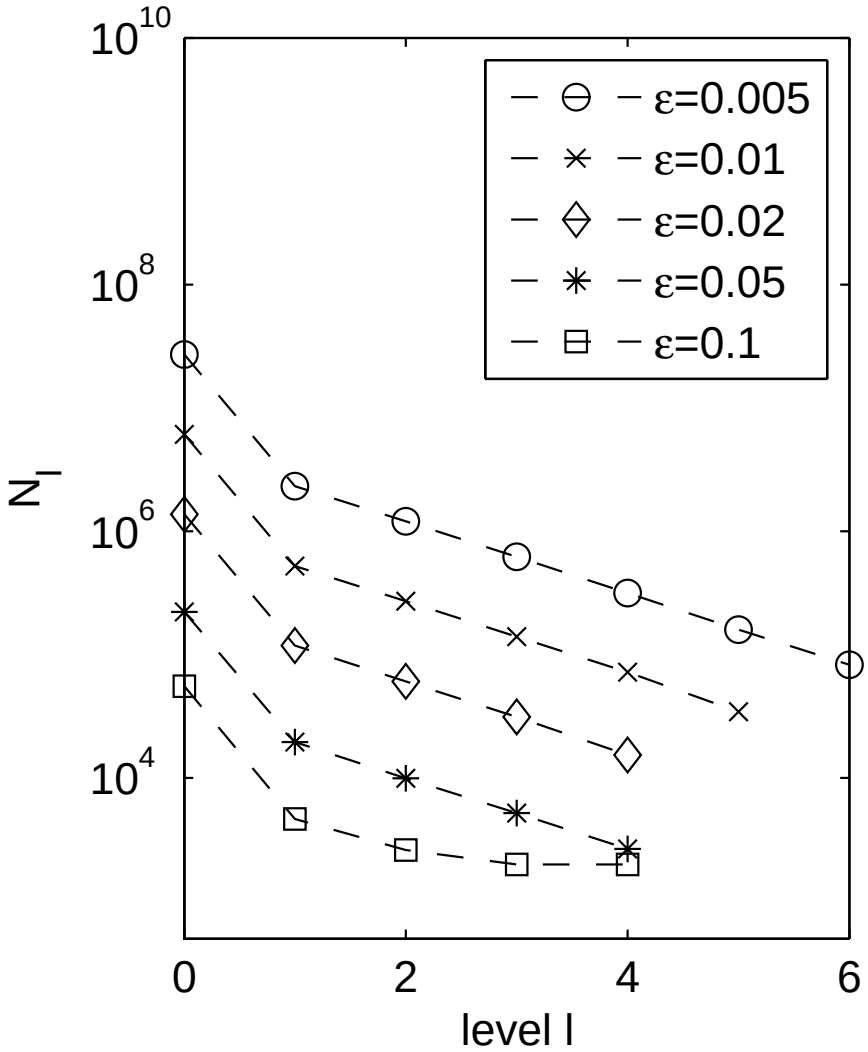
GBM: European call, $\exp(-rT)\ \max(S(T)-K,0)$

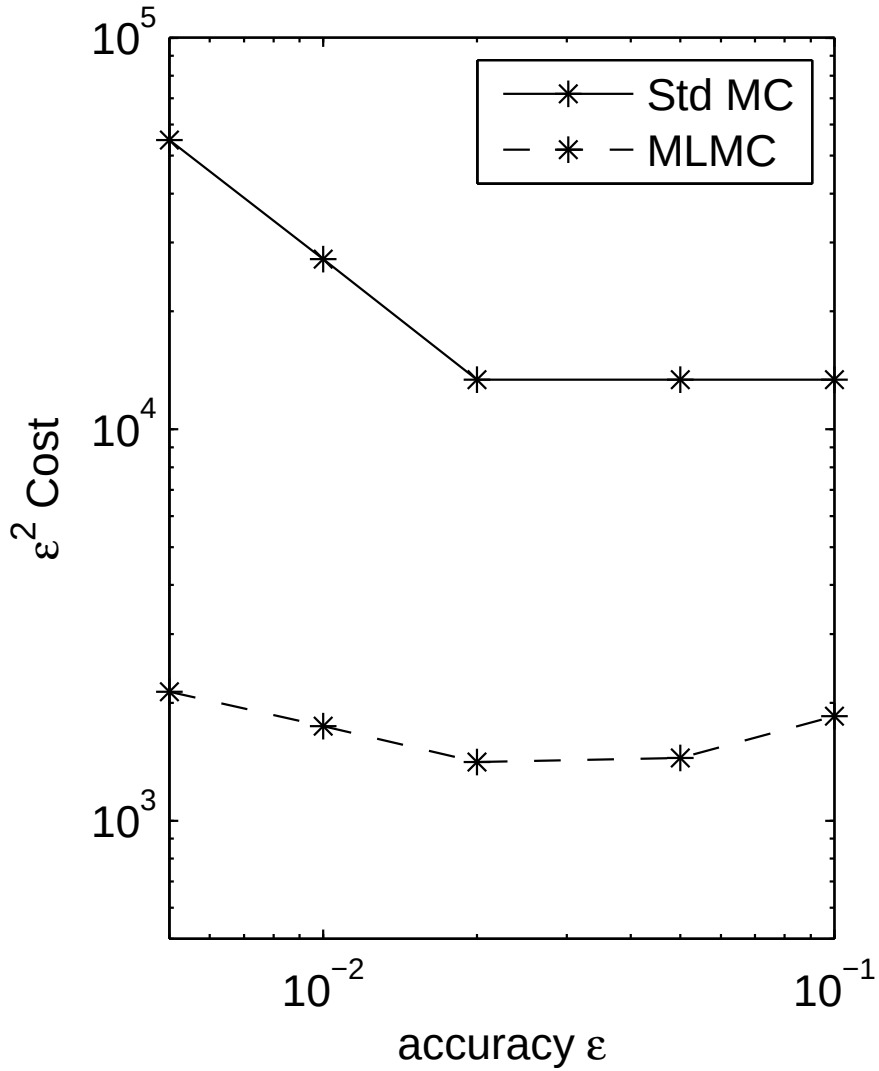




MLMC Results

GBM: European call, $\exp(-rT)\ \max(S(T)-K,0)$





MLMC Approach

So far, have kept things very simple:

- European option
- Euler discretisation
- single underlying in example

We now generalise it:

- arbitrary path-dependent options
- arbitrary discretisation
- assume certain properties for weak convergence and variance of multilevel correction
- obtain order of cost to achieve r.m.s. accuracy ε

MLMC Approach

Theorem: *Let P be a functional of the solution of a stochastic o.d.e., and $\widehat{P}_l$ the discrete approximation using a timestep $h_l = 2^{-l}\,T$.*

If *there exist independent estimators $\widehat{Y}_l$ based on N_l Monte Carlo samples, with computational complexity (cost) C_l, and positive constants $\alpha \geq \frac{1}{2}, \beta, c_1, c_2, c_3$ such that*

i) $\left|\mathbb{E}[\widehat{P}_l - P]\right| \leq c_1\, h_l^{\alpha}$

ii) $\mathbb{E}[\widehat{Y}_l] = \begin{cases} \mathbb{E}[\widehat{P}_0], & l = 0 \\ \mathbb{E}[\widehat{P}_l - \widehat{P}_{l-1}], & l > 0 \end{cases}$

iii) $\mathbb{V}[\widehat{Y}_l] \leq c_2\, N_l^{-1} h_l^{\beta}$

iv) $C_l \leq c_3\, N_l\, h_l^{-1}$

Multilevel MC Approach

then *there exists a positive constant* c_4 *such that for any* $\varepsilon < e^{-1}$ *there are values* L *and* N_l *for which the multilevel estimator*

$$\widehat{Y} = \sum_{l=0}^{L} \widehat{Y}_l,$$

has Mean Square Error $MSE \equiv \mathbb{E}\left[\left(\widehat{Y} - \mathbb{E}[P]\right)^2\right] < \varepsilon^2$

with a computational complexity C *with bound*

$$C \leq \begin{cases} c_4\, \varepsilon^{-2}, & \beta > 1, \\ c_4\, \varepsilon^{-2} (\log \varepsilon)^2, & \beta = 1, \\ c_4\, \varepsilon^{-2-(1-\beta)/\alpha}, & 0 < \beta < 1. \end{cases}$$

Milstein Scheme

The theorem suggests use of Milstein approximation – better strong convergence, same weak convergence

Generic scalar SDE:

$$\mathrm{d}S(t) = a(S,t)\,\mathrm{d}t + b(S,t)\,\mathrm{d}W(t), \quad 0<t<T.$$

Milstein scheme:

$$\widehat{S}_{n+1} = \widehat{S}_n + a\,h + b\,\Delta W_n + \tfrac{1}{2}\,b'\,b\left((\Delta W_n)^2 - h\right).$$

Milstein Scheme

In scalar case:

- $O(h)$ strong convergence
- $O(\varepsilon^{-2})$ complexity for Lipschitz payoffs – trivial
- $O(\varepsilon^{-2})$ complexity for more complex cases using carefully constructed estimators based on Brownian interpolation or extrapolation
  - digital, with discontinuous payoff
  - Asian, based on average
  - lookback and barrier, based on min/max
- This extends naturally to basket options based on a weighted average of assets linked only through the correlation in the driving Brownian motion

Milstein Scheme

Brownian interpolation: within each timestep, model the behaviour as simple Brownian motion conditional on the two end-points

$$\begin{aligned}\widehat{S}(t) &= \widehat{S}_n + \lambda(t)(\widehat{S}_{n+1} - \widehat{S}_n) \\ &\quad + b_n \left(W(t) - W_n - \lambda(t)(W_{n+1} - W_n) \right),\end{aligned}$$

where

$$\lambda(t) = \frac{t - t_n}{t_{n+1} - t_n}$$

There then exist analytic results for the distribution of the min/max/average over each timestep, and probability of crossing a barrier.

