STAT 513/413: Lecture 14
Variance reduction

(the staple of Monte Carlo courses)

Rizzo 6.3, 6.4, 6.5, 6.6, 6.7, 6.8

The beginning of our soap opera

In the forthcoming series of trasparencies ("soap opera"), we will entertain two test, "Guinea pig" functions: g_1 and g_2.

They are to be introduced shortly, now only the R code for those.

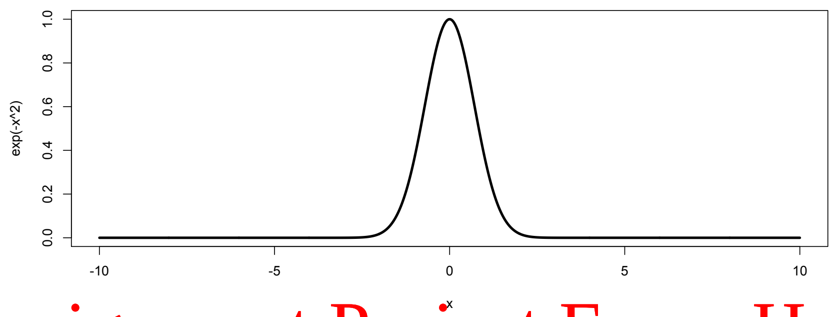
```
> g1 <- function(x) exp(-x^2)          # once again
> g2 <- function(x) exp(-x)/(1+x^2)
```

Later, we need also to know what is $\sqrt{\pi}$.

```
> sqrt(pi)
[1] 1.772454
```

Guinea pig I

The first function is $g_1(x) = e^{-x^2}$



We want to compute the integral $\int_{-\infty}^{\infty} e^{-x^2}\,dx$

Well, we actually know it is equal to $\sqrt{\pi}$, but this knowledge enables us to assess the precision of the method(s)

In the algorithms, we used to approximate the integral above by

$\int_{-10}^{10} e^{-x^2}\,dx$ or also by $2\int_{0}^{10} e^{-x^2}\,dx$ but we know now that

$\int_{-4}^{4} e^{-x^2}\,dx$ or also by $2\int_{0}^{4} e^{-x^2}\,dx$ is just enough

Once again: rectangular better than Monte Carlo

...once again - check it out:

```
> sqrt(pi)
[1] 1.772454
> x=seq(-4,4,length=11)[1:10]
> mean(g1(x))*8
[1] 1.772454
> x=seq(-10,10,length=11)[1:10]
> mean(g1(x))*20
[1] 2.073263
> x=seq(-10,10,length=21)[1:20]
> mean(g1(x))*20
[1] 1.772637
> x=seq(-10,10,length=31)[1:30]
> mean(g1(x))*20
[1] 1.772454

> integrate(g1,-Inf,Inf)
1.772454 with absolute error < 4.3e-06
```

Guinea pig II

The other function is $g_2(x) = \dfrac{e^{-x}}{1+x^2}$

and we want to compute the integral $\displaystyle\int_0^1 \frac{e^{-x}}{1+x^2}\,dx$

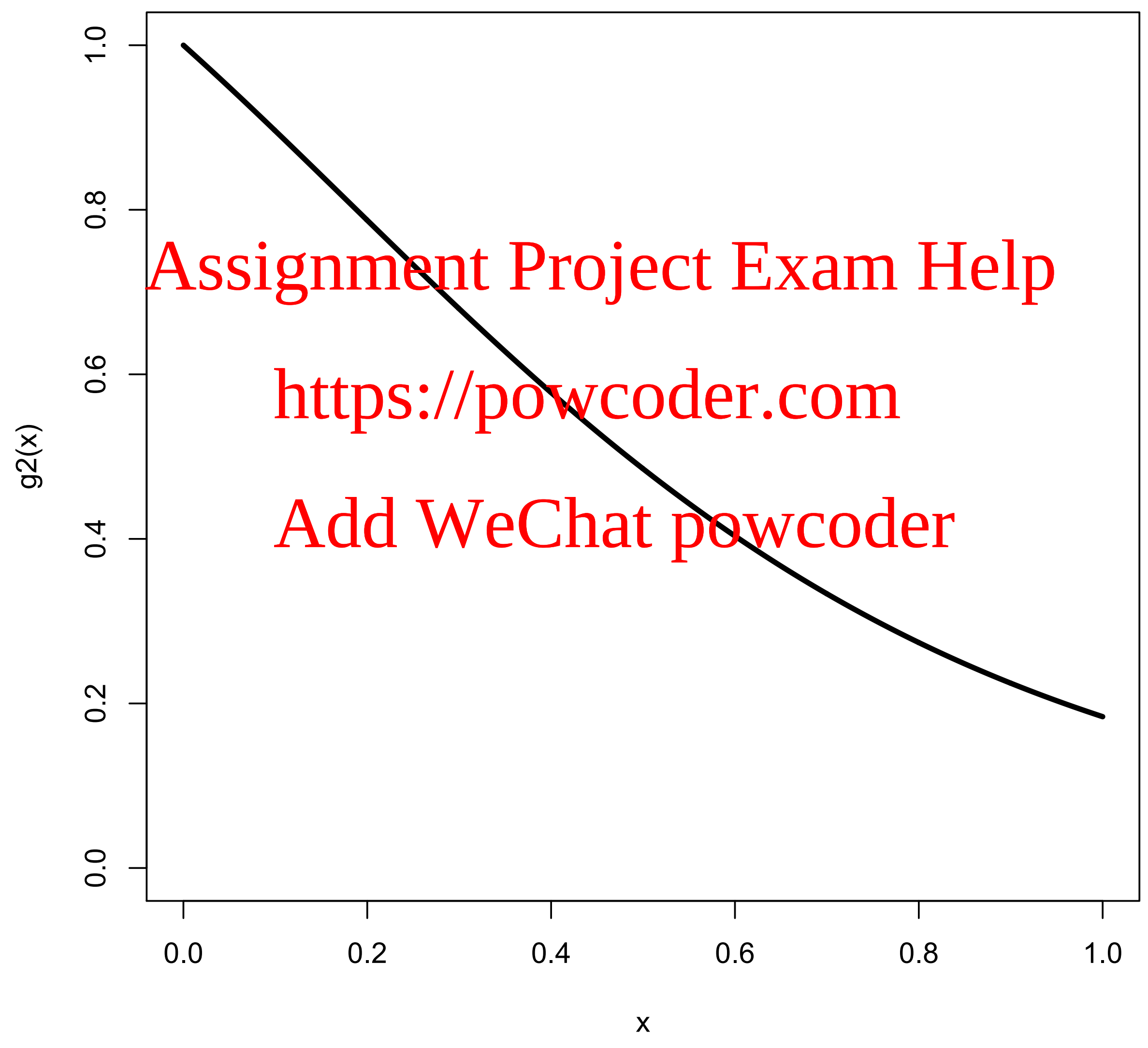
For this one, we compute the “truth” numerically

```
> x=seq(0,1,length=10001); sum(g2(x))*diff(x)[1]
[1] 0.5248563
> x=seq(0,1,length=100001); sum(g2(x))*diff(x)[1]
[1] 0.5248031
> x=seq(0,1,length=1000001); sum(g2(x))*diff(x)[1]
[1] 0.5247977
> x=seq(0,1,length=10000001); sum(g2(x))*diff(x)[1]
[1] 0.5247972
> x=seq(0,1,length=100000001); sum(g2(x))*diff(x)[1]
[1] 0.5247971
> x=seq(0,1,length=200000001); sum(g2(x))*diff(x)[1]
[1] 0.5247971
> integrate(g2,0,1)
0.5247971 with absolute error < 5.8e-15
```

