Stochastic Simulation: Lecture 14

Prof. Mike Giles

Oxford University Mathematical Institute

Details of MLMC code

In Practical 3, you will be working with a software which implements the multilevel Monte Carlo method:

- `mlmc.m` / `mlmc.py` / `mlmc.cpp`:
  "driver" code which performs the MLMC calculation using a user routine to estimate $\mathbb{E}[P_\ell - P_{\ell-1}]$ using N_ℓ samples

- `mlmc_test.m` / `mlmc_test.py` / `mlmc_test.cpp`:
  routine which does a lot of tests and then calls `mlmc` to perform a number of MLMC calculations

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

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

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` will give a warning if κ_ℓ is very large.

Kurtosis check

An extreme (but important) example is a digital option in which P always takes the value 0 or 1.

In this case we have

$$X \equiv P_\ell - P_{\ell-1} = \begin{cases} 1, & \text{probability } p \\ -1, & \text{probability } q \\ 0, & \text{probability } 1-p-q \end{cases}$$

If $p, q \ll 1$, then $\mathbb{E}[X] \approx 0$, and

$$\kappa \approx \frac{p+q}{(p+q)^2} = (p+q)^{-1} \gg 1$$

Therefore, many samples are required for a good estimate of V_ℓ, and if we don't have many samples, we may even get all $X^{(n)} = 0$, which will give an estimated variance of zero.

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.

In python code, $C_\ell = 2^{\gamma \ell}$, where γ is user input.

In MATLAB and C++ code user defines C_ℓ, for example by counting how many random numbers are generated.

MLMC – convergence check

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

$$\begin{aligned}\mathbb{E}[P - P_L] = \sum_{\ell=L+1}^{\infty} \mathbb{E}[P_\ell - P_{\ell-1}] &\approx \mathbb{E}[P_L - P_{L-1}] \sum_{k=1}^{\infty} 2^{-\alpha k} \\ &= \mathbb{E}[P_L - P_{L-1}] \,/\, (2^\alpha - 1)\end{aligned}$$

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.

Details of MATLAB MLMC code

```
% function [P, Nl, cost] = mlmc(mlmc_l,N0,eps,Lmin,Lmax,
%                                alpha,beta,gamma, varargin)
%
% multi-level Monte Carlo estimation
%
% P      = value
% Nl     = number of samples at each level
% cost   = total cost
%
% N0     = initial number of samples          > 0
% eps    = desired accuracy (rms error)       > 0
% Lmin   = minimum level of refinement        >= 2
% Lmax   = maximum level of refinement        >= Lmin
%
% alpha -> weak error is  O(2^{-alpha*l})
% beta  -> variance is    O(2^{-beta*l})
% gamma -> sample cost is O(2^{gamma*l})
%
% varargin = optional additional user variables to be passed to mlmc_l
%
% if alpha, beta, gamma are not positive, then they will be estimated
%
```

Details of MATLAB MLMC code

```
%
% mlmc_l = function for level l estimator
%
% [sums, cost] = mlmc_fn(l,N, varargin)     low-level routine
%
% inputs:  l = level
%          N = number of samples
%          varargin = optional additional user variables
%
% output: sums(1) = sum(Y)
%         sums(2) = sum(Y.^2)
%         where Y are iid samples with expected value:
%         E[P_0]           on level 0
%         E[P_l - P_{l-1}] on level l>0
%         cost = cost of N samples
```

Details of MATLAB MLMC code

```
function [P, Nl, Cl] = mlmc(mlmc_l,N0,eps,Lmin,Lmax, ...
                            alpha0,beta0,gamma0, varargin)

%
% check input parameters
%
  if (Lmin<2)
    error('error: needs Lmin >= 2');
  end

  if (Lmax<Lmin)
    error('error: needs Lmax >= Lmin');
  end

  if (N0<=0 || eps<=0)
    error('error: needs N0>0, eps>0 \n');
  end

%
% initialisation
%
  alpha = max(0, alpha0);
  beta  = max(0, beta0);
  gamma = max(0, gamma0);
```

Details of MATLAB MLMC code

```
  theta = 0.25;

  L = Lmin;

  Nl(1:L+1)       = 0;
  suml(1:2,1:L+1) = 0;
  costl(1:L+1)    = 0;
  dNl(1:L+1)      = N0;

  while sum(dNl) > 0

%
% update sample sums
%
    for l=0:L
      if dNl(l+1) > 0
        [sums cost] = mlmc_l(l,dNl(l+1), varargin{:});
        Nl(l+1)     = Nl(l+1)     + dNl(l+1);
        suml(1,l+1) = suml(1,l+1) + sums(1);
        suml(2,l+1) = suml(2,l+1) + sums(2);
        costl(l+1)  = costl(l+1)  + cost;
      end
    end
```

Details of MATLAB MLMC code

```
%
% compute absolute average, variance and cost
%
    ml = abs(   suml(1,:)./Nl);
    Vl = max(0, suml(2,:)./Nl - ml.^2);
    Cl = costl./Nl;

%
% fix to cope with possible zero values for ml and Vl
% (can happen in some applications when there are few samples)
%
    for l = 3:L+1
      ml(l) = max(ml(l), 0.5*ml(l-1)/2^alpha);
      Vl(l) = max(Vl(l), 0.5*Vl(l-1)/2^beta);
    end
```

Details of MATLAB MLMC code

```
%
% use linear regression to estimate alpha, beta, gamma if not given
%
    A = repmat((1:L)',1,2).^repmat(1:-1:0,L,1);

    if alpha0 <= 0
      x     = A \ log2(ml(2:end))';
      alpha = max(0.5,-x(1));
    end

    if beta0 <= 0
      x     = A \ log2(Vl(2:end))';
      beta  = max(0.5,-x(1));
    end

    if gamma0 <= 0
      x     = A \ log2(Cl(2:end))';
      gamma = max(0.5,x(1));
    end

%
% set optimal number of additional samples
%
    Ns  = ceil( sqrt(Vl./Cl)*sum(sqrt(Vl.*Cl)) / ((1-theta)*eps^2) );
    dNl = max(0, Ns-Nl);
```

Details of MATLAB MLMC code

```
% if (almost) converged, estimate remaining error and decide
% whether a new level is required
%
    if sum( dNl > 0.01*Nl ) == 0
      range = 0:min(2,L-1);
      rem = max(ml(L+1-range) ./ 2.^(range*alpha)) / (2^alpha - 1);

      if rem > sqrt(theta)*eps
        if (L==Lmax)
          fprintf(1,'*** failed to achieve weak convergence *** \n');
        else
          L        = L+1;
          Vl(L+1) = Vl(L) / 2^beta;
          Cl(L+1) = Cl(L) * 2^gamma;
          Nl(L+1) = 0;
          suml(1:2,L+1) = 0;
          costl(L+1) = 0;

          Ns  = ceil( sqrt(Vl./Cl) * sum(sqrt(Vl.*Cl)) ...
                                   / ((1-theta)*eps^2) );
          dNl = max(0, Ns-Nl);
        end
      end
    end
  end
```

Details of MATLAB MLMC code

```
%
% finally, evaluate multilevel estimator
%
  P = sum(suml(1,:)./Nl);
```