Milstein Scheme

Brownian extrapolation for final timestep:

$$\widehat{S}_N = \widehat{S}_{N-1} + a_{N-1}h + b_{N-1}\Delta W_N$$

Considering all possible ΔW_N gives Gaussian distribution, for which a digital option has a known conditional expectation – example in Glasserman's book of payoff smoothing to allow pathwise calculation of Greeks.

This payoff smoothing can be extended to general multivariate cases (not just baskets) through a "vibrato" Monte Carlo technique which is suitable for both efficient multilevel analysis and the computation of Greeks

MLMC Results

Basket of 5 underlying assets, each GBM with

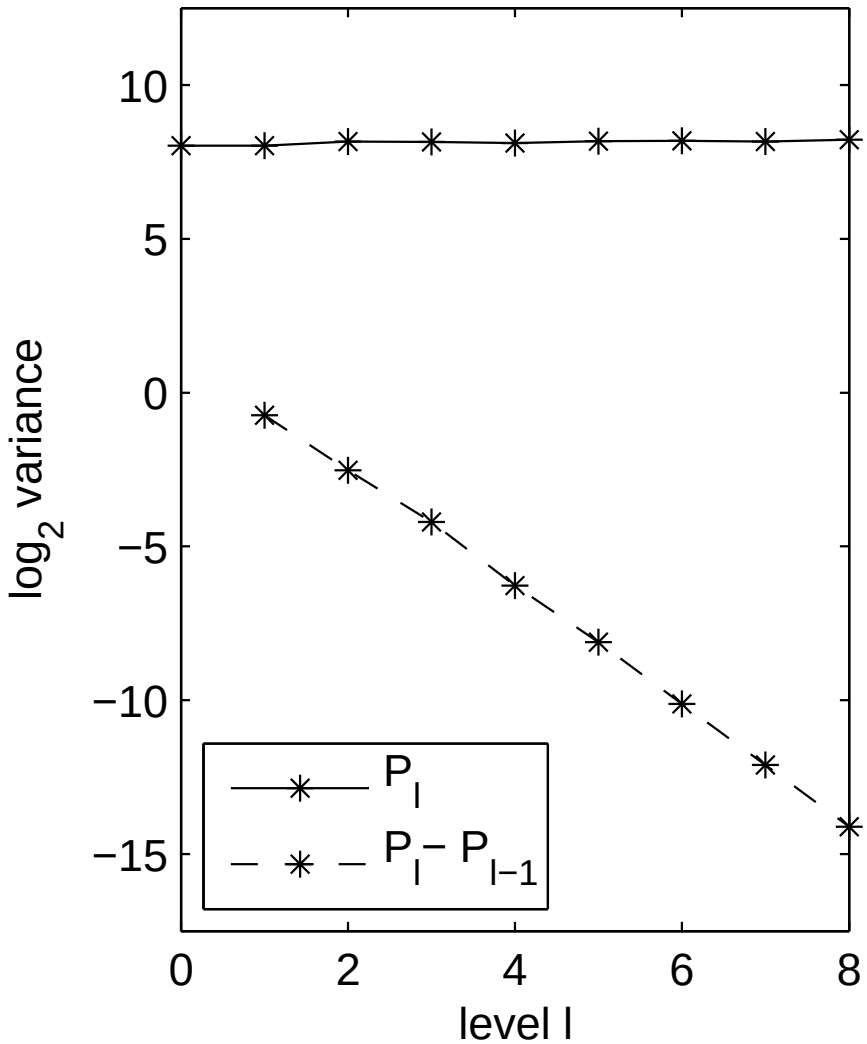
$$r = 0.05, \quad T = 1, \quad S_i(0) = 100, \quad \sigma = (0.2,\, 0.25,\, 0.3,\, 0.35,\, 0.4),$$

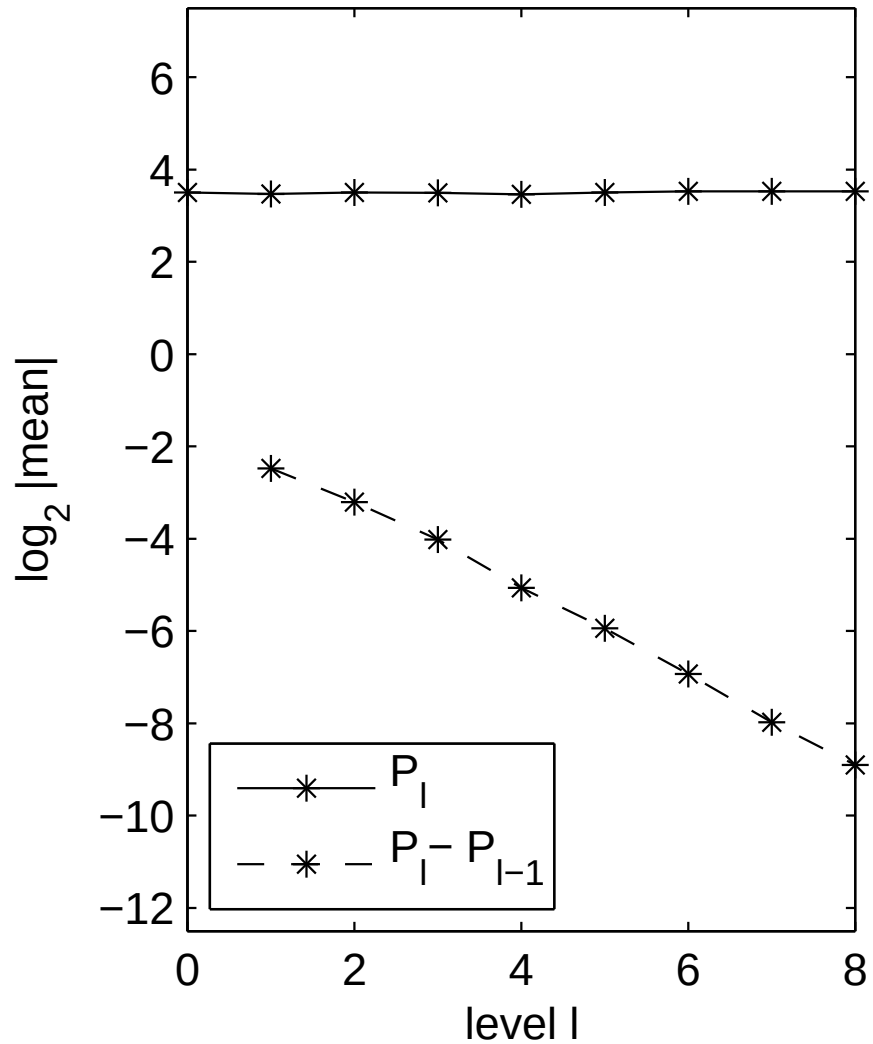
and correlation $\rho = 0.25$ between each of the driving Brownian motions.

All options are based on arithmetic average $\overline{S}$ of 5 assets, with strike $K = 100$ (and barrier $B = 85$).

MLMC Results

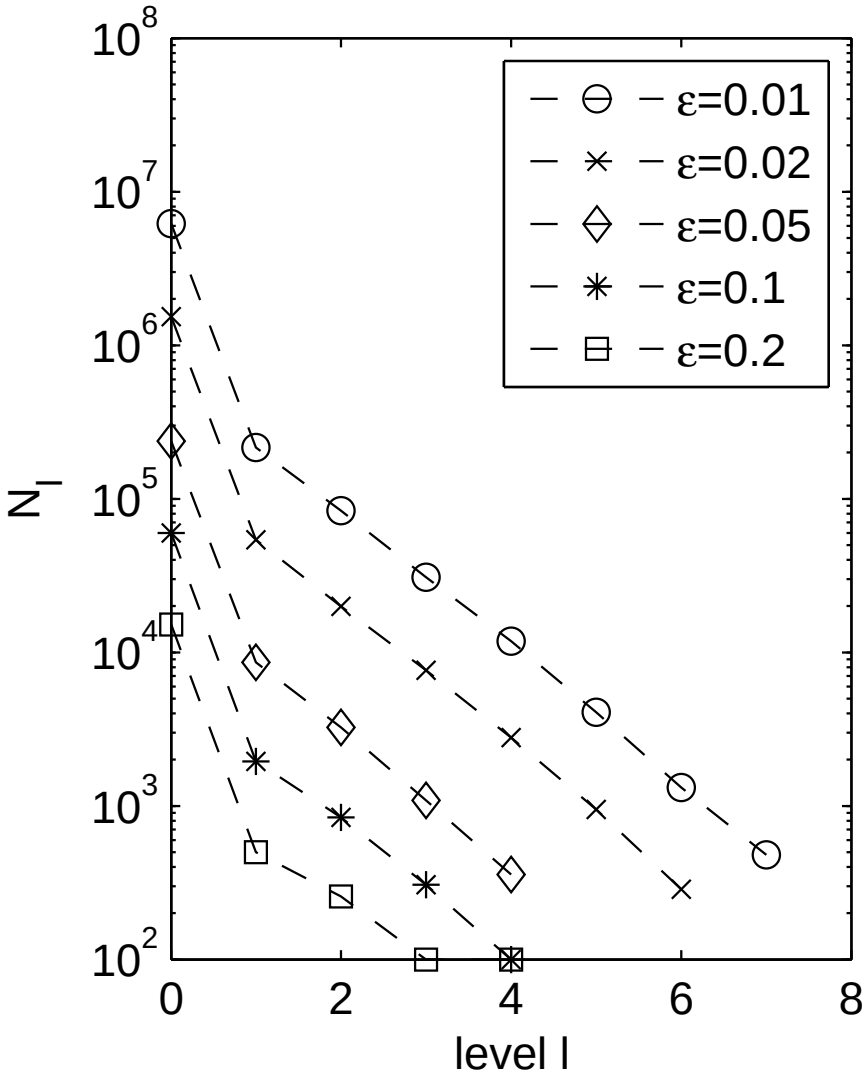
European call, $\exp(-rT)\ \max(\overline{S}(T) - K, 0)$

