(Very limited) progress on MATLAB and C/C++ implementations of an MLMC package

Mike Giles

Mathematical Institute, University of Oxford

MCQMC 2022

July 18, 2022

Outline

- motivation
- code structure
- various diagnostics and checks
- MLMC algorithm details
- languages
- parallelisation
- future plans

Motivation

- standard MLMC driving routines and associated plotting for all applications
- plenty of examples coming from various papers (current buzzword is "reproducible research")
- both make it easy for newcomers to start using MLMC, and see some of the practical details involved
- emphasis on simplicity, robustness and diagnostics
- prepared to sacrifice a little on performance, but still want good parallelisation
- also want good coverage of different languages

Code structure

- `mlmc`:
  "driver" routine which performs the MLMC calculation using a user code which generates N_ℓ samples on level ℓ and returns

$$\sum_n (P_\ell^{(n)} - P_{\ell-1}^{(n)})^m, \quad m = 1,2,3,4, \quad \sum_n (P_\ell^{(n)})^m, \quad m = 1,2, \quad \sum_n C_\ell^{(n)}$$

- `mlmc_test`:
  routine which does a lot of tests and then calls `mlmc` to perform MLMC calculations for different accuracies

- `mlmc_test_100`:
  test routine which performs 100 MLMC calculations for each accuracy to check RMS accuracy achieved

- `mlmc_plot, mlmc_plot_100`:
  routines for plotting results

Details of MLMC code

`mlmc_test` first performs a set of calculations using a fixed number of samples on each level of resolution, and produces 4 plots:

- $\log_2(V_\ell)$ versus level ℓ

  If $V_\ell \sim 2^{-\beta\ell}$ then the slope of this line should asymptote towards $-\beta$

- $\log_2(|\mathbb{E}[P_\ell - P_{\ell-1}]|)$ versus level ℓ

  If $|\mathbb{E}[P_\ell - P_{\ell-1}]| \sim 2^{-\alpha\ell}$ then the slope of this line should asymptote towards $-\alpha$

- consistency check versus level

- kurtosis versus level

Consistency check

If a, b, c are estimates for $\mathbb{E}[P_{\ell-1}]$, $\mathbb{E}[P_\ell]$, $\mathbb{E}[P_\ell - P_{\ell-1}]$, then it should be true that $a - b + c \approx 0$.

The consistency check verifies that this is true, to within the accuracy one would expect due to sampling error.

Since

$$\sqrt{\mathbb{V}[a-b+c]} \leq \sqrt{\mathbb{V}[a]} + \sqrt{\mathbb{V}[b]} + \sqrt{\mathbb{V}[c]}$$

it computes the ratio

$$\frac{|a-b+c|}{3\left(\sqrt{\mathbb{V}[a]} + \sqrt{\mathbb{V}[b]} + \sqrt{\mathbb{V}[c]}\right)}$$

The probability of this ratio being greater than 1 based on random sampling errors is extremely small. If it is, it indicates a likely programming error.

Kurtosis check

The MLMC approach needs a good estimate for $V_\ell = \mathbb{V}[P_\ell - P_{\ell-1}]$, but how many samples are need for this?

As few as 10 may be sufficient in many cases for a rough estimate, but many more are needed when there are rare outliers.

When the number of samples N is large, the standard deviation of the sample variance for a random variable X with zero mean is approximately

$$\sqrt{\frac{\kappa - 1}{N}}\ \mathbb{E}[X^2] \quad \text{where kurtosis } \kappa \text{ is defined as} \quad \kappa = \frac{\mathbb{E}[X^4]}{(\mathbb{E}[X^2])^2}$$

(`http://mathworld.wolfram.com/SampleVarianceDistribution.html`)

As well as computing κ_ℓ, `mlmc_test` gives a warning if κ_ℓ is very large.

