Week 14: Monday, May 2

Logistics

- Welcome to the last week of classes!
- Project 3 is due Friday, May 6.
- Final is Friday, May 13. Lecture Wednesday will be a review, and section this week will be devoted to practice exam problems. Come with questions.

Problem du jour

What is the expected output of the following code?

```
X  = 20*(rand(1,N)-0.5);
fX = exp(-X.^2/2);
gX = 1/20;
lX = fX./gX;
result  = mean(lX);
errbar  = std(lX)/sqrt(N);
```

Answer: The result should converge to $E[l(X)]$ where X is uniform on $[-10, 10]$ and $l(x) = 20\exp(-x^2/2)$. The probability density function for X is

$$g(x) = \begin{cases} 1/20, & -10 \le x \le 10 \\ 0, & \text{otherwise.} \end{cases}$$

and the expected value is

$$E[l(X)] = \int_{\mathbb{R}} l(x)g(x)\,dx = \int_{-20}^{20} \exp(-x^2/2)\,dx,$$

which should be quite close to $\sqrt{2\pi}$. We've created some bias by truncating at ± 10, but this error is nowhere near as large as the statistical error due to Monte Carlo.

Variance reduction

The problem above is a simple example of Monte Carlo integration. We now want to see how to make this simple example more efficient by reducing the variance of the estimator. We will approach this in a few different ways.

Importance sampling

Let us consider the computation

$$\sqrt{2\pi} = 2 \int_0^\infty \exp(-x^2/2)\, dx.$$

Using the idea of the problem du jour, we could estimate $\sqrt{2\pi}$ by drawing uniform samples on $[0, L]$ for L large enough. But this estimator has rather high variance, and the variance gets larger the larger L is. This is intuitive in that most of the sample points don't really matter to the computation, since $\exp(-x^2/2)$ decays very quickly away from zero.

The integrand $\exp(-x^2/2)$ is largest near the origin, so we get the most contribution to our integral when we have samples near zero. Therefore, it makes sense to use a method that samples more frequently near the origin, rather than sampling uniformly over some large range of x values

$$\sqrt{2\pi} = 2 \int_0^\infty \frac{\exp(-x^2/2)}{\exp(-x)} \exp(-x)\, dx = 2E[\exp(-X^2/2)/\exp(-X)]$$

where X is an exponential random variable.

```
% lec27zmean2(N)
%
% Compute sqrt(2*pi) by Monte Carlo. Use importance sampling.

function [result, err] = lec27zmean2(N)

  if nargin < 1, N = 1000; end

  Y  = -log(rand(1,N));
  fY = exp(-Y.^2/2);
  gY = exp(-Y)/2;
  lY = fY./gY;
```

result = **mean**(lY);
err = **std**(lY)/**sqrt**(N);

Control variates

I want to compute the expectation of $l(x) = \exp(x - x^2/2)$, but perhaps I've decided its too hard. But I know that most of the interesting behavior is near the origin, so perhaps I can approximate $l(x)$ by a polynomial over some interval close to zero. Let's try just interpolating by a quadratic at $x = 0$, $x = 1$, and $x = 2$, and discarding everything past $x = 2$:

$$h(x) = \begin{cases} \sqrt{e} - (\sqrt{e} - 1)(x-1)^2, & x \in [0,2] \\ 0, & \text{otherwise} \end{cases}.$$

While $h(X)$ is not identical to $l(X)$, the two random variables surely are correlated. Furthermore, we can compute $E[h(X)]$ analytically; a somewhat tedious calculus exercise yields

$$E[h(X)] = \sqrt{e}(1 - e^{-2}) - (\sqrt{e} - 1)(1 - 5e^{-2}).$$

The fact that $h(X)$ and $l(X)$ should be correlated, together with the fact that we can compute $E[h(X)]$ in closed form, makes $h(X)$ an ideal candidate to serve as a *control variate* with which we can construct a better estimator, as we shall now see.