Also here is rectangular better than Monte Carlo

```
> x=seq(0,1,length=100000001); sum(g2(x))*diff(x)[1]
[1] 0.5247971
> mean(g2(runif(100000001)))
[1] 0.5248177
> mean(g2(runif(100000001)))
[1] 0.5248393
> mean(g2(runif(100000001)))
[1] 0.5247994
```

The graph of g_2



Variance reduction

In the classical Monte Carlo, using the average to estimate an expected value, are looking at some $\frac{\sigma^2}{N}$ as a measure of precision (to use it in a Chebyshev inequality, for instance)

Now:

we are not able to do anything with N

(that is, how many repetitions we need)

but we perhaps can do something with the variance σ^2

$\rightarrow$ Variance Reduction Techniques

1. Antithetic variables (variates)

This strategy requires that the integrand, g, is *monotone*

We explain the idea of it on pairs: suppose that we have U and V independent, with same distribution, and these two are used for computing (estimating) the integral of g as folows:

$$\frac{1}{2}(g(U) + g(V))$$

The corresponding variance is then

$$\frac{1}{4}\mathrm{Var}(g(U) + g(V)) = \frac{1}{4}(\mathrm{Var}(g(U)) + \mathrm{Var}(g(V))) = \frac{1}{2}\mathrm{Var}(g(U))$$

That is less than $\mathrm{Var}(g(U))$, right?

(In practice, of course, we will use not one pair, but many pairs of random numbers)

Antithetic variables: idea

What if U and V were not independent (still with the same distribution)? Then the same variance would be

$$\frac{1}{4}(\mathrm{Var}(g(U)) + 2\,\mathrm{Cov}(g(U), g(V)) + \mathrm{Var}(g(V)))$$

which could be less than the original

$$\tfrac{1}{2}(g(U) + g(V)),$$

as long as $\mathrm{Cov}(g(U), g(V)) < 0$. How to achieve that?

If we take $V = 1 - U$, and U is uniform on $[0, 1]$, then V has the same distribution as U (right?) and $\mathrm{Cov}(U, V)$ is

$$\mathrm{Cov}(U, 1 - U) = \mathrm{Cov}(U, -U) = -\,\mathrm{Cov}(U, U) = -\,\mathrm{Var}(U) \leqslant 0$$

Typically, it is not only $-\,\mathrm{Var}(U) \leqslant 0$, but $-\,\mathrm{Var}(U) < 0$

However, we need it not for U and V, but for $g(U)$ and $g(V)$

(And then we will use it not on one pair, but on many pairs of random numbers)

Lemma (Hoeffding, 1940)

Suppose that g and h are monotone, same sense (that is: either both noincreasing, either both nondecreasing), and X is a random variable. Then

$$\mathrm{Cov}(g(X), h(X)) \geqslant 0$$

Usage: take U uniform on $[0,1]$ and, say, $g(u)$ noincreasing; define $V = 1 - U$, $h(U) = -g(1-U) = -g(V)$. Clearly, V has the same distribution as U and as $h(u) = -g(1-u)$ is nonincreasing too, the Hoeffding lemma gives

$$0 \leqslant \mathrm{Cov}(g(U), h(U)) = \mathrm{Cov}(g(U), -g(V)) = -\,\mathrm{Cov}(g(U), g(V))$$

and therefore $\qquad \mathrm{Cov}(g(U), g(V)) \leqslant 0$

The case $g(u)$ nondecreasing works the same way (check it out!)

We can take it further: suppose that we also generate some distribution there, with the inversion method using the quantile (inverse cdf) function $Q(u) = F^{-1}(u)$. Then the pair

$$g(Q(u)) \qquad -g(Q(1-u))$$

has the same monotonicity properties as $g(u)$ and $-g(1-u)$

Let us try it on Guinea pig I

So now not one, but many pairs of random numbers:

```
> u=runif(10000,0,4); v=4-u; mean(g1(c(u,v)))*8
[1] 1.77311
> u=runif(10000,0,4); v=4-u; mean(g1(c(u,v)))*8
[1] 1.768021
> u=runif(10000,0,4); v=4-u; mean(g1(c(u,v)))*8
[1] 1.783613
```

Nothing extra... Well, yeah:

```
> 2*cov(g1(u),g1(v))
[1] -0.09921393
> v2=var(g1(u)) + 2*cov(g1(u),g1(v)) + var(g1(v)); v2
[1] 0.09401626
> v1=var(g1(u)) + var(g1(v)); v1
[1] 0.1097085
> v2/v1
[1] 0.8569642
```

But note: essentially, we had to generate only half of the random numbers, `u`, the others, `v`, are computed in a straightforward way

Now the second one

```
 u=runif(10000); v=1-u; mean(g2(c(u,v)))
[1] 0.5250257
> u=runif(10000); v=1-u; mean(g2(c(u,v)))
[1] 0.5250572
> u=runif(10000); v=1-u; mean(g2(c(u,v)))
[1] 0.5247662
```

```
> 2*cov(g2(u),g2(v))
[1] -0.1155884
```

```
> v2=var(g2(u)) + 2*cov(g2(u),g2(v)) + var(g2(v)); v2
[1] 0.004348575
```

```
> v1=var(g2(u)) + var(g2(v)); v1
[1] 0.1199369
> v2/v1
[1] 0.03625719
> (v1-v2)/v1
[1] 0.9637428
```

And again, only with the half of random numbers to compute

Another example: recognize?

```
> U=runif(100000000)
> V=1-U
> var(exp(U))
[1] 0.2420872
> -exp(2)/2+2*exp(1)-3/2
[1] 0.2420356
> cov(exp(U),exp(V))
[1] -0.2342585
> -exp(2)+3*exp(1)-1
[1] -0.2342106
>
> var(exp(U)+exp(V))
[1] 0.01565305
> -3*exp(2)+10*exp(1)-5
[1] 0.01564999
>
> (var(exp(U)+exp(V)))/(var(exp(U))+var(exp(V)))
[1] 0.03232965
> (-3*exp(2)+10*exp(1)-5)/(-exp(2)+4*exp(1)-3)
[1] 0.03232993
```

Proof of the Hoeffding lemma

Let g and h be same sense monotone; then for any x, y

$$(g(x) - g(y))(h(x) - h(y)) \geqslant 0$$

(nice, eh? it works also when they are both noincreasing!)