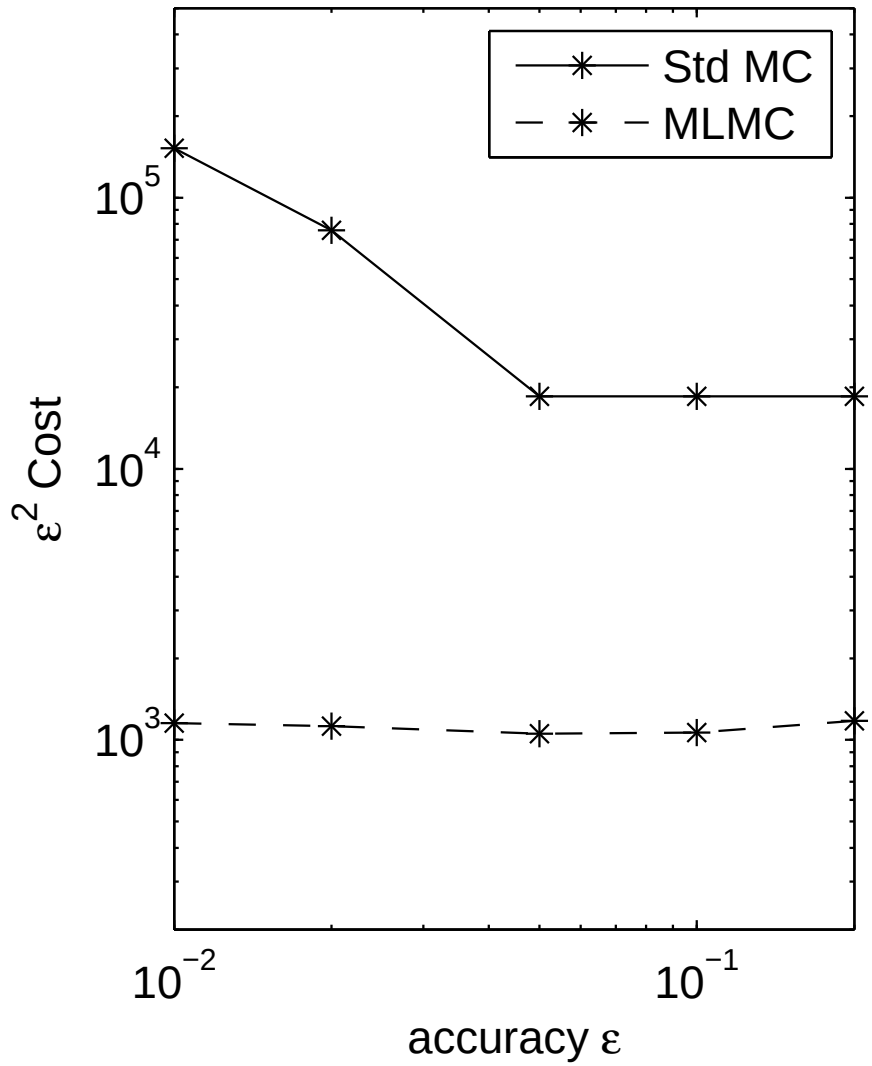




MLMC Results

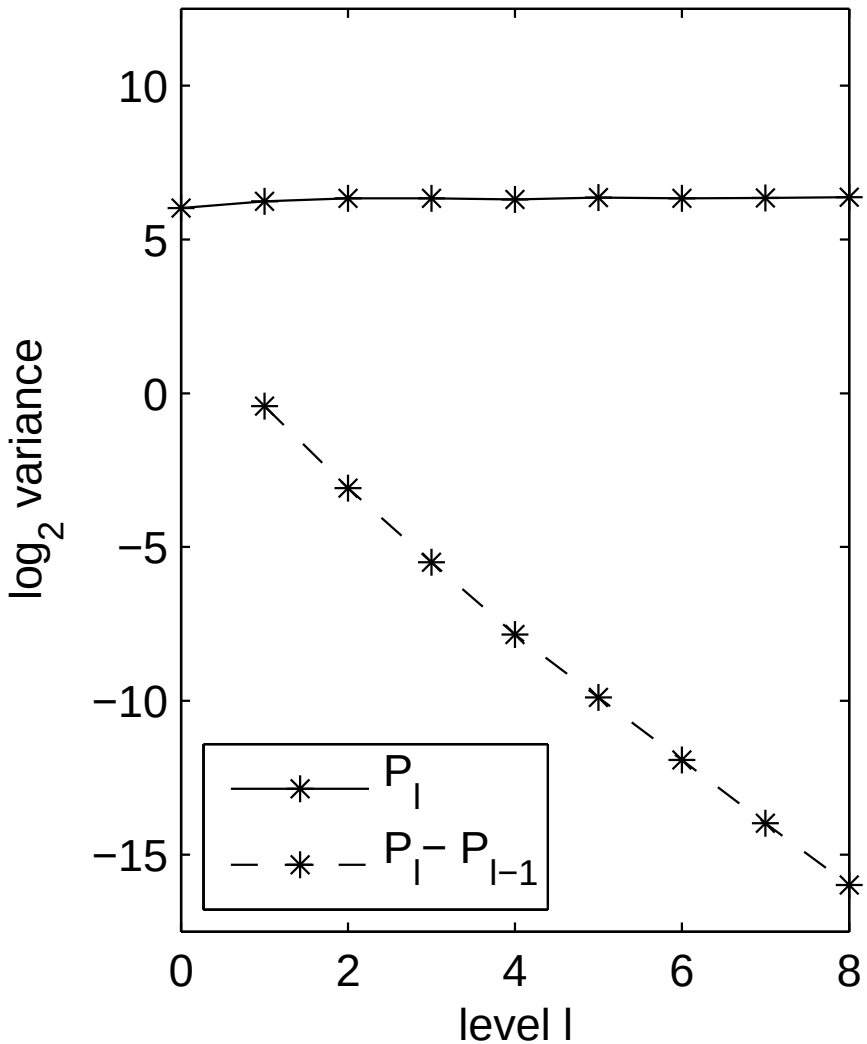
European call, $\exp(-rT)\ \max(\overline{S}(T) - K, 0)$