MLMC algorithm

start with $L = 2$, and initial target of N_0 samples
on levels $\ell = 0, 1, 2$

while extra samples need to be evaluated **do**
  evaluate extra samples on each level
  compute/update estimates for $V_\ell,\ C_\ell,\ \ell = 0, \ldots, L$
  define optimal $N_\ell,\ \ell = 0, \ldots, L$
  if no new samples needed **then**
    test for weak convergence
    if not converged **then**
      if $L = L_{max}$ **then**
        print warning message – failed to converge
      else
        set $L := L+1$, and initialise target N_L
      end if
    end if
  end if
end while

MLMC algorithm

Objective: to achieve

$$\text{MSE} \;=\; \sum_{\ell=0}^{L} V_\ell/N_\ell + (\mathbb{E}[P_L-P])^2 \;\leq\; \varepsilon^2$$

by choosing L such that

$$(\mathbb{E}[P_L-P])^2 \leq \theta\,\varepsilon^2$$

and N_ℓ such that

$$\sum_{\ell=0}^{L} V_\ell/N_\ell \leq (1-\theta)\,\varepsilon^2$$

I used to use $\theta=0.5$, but now tend to use $\theta=0.25$.

MLMC – optimal N_ℓ

Given L, optimal choice for N_ℓ is

$$N_\ell = \frac{1}{1-\theta}\ \varepsilon^{-2} \sqrt{V_\ell / C_\ell}\ \sum_{\ell'=0}^{L} \left(\sqrt{V_{\ell'}\, C_{\ell'}} \right)$$

V_ℓ is estimated from empirical variance.

User code defines C_ℓ, for example by counting how many random numbers are generated, or monitoring the execution time

MLMC – convergence check

If $\mathbb{E}[P_\ell - P_{\ell-1}] \propto 2^{-\alpha\ell}$ then the remaining error is

$$\mathbb{E}[P - P_L] = \sum_{\ell=L+1}^{\infty} \mathbb{E}[P_\ell - P_{\ell-1}] \approx \mathbb{E}[P_L - P_{L-1}] \sum_{k=1}^{\infty} 2^{-\alpha k}$$

$$= \mathbb{E}[P_L - P_{L-1}] \,/\, (2^\alpha - 1)$$

We want $|\mathbb{E}[P - P_L]| < \sqrt{\theta}\, \varepsilon$, so that gives the convergence test

$$|\mathbb{E}[P_L - P_{L-1}]| \,/\, (2^\alpha - 1) < \sqrt{\theta}\, \varepsilon$$

For robustness, we extend this check to extrapolate also from the previous two data points $\mathbb{E}[P_{L-1} - P_{L-2}]$, $\mathbb{E}[P_{L-2} - P_{L-3}]$, and take the maximum over all three as the estimated remaining error.

Coping with poor kurtosis

The total MLMC cost is approximately

$$\frac{1}{1-\theta}\,\varepsilon^{-2}\left(\sum_{\ell=0}^{L}\sqrt{V_\ell\, C_\ell}\right)^2$$

so to guard against the possibility of a low estimate of V_ℓ, I intend to use

$$V'_\ell = \max\left\{\ V_\ell,\ \tfrac{1}{2}\, V'_{\ell-1}\, C_{\ell-1}/C_\ell\ \right\}$$

for $\ell \geq 2$ with $V'_1 = V_1$, to improve the robustness of the algorithm

I'm much less concerned about a high estimate for V_ℓ as the cost coming from the finest levels is usually minimal

Languages

- MATLAB – my past preference for prototyping
- C/C++ – my current preference for performance reasons
- python – keen to support as it takes over from MATLAB for prototyping
- others? – R, Julia

Parallelisation

MATLAB:

- rather heavy-weight task parallelism
- new light-weight thread parallelism coming in next release?

C/C++: 100x performance difference between:

- single-threaded with native RNG
- OpenMP multi-threaded with MKL/VSL RNG

python: would welcome advice on how best to parallelise

Parallelisation

Key to excellent C/C++ parallel performance is:

- separate generator for each thread (own memory space so no "false sharing")
- thread-specific skip-ahead so each thread works on a different segment of RNG stream
- each thread fills a large buffer in L2 cache with random numbers, then refills when they are used up – generating lots of random numbers at the same time makes maximum use of Intel's vector units

Future plans

- MCQMC – trickiest bit is deciding on the interface between the library routine and the user routine
- nested MLMC / MIMC – my motivation is mixed precision computing within standard MLMC for SDE
- new multithreaded MATLAB? parallel python?
- automatic determination of optimal starting level?
- better documentation
- more example applications
- new software for binomial distribution
  (building on previous work for Poisson distribution)

webpage: `https://people.maths.ox.ac.uk/gilesm/mlmc/`