Now, take Y with the same distribution as X, and independent. Then we have (recall: if $Y \geqslant 0$, then $E(Y) \geqslant 0$)

$$0 \leqslant E[(g(X) - g(Y))(h(X) - h(Y))]$$

$$= E[g(X)h(X)] + E[g(Y)h(Y)] - E(g(X)h(Y)) - E(g(Y)h(X))$$

and as X and Y have the same distribution (recall: expected value depends *only* on the distribution), we can continue

$$= 2\,E(g(X)h(X)) - E(g(X)h(Y)) - E(g(Y)h(X)$$

and then use the independence of X and Y (recall...)

$$= 2\,E(g(X)h(X)) - E(g(X))\,E(h(Y)) - E(g(Y))\,E(h(X)$$

and again the same distribution of X and Y

$$= 2\,E(g(X)h(X)) - 2\,E(g(X))\,E(h(X)) = \mathrm{Cov}(g(X), h(X))$$

Hence $\mathrm{Cov}(g(X), h(X)) \geqslant 0$

Further extension

Works also in multivariate situation, when seeking an integral of

$$g(x_1, x_2, \ldots, x_p)$$

and g is monotone in every variable. The pair

$$g(Q(U_1), Q(U_2), \ldots, Q(U_p)$$

and

$$g(Q(1-U_1), Q(1-U_2), \ldots, Q(1-U_p)$$

works in the same way as in the univariate case

By the way: should g be *same sense* monotone in every variable?

(Beware the textbook notation: my N is their m, my p is their n)

Unbiased estimator?

Well, everything is fine, the total variance is reduced...

... but are we still computing the right thing?

In other words, is it still the unbiased estimator?

Again, we demonstrate it on pairs:

$$E\left(\frac{1}{2}(g(U)+g(V))\right) = \frac{1}{2}E(g(U)+g(V))$$

$$= \frac{1}{2}(E(g(U))+E(g(V))) = \frac{1}{2}2\,E(g(U)) = E(g(U))$$

It works regardless whether U and V are independent or not: the expected value of sum is *always* sum of expected values

We only need that U and V, and then $g(U)$ and $g(V)$ have the same distribution, as then $E(g(U)) = E(g(V))$

2. Control variates (variables)

Now we need a function $f(x)$ we *know* the integral of - let us say it is ν. It will be good also when $f(x)$ is close to the integrand $g(x)$ - we can measure this this closeness by correlation (recall the definition)

$$\mathrm{Cor}(f(X), g(X)) = \frac{\mathrm{Cov}(f(X), g(X))}{\sqrt{\mathrm{Var}(f(X))\,\mathrm{Var}(g(X))}}$$

(Which means: $f(x)$ and $g(x)$ are close if this correlation is positive and high - that is, the closer to 1 the correlation is, the close $f(x)$ and $g(x)$ are considered to be)

We then estimate the integral via a regression

$$E(g(X)) - \frac{\mathrm{Cov}(g(X), f(X))}{\mathrm{Var}(f(X))}\big(E(h(X)) - \nu\big)$$

with a possibility to estimate the coefficient $A = -\frac{\mathrm{Cov}(g(X), f(X))}{\mathrm{Var}(f(X))}$

in a preliminary (smaller) Monte Carlo experiment

That is

We compute

$$\hat{A} = -\frac{\sum_{i=1}^{m}\left(g(Y_i) - \frac{1}{m}\sum_{i=1}^{m} g(Y_i)\right)\left(f(Y_i) - \frac{1}{m}\sum_{i=1}^{m} f(Y_i)\right)}{\sum_{i=1}^{m}\left(f(Y_i) - \frac{1}{m}\sum_{i=1}^{m} f(Y_i)\right)^2}$$

and then estimate the integral by

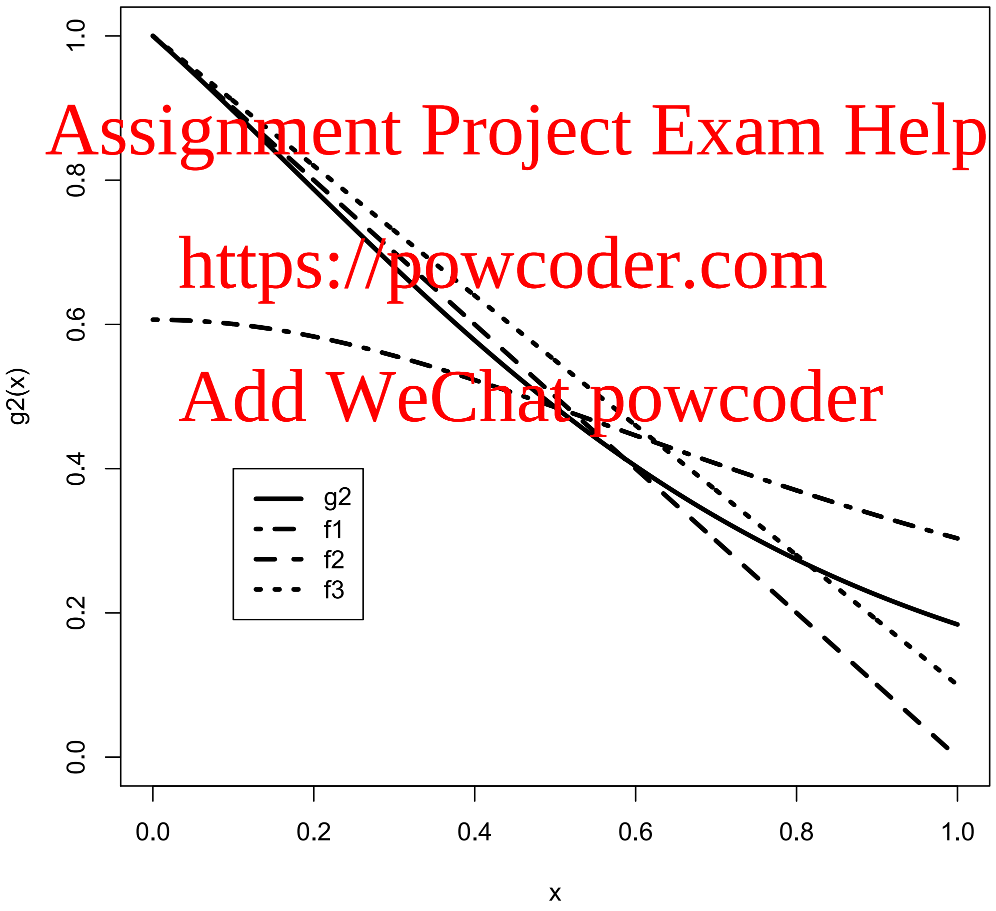
$$\frac{1}{n}\sum_{i=1}^{n}(g(X_i)+\hat{A}(f(X_i))-\nu)) = \frac{1}{n}\sum_{i=1}^{n} g(X_i) + \hat{A}\left(\frac{1}{n}\sum_{i=1}^{n}(f(X_i) - \nu)\right)$$

$$= \frac{1}{n}\sum_{i=1}^{n} g(X_i) + \frac{\hat{A}}{n}\sum_{i=1}^{n} f(X_i) - \hat{A}\nu$$

Now, for our Guinea pig II, g_2

For the control variate, the textbook suggests $f_1(x) = \dfrac{e^{-0.5}}{1+x^2}$,

but my choice would be simply $f_2(x) = 1 - x$ (or $f_3(x) = 1 - 0.9x$?)