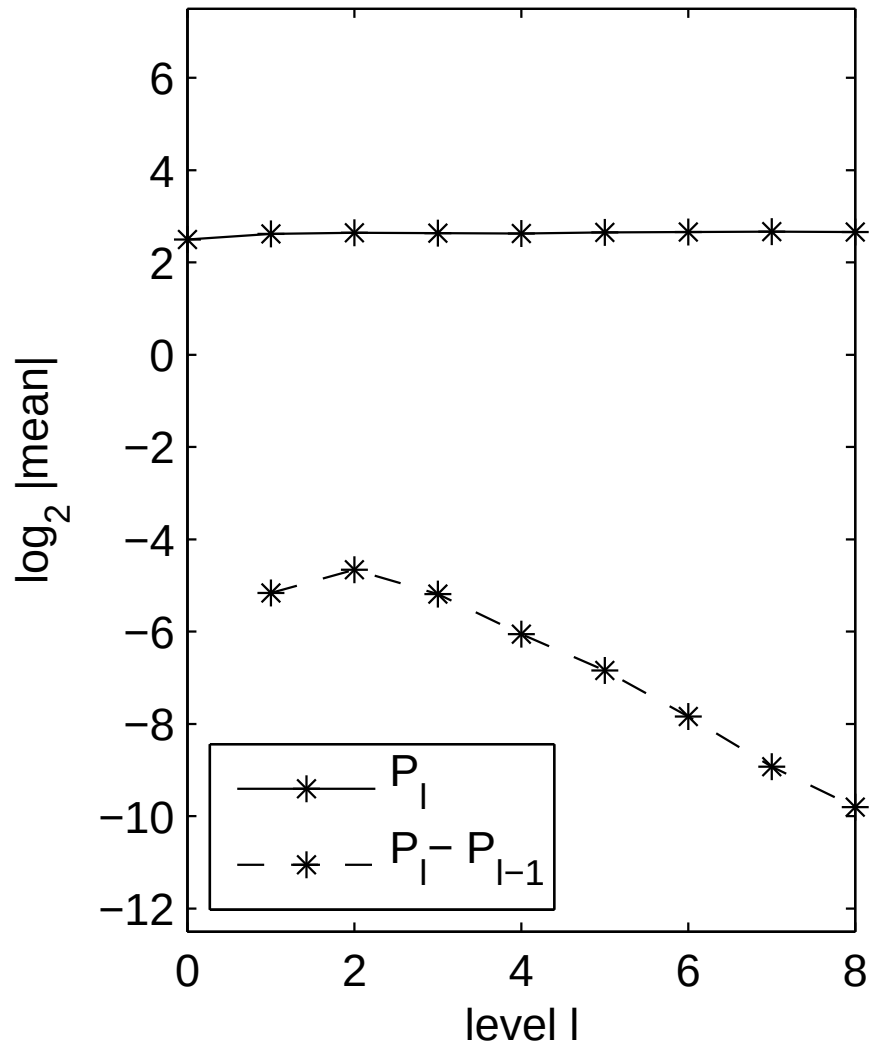




MLMC Results

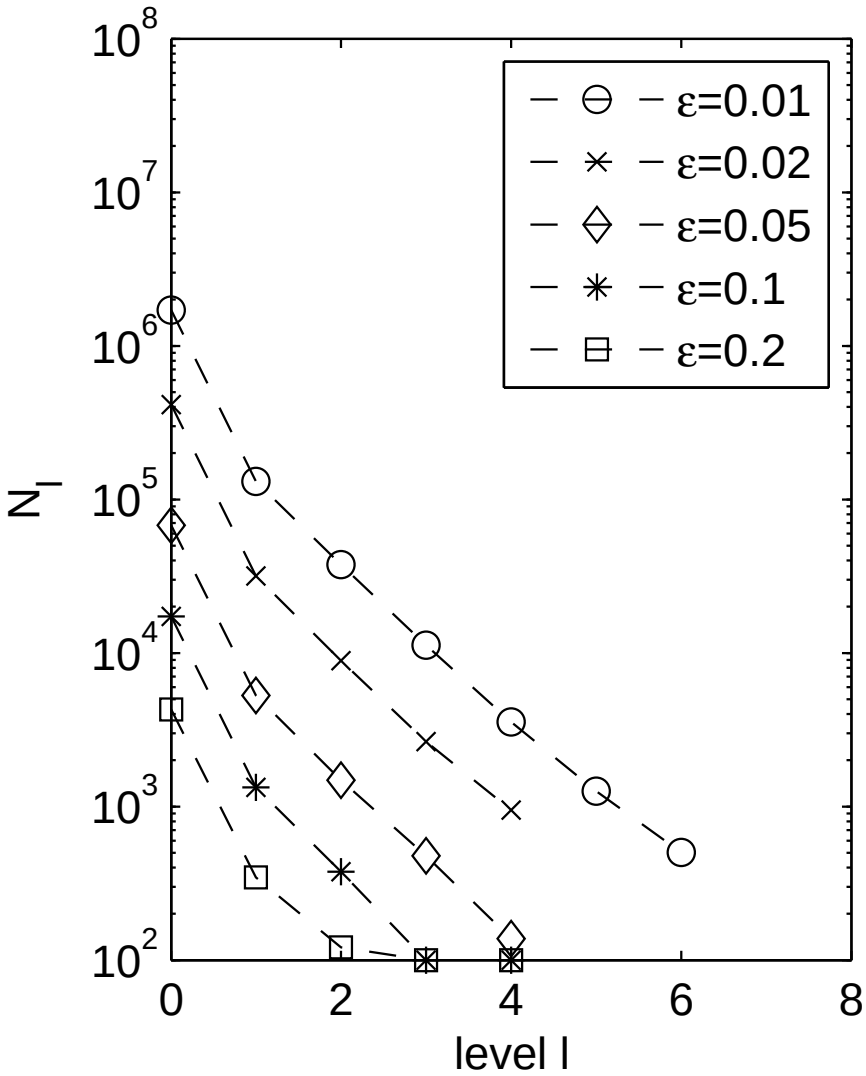
Asian option, $\exp(-rT)\ \max(T^{-1}\int_0^T \overline{S}(t)\,\mathrm{d}t - K, 0)$

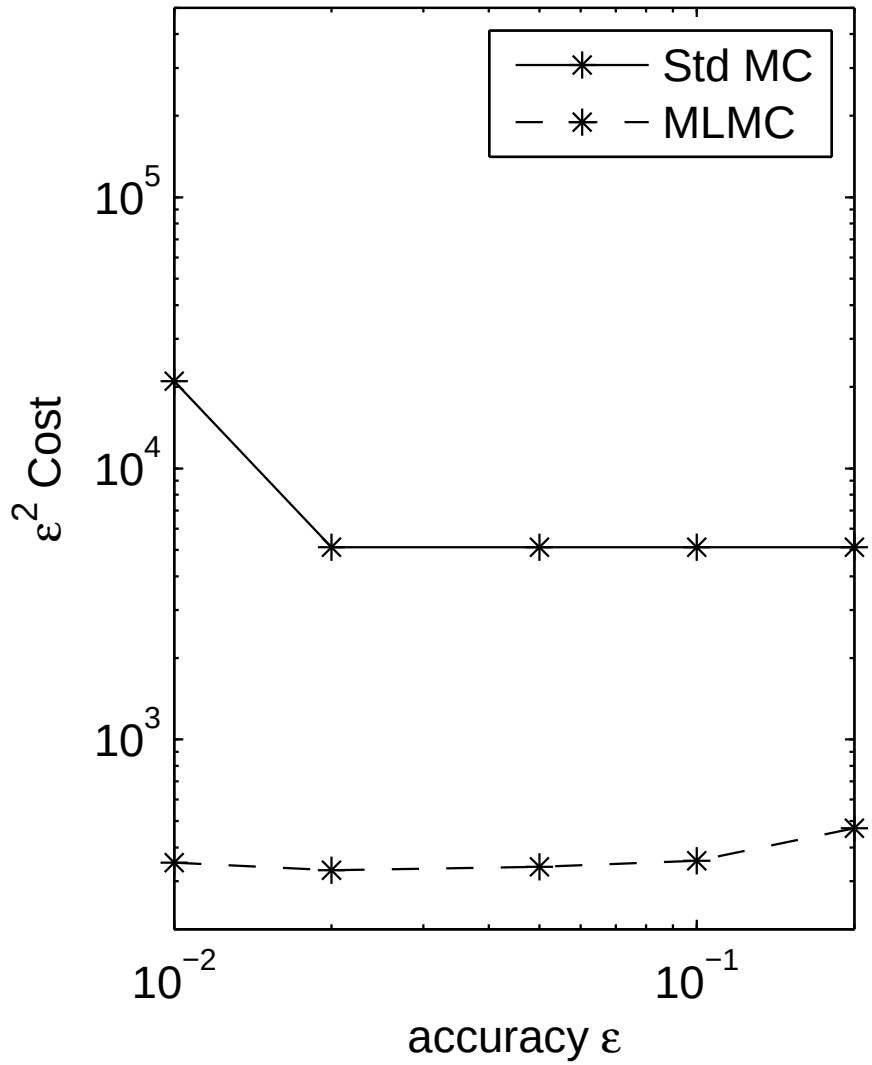




MLMC Results

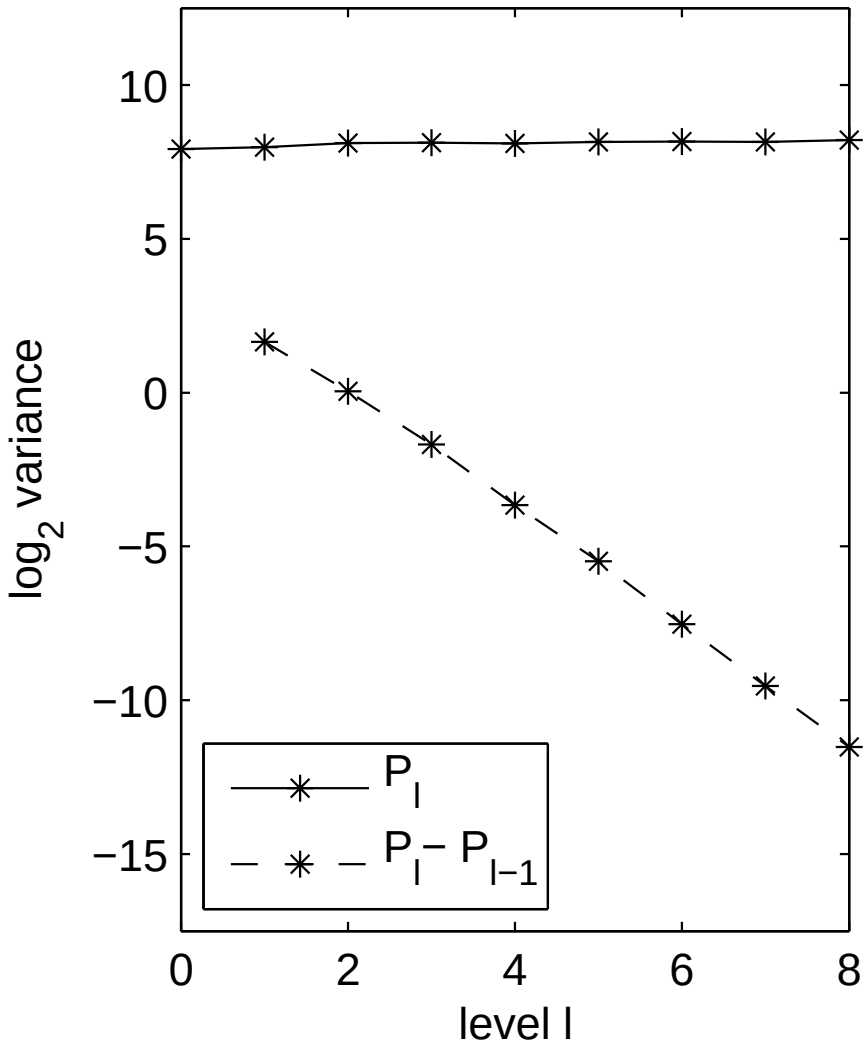
Asian option, $\exp(-rT)\ \max(T^{-1}\int_0^T \overline{S}(t)\,\mathrm{d}t - K, 0)$