First, note that

$$E[l(X)] = E[l(X) - ch(X)] + cE[h(X)].$$

So $\hat{l}_c(X) = l(X) - ch(X) + cE[h(X)]$ has the same expected value that $l(X)$ does; but

$$\operatorname{var}[\hat{l}_c(X)] = \operatorname{var}[l(X)] - 2c \operatorname{cov}[l(X), h(X)] + c^2 \operatorname{var}[h(X)].$$

If we choose $c_* = \operatorname{cov}[l(X), h(X)] / \operatorname{var}[h(X)]$, we have

$$\operatorname{var}[\hat{l}_{c_*}(X)] = \operatorname{var}[l(X)] \left(1 - \operatorname{corr}[l(X), h(X)]^2\right).$$

If $l(X)$ and $h(X)$ are highly correlated, then $\hat{l}_{c_*}(X)$ may have a much lower variance than $l(X)$. Of course, computing the covariance analytically is hard, but we can always do it numerically.

```
% lec27zmean4(N)
%
% Compute sqrt(2*pi) by Monte Carlo. Use importance sampling +
% a control variate.

function [result, err] = lec27zmean4(N)

  if nargin < 1, N = 1000; end

  Y  = -log(rand(1,N));
  fY = exp(-Y.^2/2);
  gY = exp(-Y)/2;
  lY = fY./gY;

  hY  = ( sqrt(e)-(sqrt(e)-1)*(Y-1).^2 ).*double( Y<2 );
  EhY = ( sqrt(e)*(1-e^-2) - (sqrt(e)-1)*(1-5*e^-2) );

  cs     = -sum( (lY-mean(lY)) .* (hY-EhY) )/sum( (hY-EhY).^2 );
  W      = lY + cs*(hY-EhY);
  result = mean(W)
  err    = std(W)/sqrt(N);
```

Antithetic variables

Now let's turn to the problem of computing $\pi/4$ by throwing darts at $[0,1]^2$ and seeing what fraction lie inside the unit circle. Note that if (X_i, Y_i) is a uniform random sample from the square, then $(1 - X_i, 1 - Y_i)$ is a correlated sample. It turns out that if ϕ is the indicator for the unit circle, then $\phi(X_i, Y_i)$ and $\phi(1 - X_i, 1 - Y_i)$ have negative covariance; this makes sense, since only one of them can be outside the unit circle (though both could be the same). Therefore, the estimator $\phi(X, Y)/2 + \phi(1 - X, 1 - Y)/2$ actually has lower variance than $\phi(X, Y)$. This is the method of *antithetic variables*.

```
% lec27pi_mc(N)
%
% Compute pi by Monte Carlo. Use antithetic variables.

function [result, err] = lec27pi_mc2(N,d)
```

```
if nargin < 1, N = 1000; end
if nargin < 2, d = 2;    end

% Version 2: Antithetic variates
XY      = rand(d,N);
XY2     = 1-XY;
trials1 = double(sum(XY.^2,1)<1);
trials2 = double(sum(XY2.^2,1)<1);
trials  = (trials1+trials2)/2;
result  = mean(trials);
shat    = std(trials);
err     = shat/sqrt(N);
```

A concluding note

I will not ask you about variance reduction on the final exam, but if you ever find yourself using Monte Carlo methods, it is worth knowing the basic ideas of these methods. Much of the sophistication in getting Monte Carlo methods to run fast is in finding good variance reduction techniques.

Accelerating another method to compute π

Recall way back at the beginning of the semester, we discussed a way of computing the semiperimeters s_k of 2^k-gons as a way of computing π. This turned out to be equivalent to finding a recurrence to compute

$$s_k = 2^k \sin(2^{-k}\pi),$$

and when we Taylor expand the sine function, we have

$$s_k = \pi + C_1 4^{-k} + C_2 16^{-k} + C_3 64^{-k} + \ldots$$

where C_1, C_2, C_3, etc. are constants that do not depend on k. The most straightforward estimate, $\pi \approx s_k$, therefore has a predictable error that is roughly proportional to 4^{-k}. But what if we tried to combine several semiperimeters to get a better estimate?

Let's start simple, and try taking a linear combination of s_k and s_{k+1}:

$$\alpha s_k + \beta s_{k+1} = (\alpha + \beta)\pi + (4\alpha + \beta)C_1 4^{-k-1} + O(16^{-k}).$$

If we choose $\alpha = -1/3$ and $\beta = 4/3$, we solve the linear system $\alpha + \beta = 1$ and $4\alpha + \beta = 0$. Therefore,

$$\frac{4s_{k+1} - s_k}{3} = \pi + O(16^{-k}).$$

If we take an appropriate combination of s_{k+2}, s_{k+1}, and s_k, we can do even better, getting an approximation with error $O(64^{-k})$. This is linear convergence, but it is ferociously fast nonetheless.