So, let us roll like in the textbook

```
> f1 = function (x) exp(-0.5)/(1+x^2)
> set.seed(007)
> u = runif(10000)
> cor(g2(u),f1(u))
[1] 0.9737252
> A = -cov(g2(u),f1(u))/var(f1(u)) ; A
[1] -2.44699
> u = runif(100000)
> mean(g2(u))
[1] 0.5253908
> mean(g2(u) + A*(f1(u)- exp(-0.5)*(pi/4)))
[1] 0.5250312
> v1=var(g2(u)) ; v1
[1] 0.06027686
> v2=var(g2(u) + A*(f1(u)- exp(-0.5)*(pi/4)))
> v2
[1] 0.003117086
> (v1-v2)/v1
[1] 0.9482872
```

And now like in a bigger book

```
> f2 = function(x) 1-x
> set.seed(007)
> u = runif(10000)
> cor(g2(u),f2(u))
[1] 0.9905712
> A = -cov(g2(u),f2(u))/var(f2(u)) ; A
[1] -0.8407234
> u = runif(100000)
> mean(g2(u))
[1] 0.5253908
> mean(g2(u) + A*(f2(u) - 0.5))
[1] 0.5249112
> v1=var(g2(u)) ; v1
[1] 0.06027686
> v2=var(g2(u) + A*(f2(u) - 0.5))
> v2
[1] 0.001115375
> (v1-v2)/v1
[1] 0.9814958
```

And now: you roll! (Exactly as I did before)

That is, with f_3, and in the same way as for f_1 and f_2 before

Can you do even better?

Preliminary Monte Carlo to estimate $\mathtt{A}$?

This suggestion has more to do with the certain theoretical vindication of the technique than with the practice (in the practice we can do it one way or another, with little impact on the result)

If the estimate $\hat{A}$ comes from a preliminary Monte Carlo run

it means that the random numbers it depends on are different than the X_i's that enter

$$\frac{1}{n}\sum_{i=1}^{n}(g(X_i)) + \hat{A}\left(\frac{1}{n}\sum_{i=1}^{n}(f(X_i) - \nu)\right)$$

If the random numbers Y_i are different, then they are independent from X_i

and consequently $\hat{A}$, that depends on Y_i, can be considered independent of the random quantities dependent on X_i - so that

(Recall: $E(XY) = E(X)\,E(Y)$, if X and Y are independent)

The estimator is unbiased

If X_i all have the same distribution, so that

$E(g(X_1)) = E(g(X_2)) = \cdots = \mu$

and also $E(f(X_1)) = E(f(X_2)) = \cdots = \nu$

and $\hat{A}$ is independent on the X_i's, then

$$
\begin{aligned}
E\left(\frac{1}{N}\sum_{i=1}^{N} g(X_i) + \frac{\hat{A}}{N}\sum_{i=1}^{N}(f(X_i) - \nu)\right) \\
&= \frac{1}{N}\sum_{i=1}^{N} E(g(X_i)) + E\left(\frac{\hat{A}}{N}\sum_{i=1}^{N}(f(X_i) - \nu)\right) \\
&= \frac{1}{N}\sum_{i=1}^{N} E(g(X_i)) + E(\hat{A})\, E\left(\frac{1}{N}\sum_{i=1}^{N}(f(X_i) - \nu)\right) \\
&= \frac{1}{N}\sum_{i=1}^{N} E(g(X_i)) + E(\hat{A})\left(\frac{1}{N}\sum_{i=1}^{N}(E(f(X_i)) - \nu)\right) \\
&= \frac{1}{N}\sum_{i=1}^{N} E(g(X_i)) + E(\hat{A})0 = \frac{1}{N}\sum_{i=1}^{N} E(g(X_i)) = \mu
\end{aligned}
$$

Other than that?

Other than that, the Y_i's and X_i's could be the same random numbers - but then we

could not vindicate the estimate by proving that it is unbiased

Nevertheless, the way of estimation can be still vindicated by the fact that for $n \to \infty$ the computed expression still approaches the correct quantity μ (in the sense of convergence in probability)

but yeah, if it is not unbiased, it is difficult to get the exact performance guarantees

But, as I said already, numerically the difference is minimal

and the regression interpretation nicely stands out

Numerically, the difference is minimal

```
> set.seed(007)
> u=runif(10000)
> A = -cov(g2(u),f2(u))/var(f2(u))
> A
[1] -0.8407234
> mean(g2(u))
[1] 0.524227
> mean(g2(u) + A*(f2(u)-0.5))
[1] 0.5247263
> v1=var(g2(u))
> v1
[1] 0.0599499
> v2=var(g2(u) + A*(f2(u)-0.5))
> v2
[1] 0.001125176
> (v1-v2)/v1
[1] 0.9812314
```

And it is indeed regression

```
> set.seed(007)
> u=runif(10000)
> z=f2(u)
> reg=lm(g2(u)~z)
> summary(reg)
...
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 0.1043645  0.0006707   155.6   <2e-16 ***
z           0.8407234  0.0011629   723.0   <2e-16 ***
...
> predict(reg,list(z=0.5))
        1
0.5247263
```

This shows connection to the regression estimator from sample survey methodology

And also shows a path to generalization: if we have several control variates, we can do “multiple regression” then!

Beware, however

```
>  predict(reg,list(z=0.5),se=TRUE)$se.fit^2
[1] 1.125293e-07
```

Yeah, divided by N, or $N-1$, or $N-2$... Not really:

```
> var(g2(u) - coef(reg)[2]*(f2(u)-0.5))
[1] 0.001125176
> predict(reg,list(z=0.5),se=TRUE)$se.fit^2*(length(u))
[1] 0.001125293
> predict(reg,list(z=0.5),se=TRUE)$se.fit^2*(length(u)-1)
[1] 0.001125181
> predict(reg,list(z=0.5),se=TRUE)$se.fit^2*(length(u)-2)
[1] 0.001125068
```

Hmm, which one...?

```
> summary(reg)$sigma^2*(length(u)-2)/(length(u)-1)
[1] 0.001125176
```

This one!

Just for fun

```
> set.seed(007)
> u = runif(10000)
> z=f2(u); y=f1(u)
> reg=lm(g2(u)~z+y); cf=coef(reg)
> predict(reg,list(z=0.5,y=exp(-.5)*(pi/4)))
        1
0.5247242
> mean(g2(u)-cf[2]*(f2(u)-0.5)-cf[3]*(f1(u)-exp(-.5)*(pi/4)))
[1] 0.5247242
> v3=var(g2(u)-cf[2]*(f2(u)-0.5)-cf[3]*(f1(u)-exp(-.5)*(pi/4)))
> v3
[1] 0.0005562679
> (v1-v3)/v1
[1] 0.9907211
```

Yeah, and also: why `length(u)-3` and not `length(u)-2` this time?

```
> summary(reg)$sigma^2*(length(u)-3)/(length(u)-1)
[1] 0.0005562679
```

The difference seems to be really small

```
> set.seed(007)
> u = runif(100000)
> z = f2(u); reg1=lm(g2(u)~z)
>  coef(reg1)
(Intercept)           z
  0.1042988   0.8411960
> predict(reg1,list(z=0.5))
        1
0.5248968
> uu=runif(100000)
> mean(g2(uu)-coef(reg1)[2]*(f2(uu)-0.5))
[1] 0.524816
> zz=f2(uu); reg2=lm(g2(uu)~zz)
> predict(reg2,list(zz=0.5))
        1
0.5248153
> reg3=reg2; reg3$coefficients[2] = coef(reg1)[2]
> predict(reg3,list(zz=0.5))
        1
0.5251825
```