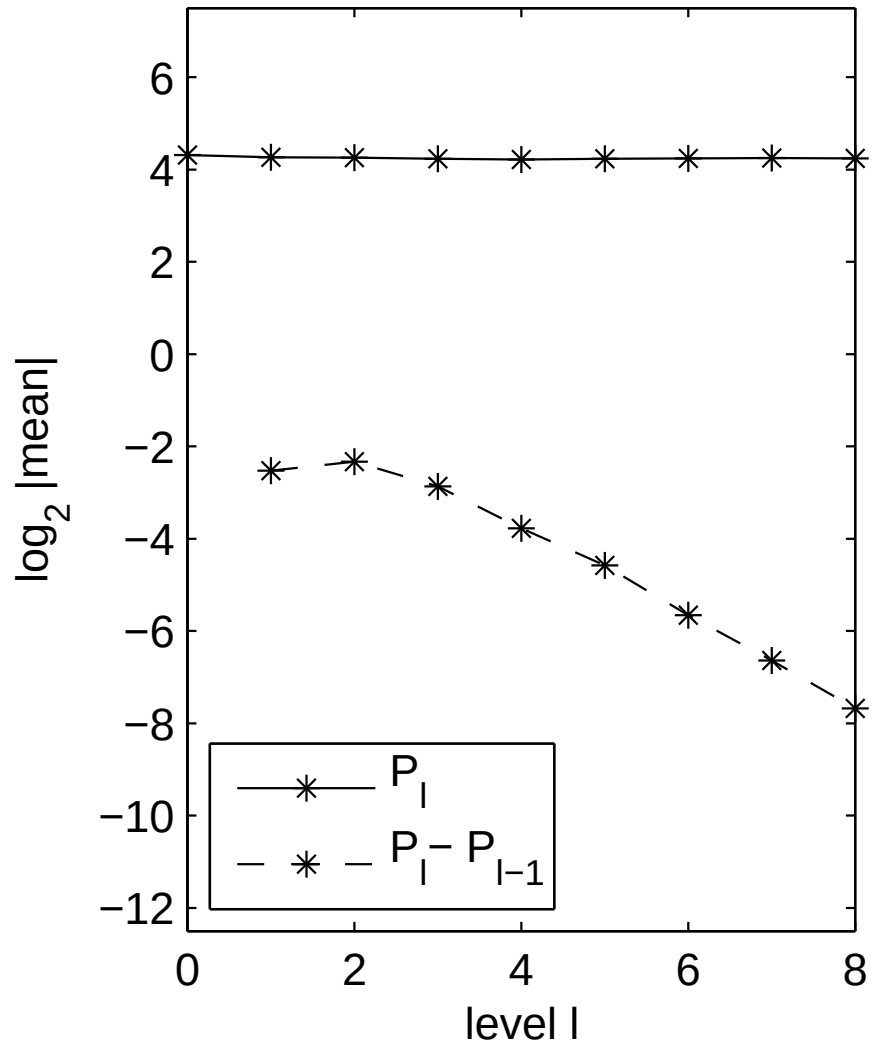




MLMC Results

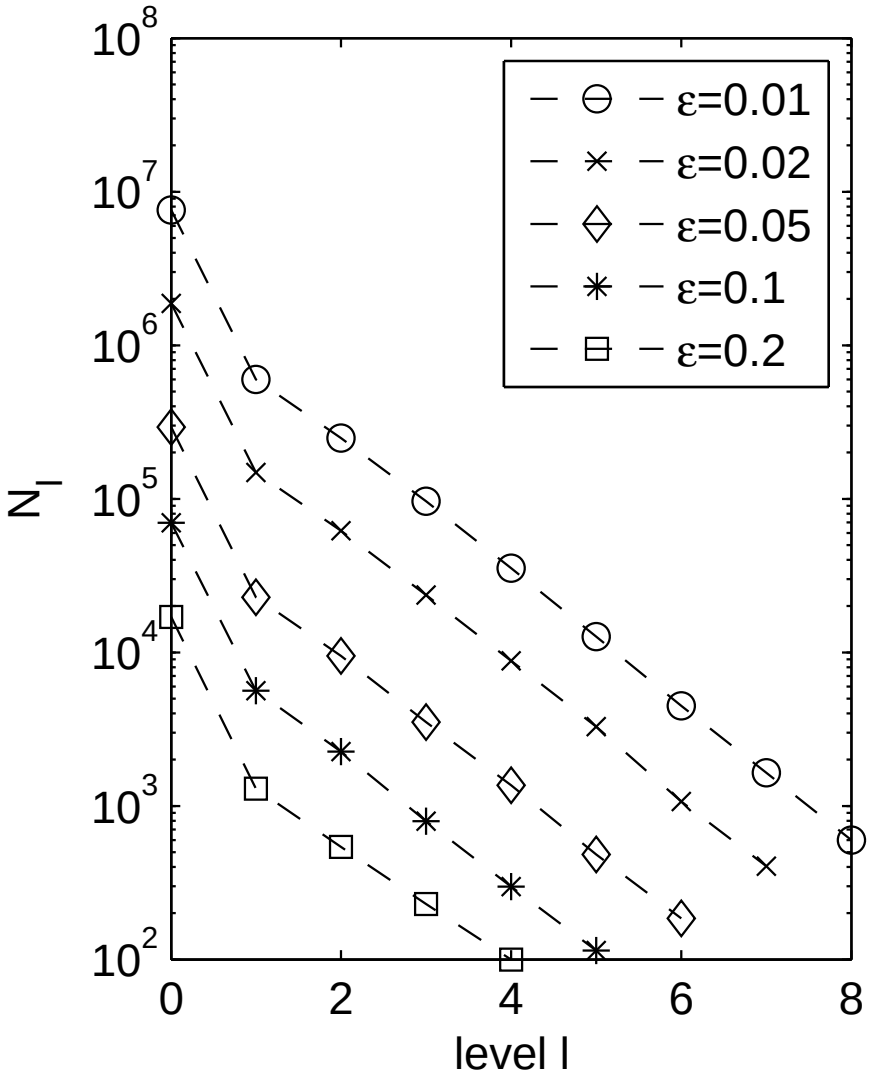
Lookback option, $\exp(-rT)\,(\overline{S}(T) - \min_{0<t<T} \overline{S}(t))$