A little bit more fun

```
> set.seed(007)
> u = runif(10000)
> z=f2(u); y=f1(u)
> reg=lm(g2(u)~z+y); cf=coef(reg)
> u=runif(100000)
> mean(g2(u))
[1] 0.5253908
> mean(g2(u)-cf[2]*(f2(u)-0.5)-cf[3]*(f1(u)-exp(-.5)*(pi/4)))
[1] 0.5248127
> v1=var(g2(u))
> v1
[1] 0.06027686
> v3=var(g2(u)-cf[2]*(f2(u)-0.5)-cf[3]*(f1(u)-exp(-.5)*(pi/4)))
> v3
[1] 0.0005455055
> (v1-v3)/v1
[1] 0.99095
```

Of course, `summary(reg)` does not work now (but is close...)

```
> summary(reg)$sigma^2*(length(u)-3)/(length(u)-1)
[1] 0.0005563681
```

However

However, if we want to maintain unbiasedness in theory, we are still supposed to estimate the regression coefficients in a *preliminary* Monte Carlo run...

...loosing thus the nice possibility of using `lm()`

3. Importance sampling

Simple Monte Carlo: suppose that $f(x) = s\ell(x)$ is a probability density

The integral $\displaystyle\int g(x)\ell(x)\,dx = \frac{1}{s}\int g(x)s\ell(x)\,dx$ is estimated by

$$\frac{1}{sN}\sum_{i=1}^{N} g(X_i) \quad \text{where } X_i \text{ follow the density } s\ell(x) = f(x)$$

In the typical case, $\ell(x) = 1$, $s = \dfrac{1}{v-u}$, so that $f(x)$ is density of the uniform distribution on $[u, v]$ and the integral

$$\int_u^v g(x)\,dx = (v-u)\int_u^v g(x)\frac{1}{v-u}\,dx$$

is estimated by $\displaystyle\frac{v-u}{N}\sum_{i=1}^{N} g(X_i)$

The idea: how about taking for $s\ell$ something better than uniform?

What can be better?

We estimate $\int g(x)\,dx = \int \frac{g(x)}{s\ell(x)} s\ell(x)\,dx = \frac{1}{s}\int \frac{g(x)}{\ell(x)} f(x)\,dx$

by $\frac{1}{sN}\sum_{i=1}^{N} \frac{g(Y_i)}{f(Y_i)}$ where Y_i follow the density $s\ell(x) = f(x)$

Which f will yield "better" estimates here?

Those that will yield smaller $\operatorname{Var}\left(\frac{g(Y_i)}{\ell(Y_i)}\right)$

for instance, those that make $\frac{g(Y_i)}{\ell(Y_i)}$ more "constant"

- which can be interpreted as that f targets more "important" places regarding g - "importance sampling"

Also: is it still an unbiased estimator? Of course - it is pretty much simple Monte Carlo, only with a bit more sophisticated ℓ

A paradox

Think again of the integral $\displaystyle\int_0^1 \frac{1}{\sqrt{x(1-x)}}$

Somebody would tell us: indeed, $\displaystyle\frac{1}{\pi\sqrt{x(1-x)}}$ is a density

of a so-called arc-sine distribution

its name coming from the fact that its cumulative distribution function involves arc-sine function

arc-sine is easily inverted to a sine

so we can easily program a function generating random numbers from this distribution

and then put $s = 1/\pi$ and ℓ be the integrand above

and then we are to integrate a function $h = 1$ over $[0, 1]$

which is sooo easy: $\displaystyle\frac{1}{N}\sum_{i=1}^{n} h(X) = \frac{1}{N}\sum_{i=1}^{n} 1 = 1 \qquad \ldots$

The catch

... and the integral is then 1 multiplied by $1/s = \pi$

- it is π?

The catch is that we knew, more or less, the integral of g

so why compute something we know?

Textbook, Example 6.11 (2019 edition)

Take Guinea pig II $\int_0^1 \frac{e^{-x}}{1+x^2}\,dx$ with five choices for $f(x) = c\ell(x)$

$f_0(x) = 1 \quad 0 < x < 1$ (simple Monte Carlo, uniform f)

$f_1(x) = e^{-x} \quad 0 < x < \infty$ (exponential distribution with $\lambda = 1$)

$f_2(x) = \frac{1}{\pi(1+x^2)} \quad -\infty < x < \infty$ (Cauchy distribution, t_1)

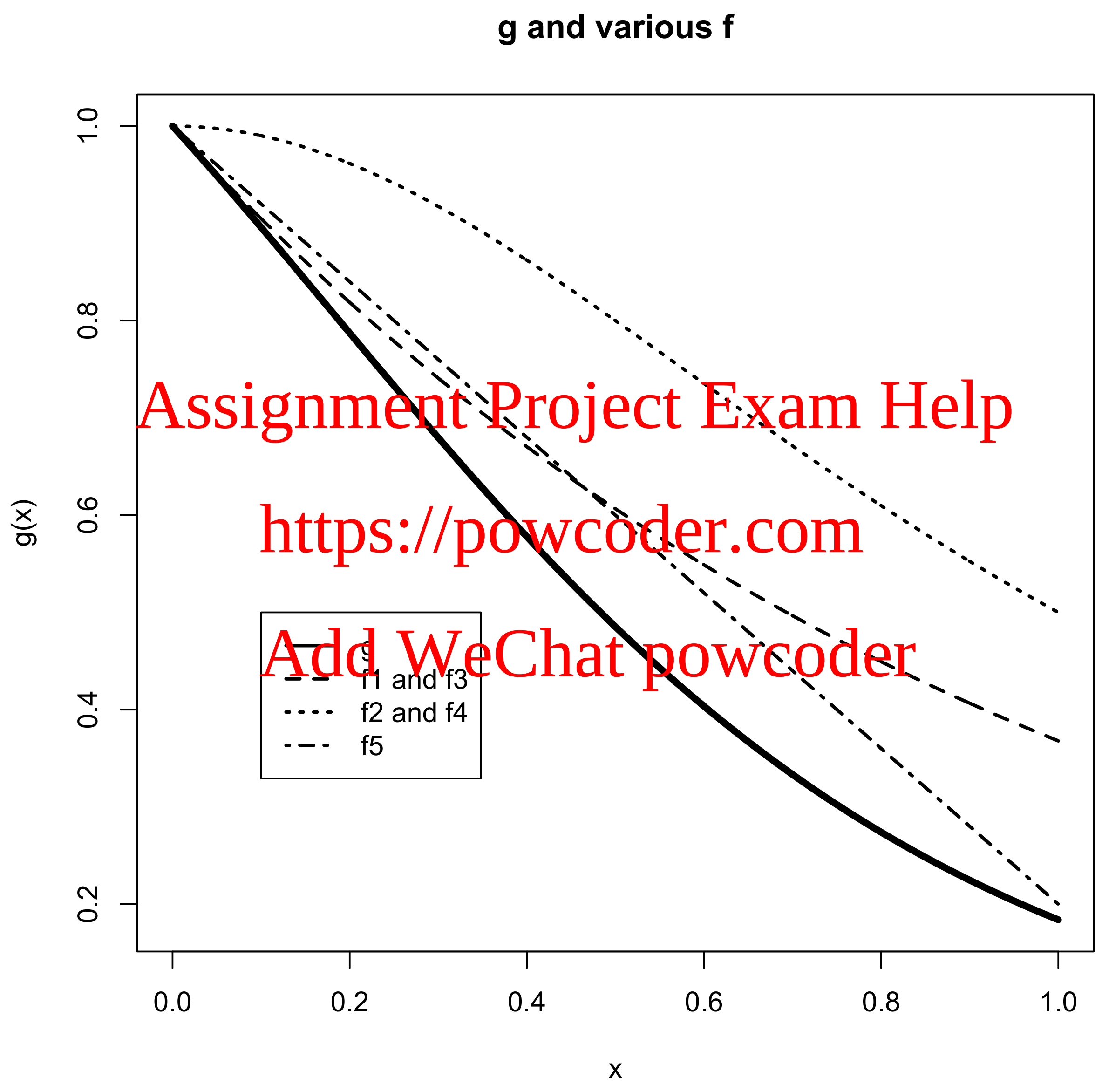
$f_3(x) = ce^{-x} \quad 0 < x < 1$ $(c = 1/(1 - 1/e))$