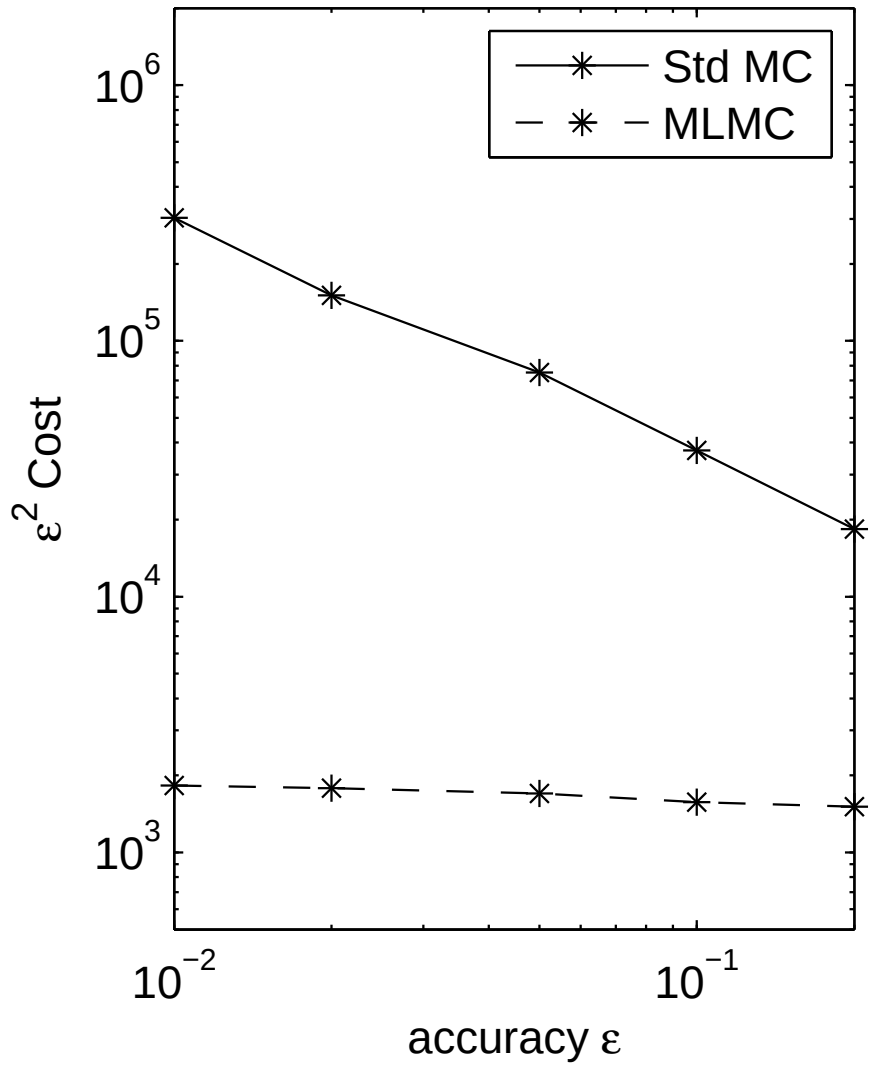




MLMC Results

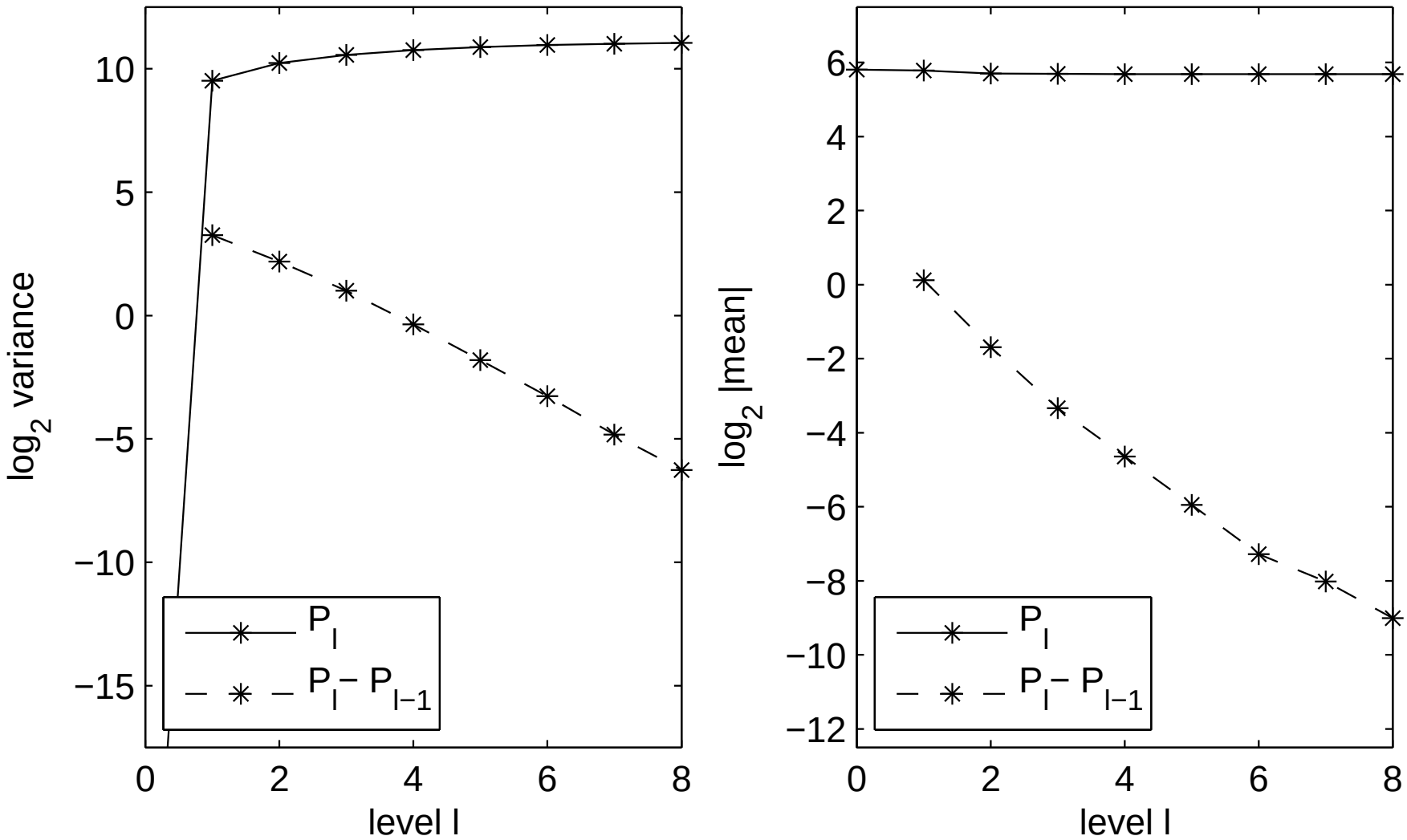
Lookback option, $\exp(-rT)\,(\overline{S}(T) - \min_{0<t<T} \overline{S}(t))$





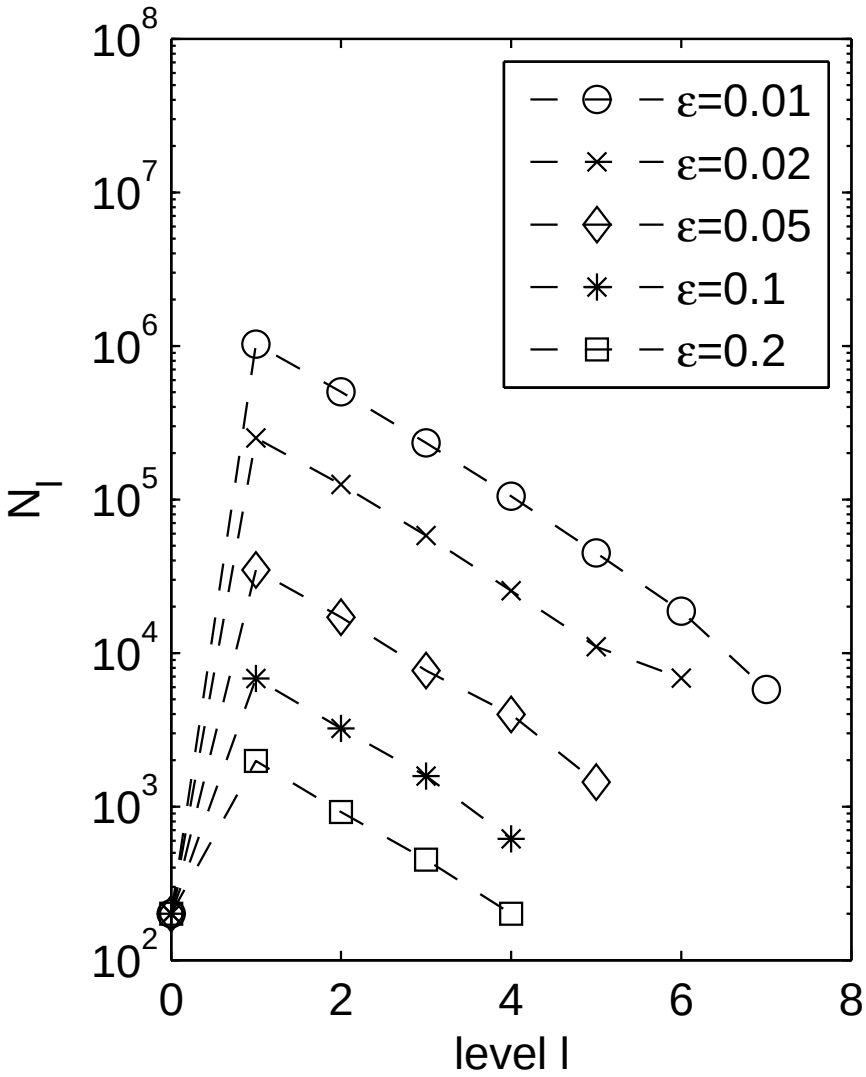
MLMC Results

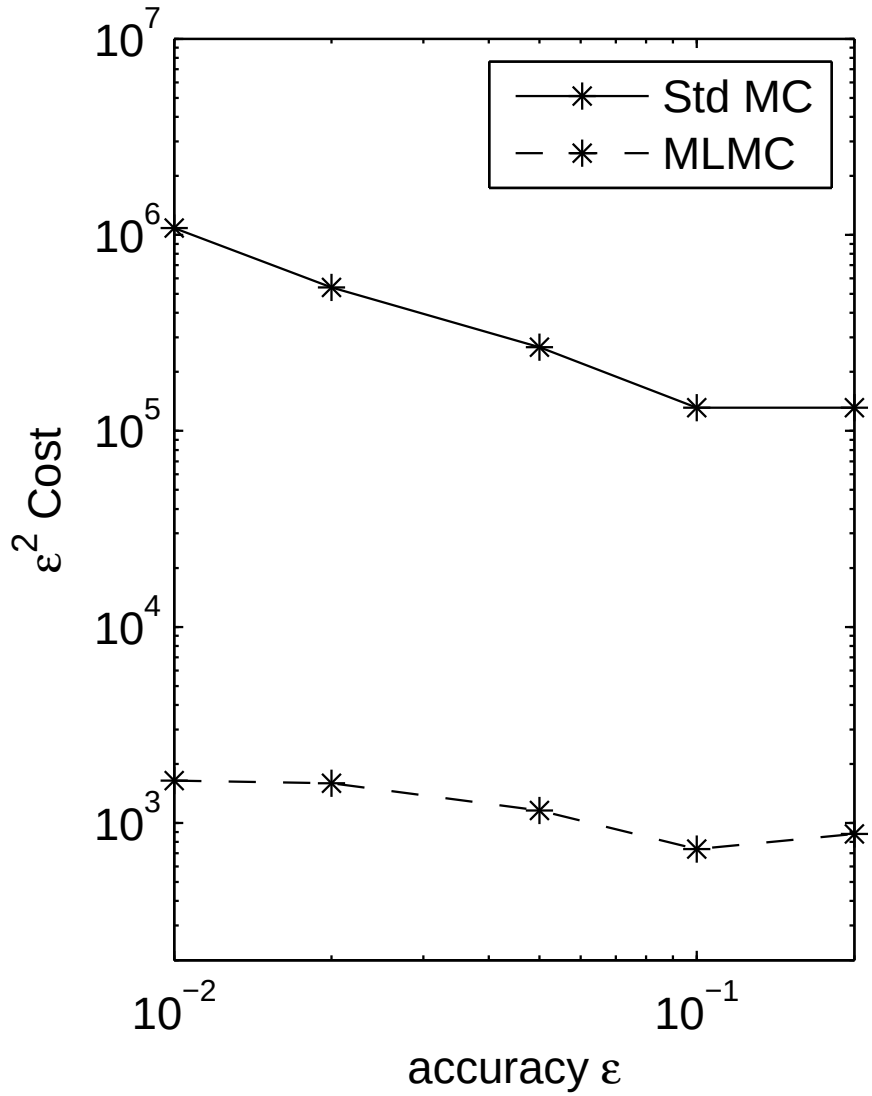
Digital option, $100\ \exp(-rT)\,\mathbf{1}_{\overline{S}(T)>K}$



MLMC Results

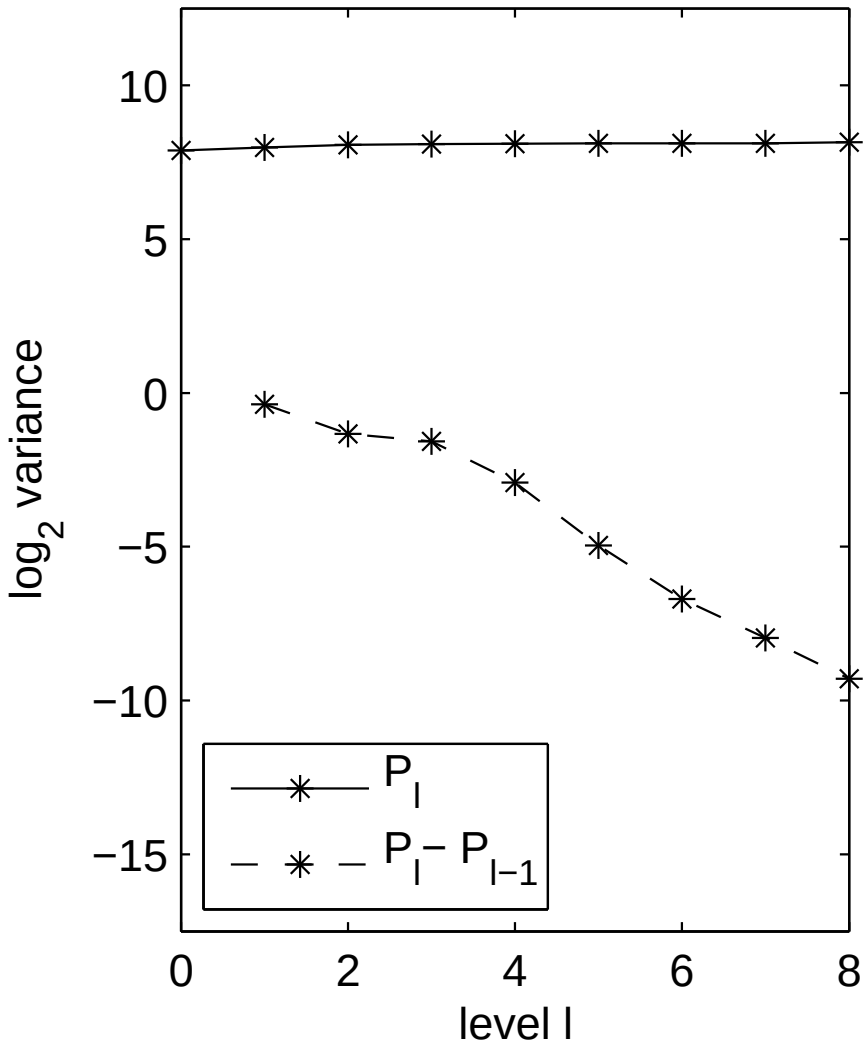
Digital option, $100 \exp(-rT) \mathbf{1}_{\overline{S}(T)>K}$

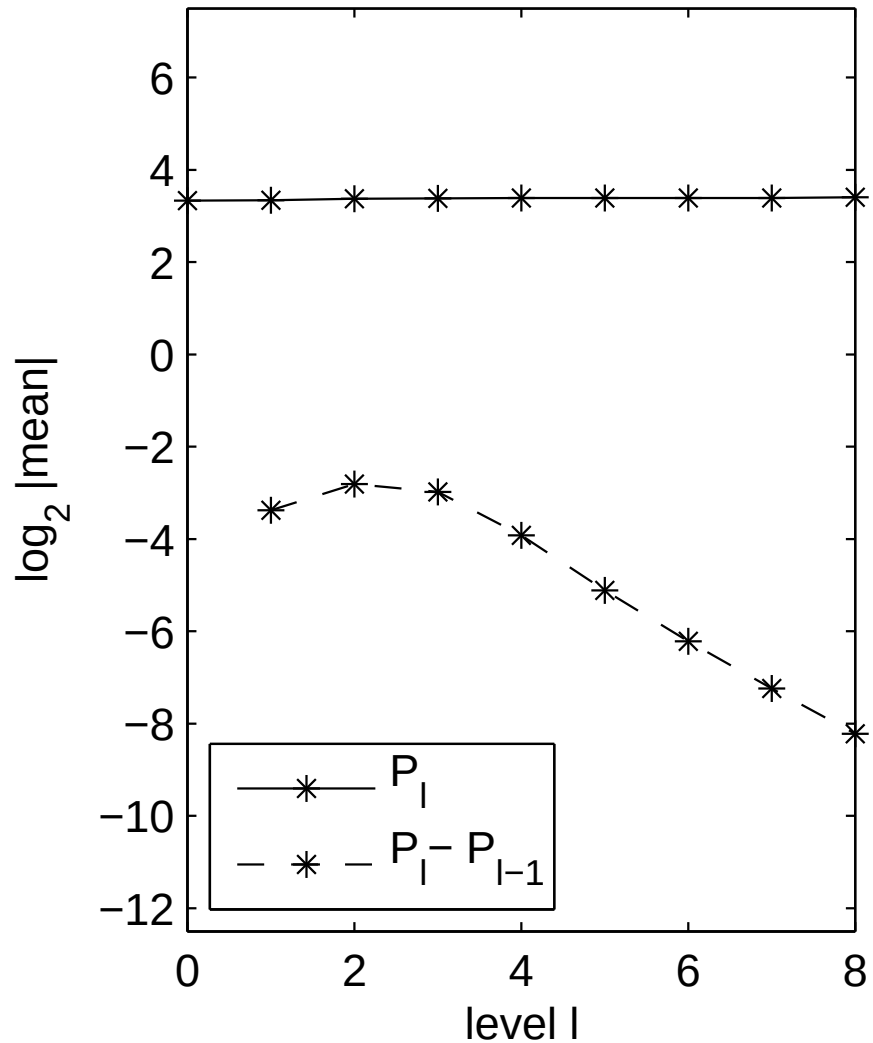




MLMC Results

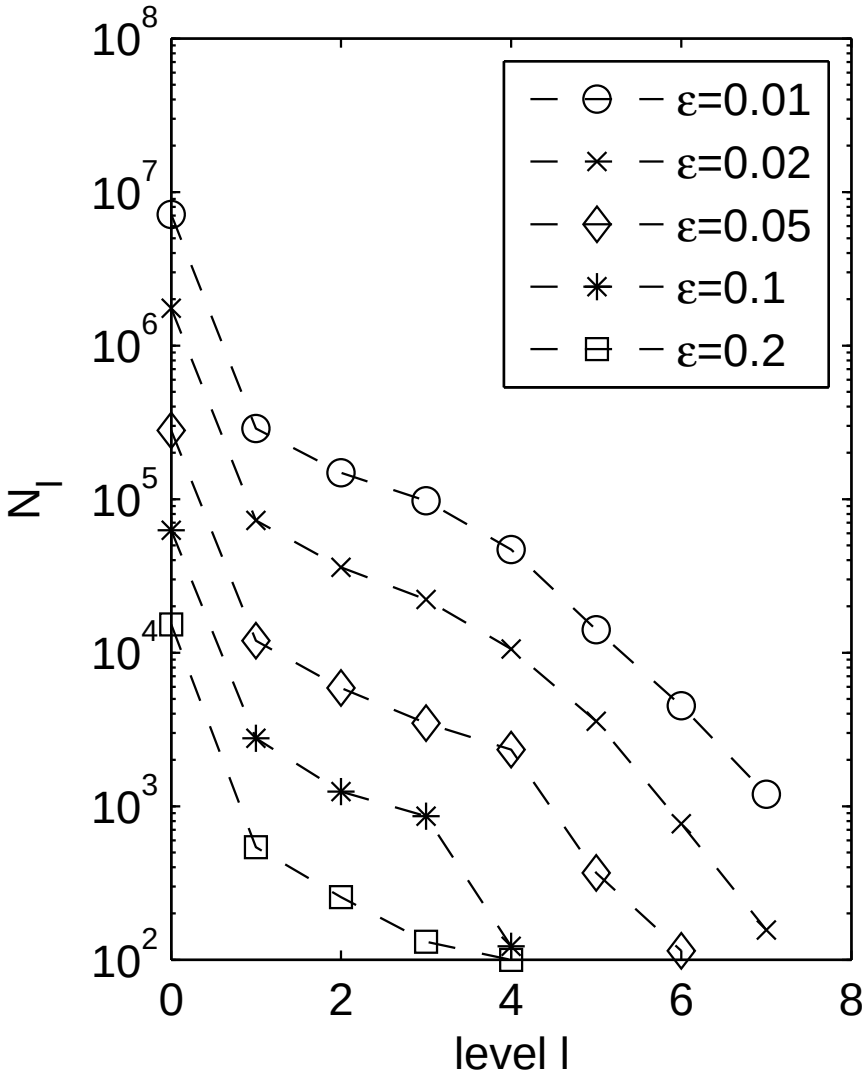
Barrier option, $\exp(-rT)\ \max(\overline{S}(T)-K,0)\ \mathbf{1}_{\min_{0<t<T}\overline{S}(t)>B}$

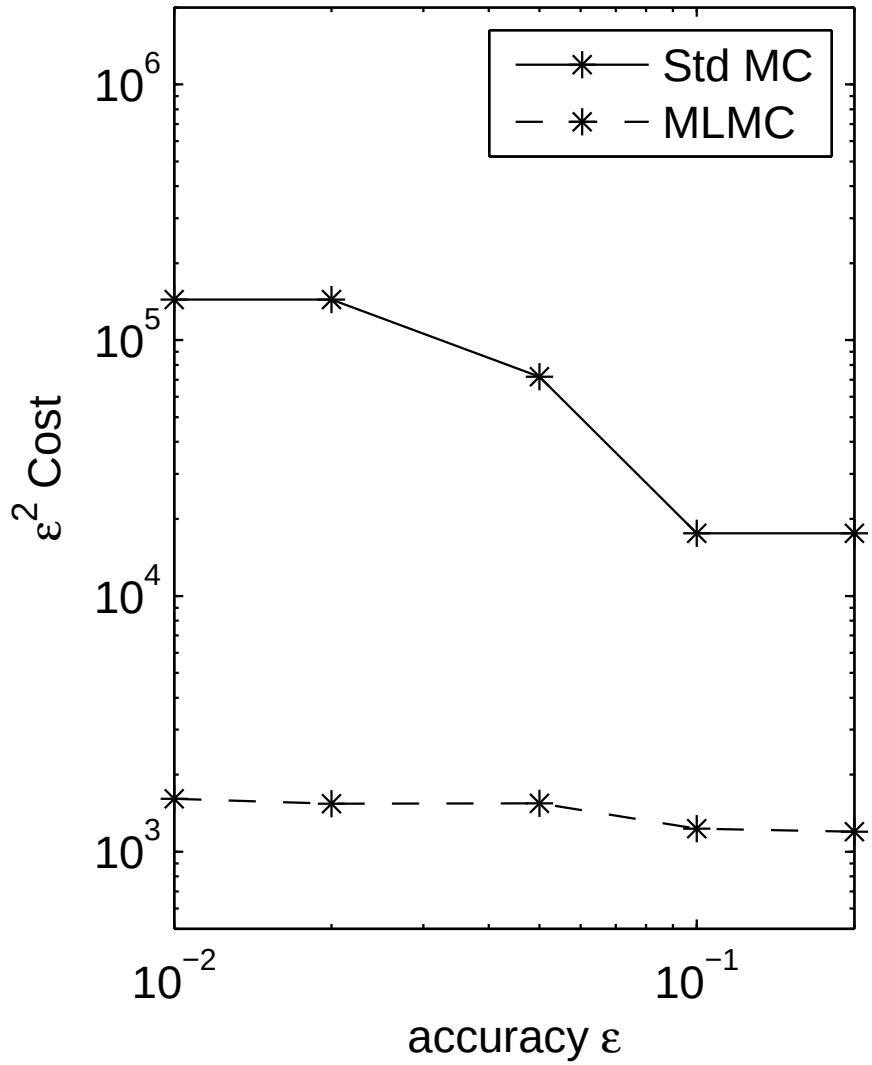




MLMC Results

Barrier option, $\exp(-rT)\ \max(\overline{S}(T)-K,0)\ \mathbf{1}_{\min_{0<t<T}\overline{S}(t)>B}$