$f_4(x) = \frac{4}{\pi(1+x^2)} \quad 0 < x < 1$ (a density)

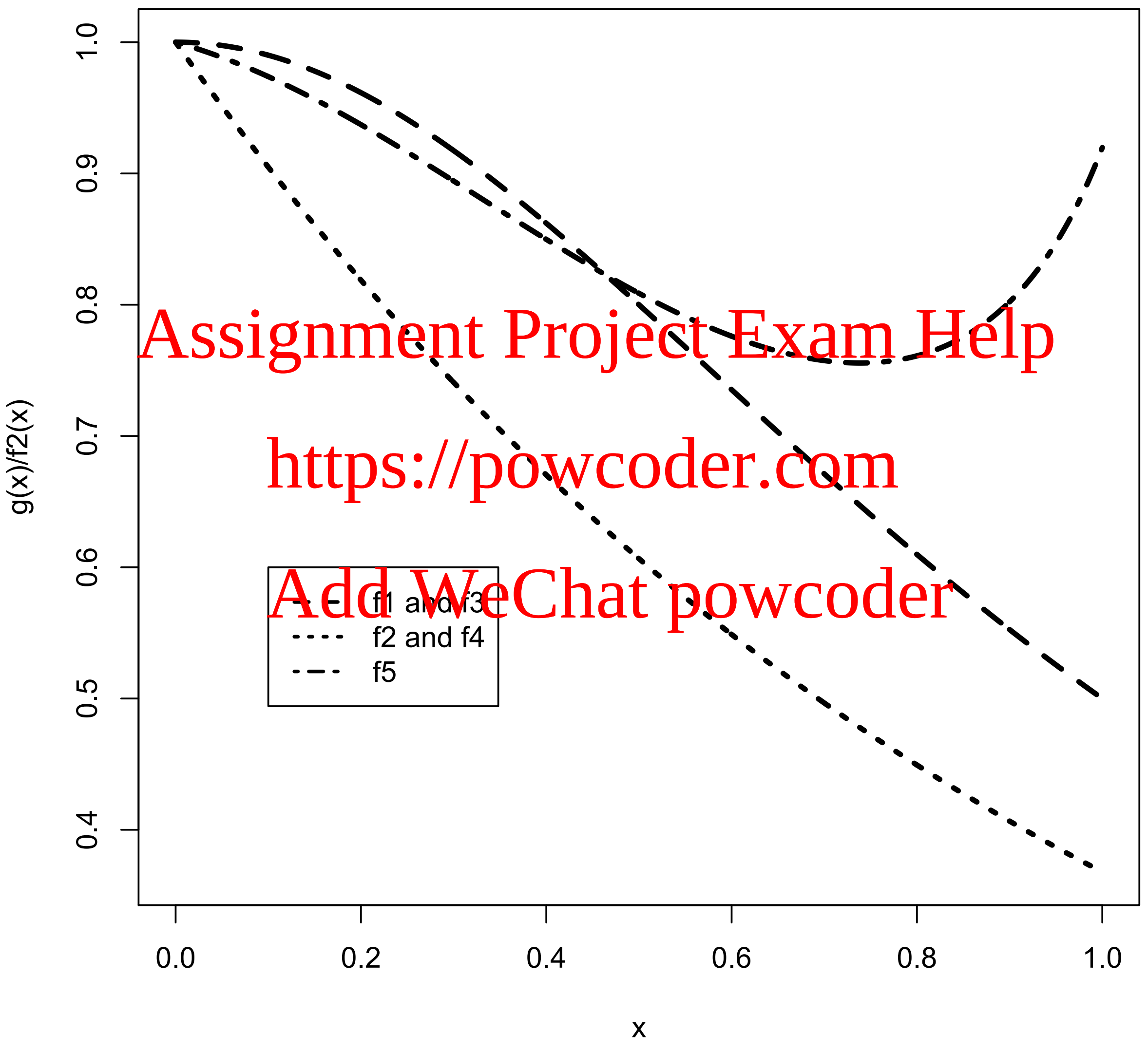
We add one more

$f_5(x) = c(1 - 0.8 * x) \quad 0 < x < 1$ $(c = 1/0.6)$

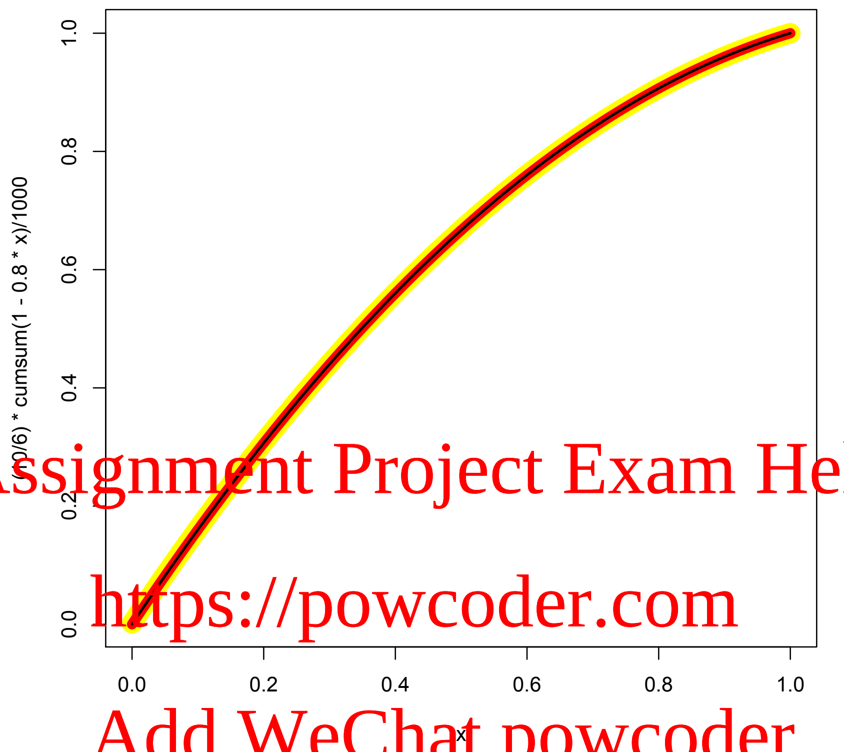
A picture is worth 1000 (or 10000?) words