MLMC Numerical Analysis

option	Euler		Milstein	
	numerics	analysis	numerics	analysis
Lipschitz	$O(h)$	$O(h)$	$O(h^2)$	$O(h^2)$
Asian	$O(h)$	$O(h)$	$O(h^2)$	$O(h^2)$
lookback	$O(h)$	$O(h)$	$O(h^2)$	$o(h^{2-\delta})$
barrier	$O(h^{1/2})$	$o(h^{1/2-\delta})$	$O(h^{3/2})$	$o(h^{3/2-\delta})$
digital	$O(h^{1/2})$	$O(h^{1/2}\log h)$	$O(h^{3/2})$	$o(h^{3/2-\delta})$

Table: convergence for V_l as observed numerically and proved analytically for both the Euler and Milstein discretisations. δ can be any strictly positive constant.

MLMC Numerical Analysis

Analysis for Euler discretisation for scalar SDE:

- lookback and barrier: Giles, Higham & Mao (*Finance & Stochastics, 13(3), 2009*)
- digital: Avikainen (*Finance & Stochastics, 13(3), 2009*)

Analysis for Milstein discretisation for scalar SDE:

- Giles, Debrabant & Rößler (TU Darmstadt)
- uses boundedness of all moments to bound the contribution to V_l from "extreme" paths
  (e.g. for which $\max_n |\Delta W_n| > h^{1/2-\delta}$ for some $\delta > 0$)
- uses asymptotic analysis to bound the contribution from paths which are not "extreme"

Milstein scheme

Milstein scheme for multi-dimensional SDEs generally requires Lévy areas:

$$A_{jk,n} = \int_{t_n}^{t_{n+1}} (W_j(t) - W_j(t_n))\, \mathrm{d}W_k - (W_k(t) - W_k(t_n))\, \mathrm{d}W_j.$$

- $O(h^{1/2})$ strong convergence in general if omitted
- Can still get good convergence for Lipschitz payoffs by using $W^c(t) = \frac{1}{2}(W^{f1}(t) + W^{f2}(t))$ with two fine paths created by antithetic Brownian Bridge construction
- For barrier and digital options, need to simulate Lévy areas – tradeoff between cost and accuracy, optimum may require $O(h^{3/2})$ sub-sampling of Brownian paths, giving $O(h^{3/4})$ strong convergence

Other/future work

- Quasi-Monte Carlo:
  - uses deterministic sample "points" to achieve an error which is nearly $O(N^{-1})$ in the best cases
- Greeks:
  - the multilevel approach should work well, combining pathwise sensitivities with "vibrato" treatment to cope with lack of smoothness
- variance-gamma, CGMY and other Lévy processes
- American options – the next big challenge:
  - instead of Longstaff-Schwartz approach, view it as an exercise boundary optimisation problem, and use multilevel optimisation?

Conclusions

Multilevel Monte Carlo method has already achieved

- improved order of complexity
- significant benefits for model problems

but there is still a lot more research to be done, both theoretical and applied.

M.B. Giles, “Multilevel Monte Carlo path simulation”, *Operations Research*, 56(3):607-617, 2008.

M.B. Giles. “Improved multilevel Monte Carlo convergence using the Milstein scheme”, pp. 343-358 in *Monte Carlo and Quasi-Monte Carlo Methods 2006*, Springer, 2007.

Papers are available from:
`www.maths.ox.ac.uk/~gilesm/finance.html`

Computational finance on GPUs

Outline:

- current trends with CPUs
- rise of GPUs
- use in HPC
- use in finance
- my experience
- why I think GPUs will stay faster than CPUs

Intel CPUs

- move to faster clock frequencies stopped due to high power consumption – big push now is to multicore chips
- current chips have up to 4 cores, each with a small SSE vector unit (4 `float` or 2 `double`)
- in next 2 years, “Westmere” likely to go up to 10 cores with AVX vectors twice as long
- technologically, many more cores are possible, but will the applications demand it, or is future direction towards low-power low-cost mobile CPUs?
- key point is that cores are general purpose, independent, able to execute different processes simultaneously

GPUs

- many-core chips (up to 240 cores on NVIDIA chips)
- simplified logic (minimal caching, no out-of-order execution, no branch prediction) means much more of the chip is devoted to floating-point computation
- usually arranged as multiple units with each unit being effectively a vector unit, all cores doing the same thing at the same time, and all units executing the same program
- very high bandwidth (up to 140GB/s) to graphics memory (up to 4GB)
- not general purpose – aimed at naturally parallel applications like graphics and Monte Carlo simulations

GPU vendors

- NVIDIA: up to 30×8 cores at present
- AMD (ATI): comparable hardware, but poor software development environment at present
- IBM: Cell processor has 1 PowerPC unit plus 8 SPE vector units – relatively hard to program
- Intel: “Larrabee” GPU due out in Q1 2010, with 16-24 unit each with a vector unit – software support for first-generation product not yet clear

High-end HPC

- RoadRunner system at Los Alamos in US
  - first Petaflop supercomputer
  - IBM system based on Cell processors
- TSUBAME system at Tokyo Institute of Technology
  - 170 NVIDIA Tesla servers, each with 4 GPUs
- GENCI / CEA in France
  - Bull system with 48 NVIDIA Tesla servers
- within UK
  - Cambridge is getting a cluster with 32 Teslas
  - other universities are getting smaller clusters

Use in computational finance

- BNP Paribas has announced production use of a small cluster
  - 2 NVIDIA Tesla units (8 GPUs, each with 240 cores)
  - replacing 250 dual-core CPUs
  - factor 10x savings in power (2kW vs. 25kW)
- lots of other banks doing proof-of-concept studies
  - my impression is that IT groups are very keen; quants are concerned about effort involved
- I'm working with NAG to provide a random number generation library to simplify the task

Finance ISVs

Several ISV's now offer software based on NVIDIA's CUDA development environment:

- SciComp
- Quant Catalyst
- UnRisk
- Hanweck Associates
- Level 3 Finance
- others listed on NVIDIA CUDA website

Many of these are small, but it indicates the rapid take-up of this new technology

Programming

Big breakthrough in GPU computing has been NVIDIA's development of CUDA programming environment

- C plus some extensions and some C++ features
- host code runs on CPU, CUDA code runs on GPU
- explicit movement of data across the PCIe connection
- very straightforward for Monte Carlo applications, once you have a random number generator
- significantly harder for finite difference applications
- see example codes on my website

Programming

Next major step is development of OpenCL standard

- pushed strongly by Apple, which now has NVIDIA GPUs in its entire product range, but doesn't want to be tied to them forever
- drivers are computer games physics, MP3 encoding, HD video decoding and other multimedia applications
- based on CUDA and supported by NVIDIA, AMD, Intel, IBM and others, so developers can write their code once for all platforms
- first OpenCL compilers likely later this year
- will need to re-compile on each new platform, and maybe also re-optimise the code – auto-tuning is one of the big trends in scientific computing

My experience

- Random number generation (mrg32k3a/Normal):
  - 2000M values/sec on GTX 280
  - 70M values/sec on Xeon using Intel's VSL library
- LIBOR Monte Carlo testcase:
  - 180x speedup on GTX 280 compared to single thread on Xeon
- 3D PDE application:
  - factor 50x speedup on GTX 280 compared to single thread on Xeon
  - factor 10x speedup compared to two quad-core Xeons

GPU results are all single precision – double precision is up to 4 times slower, probably factor 2 in future.

Why GPUs will stay ahead

Technical reasons:

- SIMD cores (instead of MIMD cores) means larger proportion of chip devoted to floating point computation
- tightly-coupled fast graphics memory means much higher bandwidth

Commercial reasons:

- CPUs driven by price-sensitive office/home computing; not clear these need vastly more speed
- CPU direction may be towards low cost, low power chips for mobile and embedded applications
- GPUs driven by high-end applications – prepared to pay a premium for high performance

What is needed now?

- more libraries and program development tools to reduce programming effort
- more ISV application codes
- more education / training in parallel computing in universities
- fast development of the OpenCL standard and compilers
- continued 10x superiority in price/performance and energy efficiency relative to CPUs

Further information

LIBOR and finite difference test codes
`www.maths.ox.ac.uk/~gilesm/hpc/`

NAG parallel random number generator
(John Holden, Anthony Ng, Robert Tong)
`info@nag.co.uk`

NVIDIA's CUDA homepage
`www.nvidia.com/object/cuda_home.html`

NVIDIA's computational finance page
`www.nvidia.com/object/computational_finance.html`

Nomura:
Philip Pratt (London), Faisal Sharji (New York)