This one is worth 10000 (plotting g/f)



Digression: and this one may be also helpful



```
> x <- y <- seq(0,1,length=1000)
> plot(x,(10/6)*cumsum(1-0.8*x)/1000,
+ type="l",lwd=16,col="yellow")
> lines(x,(10/6)*(x-0.4*x^2),col="red",lwd=8)
> lines((5-sqrt(25-24*y))/4,y,lwd=2
```

In other words: why not to verify your calculus also numerically

The code, 1st part

```
# m <- 10000
theta.hat <- se <- numeric(6)
g <- function(x) exp(-x)/(1+x^2)*(x>=0)*(x<=1)

f0 <- 1
x <- runif(m)
fg <- g(x)/f0
theta.hat[1] <- mean(fg)
se[1] <- sd(fg)

f1 <- function(x) exp(-x)
x <- rexp(m,1)
fg <- g(x)/f1(x)
theta.hat[2] <- mean(fg)
se[2] <- sd(fg)
```

The code, 2nd part

```
f2 <- function(x) 1/(1+x^2)
con <- 1/pi
x <- rcauchy(m)
i <- c(which(x>1),which(x<0))
x[i] <- 2
fg <- g(x)/f2(x)
theta.hat[3] <- mean(fg)/con
se[3] <- sd(fg)/con

f3 <- f1
con <- 1/(1-exp(-1))
u <- runif(m)
x <- -log(1-u/con)
fg <-  g(x)/f3(x)
theta.hat[4] <- mean(fg)/con
se[4] <- sd(fg)/con
```

The code, 3rd (and last) part

```
f4 <- f2
con <- 4/pi
u <- runif(m)
x <- tan(pi*u/4)
fg <-  g(x)/f4(x)
theta.hat[5] <- mean(fg)/con
se[5] <- sd(fg)/con

f5 <- function(x) 1-0.8*x
con <- 1/(1-0.8/2)
u <- runif(m)
x <- (5-sqrt(25-24*u))/4
fg <-  g(x)/f5(x)
theta.hat[6] <- mean(fg)/con
se[6] <- sd(fg)/con
```

And the Oscar goes to...

```
> m=100000; source("import.R")
> rbind(round(theta.hat,7),round(se/sqrt(m),7),rank(se))
          [,1]      [,2]      [,3]      [,4]      [,5]      [,6]
[1,] 0.5244982 0.5239369 0.5235613 0.5241246 0.5255023 0.5246502
[2,] 0.0007743 0.0013231 0.0030062 0.0003067 0.0004470 0.0001551
[3,] 4.0000000 5.0000000 6.0000000 2.0000000 3.0000000 1.0000000
> source("import.R")
> rbind(round(theta.hat,7),round(se/sqrt(m),7),rank(se))
          [,1]      [,2]      [,3]      [,4]      [,5]      [,6]
[1,] 0.5232864 0.5260607 0.5252418 0.5248760 0.5256166 0.5248048
[2,] 0.0007729 0.0013235 0.0030139 0.0003067 0.0004461 0.0001555
[3,] 4.0000000 5.0000000 6.0000000 2.0000000 3.0000000 1.0000000
> source("import.R")
> rbind(round(theta.hat,7),round(se/sqrt(m),7),rank(se))
          [,1]      [,2]      [,3]      [,4]      [,5]      [,6]
[1,] 0.5255765 0.5248700 0.5247340 0.5250224 0.5245876 0.5248881
[2,] 0.0007743 0.0013232 0.0030035 0.0003072 0.0004457 0.0001554
[3,] 4.0000000 5.0000000 6.0000000 2.0000000 3.0000000 1.0000000
> (se[1]^2-se[6]^2)/se[1]^2
[1] 0.9600556
```

4. Stratified sampling: divide et impera

Divide the domain of integration into several subdomains (some overlaps on boundaries are allowed when the sampled distributions are continuous) and then compute the integrals separately, splitting random numbers among them

As before, $f(x) = s\ell(x)$

So, we are computing $\quad \mu = \int_A g(x)f(x)\,dx$

$$= \int_{A_1} g(x)f(x)\,dx + \int_{A_2} g(x)f(x)\,dx + \int_{A_r} g(x)f(x)\,dx$$

after splitting the domain of integration: $\quad A = A_1 \cup A_2 \cup \cdots \cup A_r$

Some more details and notation

The density f on A_i needs to be made a probability density

$$f_{A_i}(x) = \frac{f(x)}{p_i} \qquad \text{where } p_i = \int_{A_i} f(x)\,dx$$

The integrals in the *strata* are

$$\int_{A_i} g(x)f(x)\,dx = p_i \int_{A_i} g(x)\frac{f(x)}{p_i}\,dx = p_i \int_{A_i} g(x) f_{A_i}(x)\,dx$$

We use the notation $\mu_i = \displaystyle\int_{A_i} g(x) f_{A_i}(x)\,dx$

- the expected value of $g(Y)$

whenever Y follows the distribution with density f_{A_i}

We add also:

$$\sigma_i^2 = \frac{1}{p}\int_{A_i} (g(x) - \mu_i))^2 f(x)\,dx = \int_{A_i} (g(x) - \mu_i))^2 f_{A_i}(x)\,dx$$

- the variance of $g(Y)$

whenever Y follows the distribution with density f_{A_i}

The algorithm

The integral is then computed (estimated) as the sum of the integrals in the strata; the latter are computed separately

with respectively N_i repetitions and

Y_{ij} sampled independently from the distribution with density f_{A_i}:

$$\frac{p_1}{N_1}\sum_{j=1}^{N_1} g(Y_{1j}) + \frac{p_2}{N_2}\sum_{j=1}^{N_2} g(Y_{2j}) + \cdots + \frac{p_r}{N_r}\sum_{j=1}^{N_r} g(Y_{rj})$$

We denote the variance of this sum by $\sigma^2(N_1, N_2, \ldots, N_r)$

It is equal to $$\frac{p_1^2\sigma_1^2}{N_1} + \cdots + \frac{p_r^2\sigma_r^2}{N_r}$$

Is it worth the effort?

We want to compare the variance $\sigma^2(N_1, N_2, \ldots, N_r)$ of the introduced stratified scheme to the variance, denoted by $\sigma^2(N)$,

of the simple Monte Carlo estimator $$\frac{1}{N}\sum_{j=1}^{N} g(X_j)$$

with $N = N_1 + N_2 + \cdots + N_r$ random numbers X_j sampled from f

We can also put it this way: for given N, is there a way to allocate N_i to the strata so that

when $N = N_1 + N_2 + \cdots + N_r$

then $\sigma^2(N_1, N_2, \ldots, N_r) \leqslant \sigma^2(N)$

In particular, is it always possible?

Let us try splitting domain to halves for g_2

```
> set.seed(007)
> u=runif(10000)
> mean(g2(u))
[1] 0.524227
> v1=var(g2(u))/10000
> v1
[1] 5.99499e-06
> u1=u[1:5000]/2
> u2=1/2+u[5001:10000]/2
> (1/2)*mean(g2(u1)) + (1/2)*mean(g2(u2))
[1] 0.5248533
> v2=(1/4)*var(g2(u1))/5000 + (1/4)*var(g2(u2))/5000
> v2
[1] 1.526541e-06
> (v1-v2)/v1
[1] 0.7453639
```

Yes, it is always possible

... and there is a theorem for that - which says:

If $p_1 + p_2 + \cdots + p_r = 1$, then $\sigma^2(Np_1, Np_2, \ldots, Np_r) \leqslant \sigma^2(N)$

Interestingly, it is not the best we can do.

The *best* (minimum variance) allocation is

$$\frac{N\sigma_1 p_1}{\sigma_1 p_1 + \cdots + \sigma_r p_r}, \frac{N\sigma_2 p_2}{\sigma_1 p_1 + \cdots + \sigma_r p_r}, \ldots, \frac{N\sigma_r p_r}{\sigma_1 p_1 + \cdots + \sigma_r p_r}$$

so if we happen to know σ_i ...

Proof of the theorem

$$\sigma^2(N) = \frac{1}{N}\int (g(x)-\mu)^2 f(x)\,dx = \frac{1}{N}\sum_{i=1}^{r}\int_{A_i}(g(x)-\mu)^2 f(x)\,dx$$

$$= \frac{1}{N}\sum_{i=1}^{r}\int_{A_i}(g(x)-\mu_i+\mu_i-\mu)^2 f(x)\,dx$$

$$= \frac{1}{N}\sum_{i=1}^{r}\int_{A_i}(g(x)-\mu_i)^2 f(x)\,dx + \frac{1}{N}\sum_{i=1}^{r}\int_{A_i}(\mu_i-\mu)^2 f(x)\,dx$$

$$+\frac{2}{N}\sum_{i=1}^{r}\int_{A_i}(g(x)-\mu_i)(\mu_i-\mu)f(x)\,dx$$

The last term is $\displaystyle \frac{2}{N}\sum_{i=1}^{r}(\mu_i-\mu)p_i\int_{A_i}(g(x)-\mu_i)\frac{f(x)}{p_i}\,dx$

$$= \frac{2}{N}\sum_{i=1}^{r}(\mu_i-\mu)p_i\left(\int_{A_i} g(x)f_{A_i}(x)\,dx - \mu_i\int_{A_i} f_{A_i}(x)\,dx\right)$$

$$= \frac{2}{N}\sum_{i=1}^{r}(\mu_i-\mu)p_i(\mu_i-\mu_i) = 0$$

Hence (proof continued)

$$
\begin{aligned}
\sigma^2(N) &= \frac{1}{N}\sum_{i=1}^{r}\int_{A_i}(g(x)-\mu_i)^2 f(x)\,dx + \frac{1}{N}\sum_{i=1}^{r}\int_{A_i}(\mu_i-\mu)^2 f(x)\,dx \\
&= \frac{1}{N}\sum_{i=1}^{r} p_i\int_{A_i}(g(x)-\mu_i)^2\frac{f(x)}{p_i}\,dx + \frac{1}{N}\sum_{i=1}^{r}(\mu_i-\mu)^2\int_{A_i} f(x)\,dx \\
&= \frac{1}{N}\sum_{i=1}^{r} p_i\int_{A_i}(g(x)-\mu_i)^2 f_{A_i}(x)\,dx + \frac{1}{N}\sum_{i=1}^{r}(\mu_i-\mu)^2 p_i \\
&= \frac{1}{N}\sum_{i=1}^{r} p_i\sigma_i^2 + \frac{1}{N}\sum_{i=1}^{r}(\mu_i-\mu)^2 p_i = \sum_{i=1}^{r}\frac{p_i^2\sigma_i^2}{Np_i} + \frac{1}{N}\sum_{i=1}^{r}(\mu_i-\mu)^2 p_i \\
&= \sigma^2(Np_1, Np_2, \dots, Np_r) + \frac{1}{N}\sum_{i=1}^{r}(\mu_i-\mu)^2 p_i
\end{aligned}
$$

Which means that $\sigma^2(N) \geqslant \sigma^2(Np_1, Np_2, \dots, Np_r)$, as

$$0 \leqslant \frac{1}{N}\sum_{i=1}^{r}(\mu_i-\mu)^2 p_i$$

(and the difference is bigger if μ_i are more spread)

Note: the ultimate stratification is...

```
> u=runif(200)
> uu=matrix(u/100,100,2)
> mean(g2(u))
[1] 0.5202573
> v1=var(g2(u))
> v1
[1] 0.05879549
> z=seq(0,1,len=101)[-101]
> sum(apply(g2(uu+cbind(z,z)),1,mean)*(1/100))
[1] 0.5248075
> v2=sum(apply(g2(uu+cbind(z,z)),1,var)*(1/10000))
> v2
[1] 6.269604e-08
> (v1-v2)/v1
[1] 0.9999989
```

Finally: is it an unbiased estimator?

That is: what is the expected value of

$$\frac{p_1}{N_1}\sum_{j=1}^{N_i} g(Y_{1j}) + \frac{p_2}{N_2}\sum_{j=1}^{N_2} g(Y_{2j}) + \cdots + \frac{p_r}{N_r}\sum_{j=1}^{N_r} g(Y_{rj})$$

$$E\left(\frac{p_1}{N_1}\sum_{j=1}^{N_i} g(Y_{1j}) + \frac{p_2}{N_2}\sum_{j=1}^{N_2} g(Y_{2j}) + \cdots + \frac{p_r}{N_r}\sum_{j=1}^{N_r} g(Y_{rj})\right) =$$

Some summary at the end

Monte-Carlo: we use random numbers as if they were governed by the rules of probability ($\rightarrow$ Creed of Monte Carlo), the probability we know

to compute ($\rightarrow$ estimate) some quantity

(usually the one that is difficult to obtain otherwise)

Because we believe in probabilistic behavior of random numbers ($\rightarrow$ Creed of Monte Carlo), the mathematical analysis of Monte Carlo is conducted using probability theory, which helps us to figure out

how much our computed results ($\rightarrow$ estimates)

are close to the desired quantities ($\rightarrow$ estimated targets)

The minimal requirement here is *consistency*: it formalizes that

in probabilistic way ($\rightarrow$ convergence in probability)

our estimates are approaching the estimated targets when the number of repetitions grows

Consistency is a minimal requirement here; *without it, our computational efforts are doomed*

Estimating expected values

A common estimated targets are expected values ($\rightarrow$ integrals) of various kinds (including probabilities as a special case)

For this particular case, it is important that the expected values (integral) exists and is finite, as then

> the law of large numbers (Kolmogorov version) provides consistency, once expected values are estimated by averages

It is also important that the estimators are *unbiased* (as averages usually are), because then we can obtain performance guarantees (similar to error estimates in classical numerical analysis) via

> powerful inequalities of probability theory: Chebyshev, Hoeffding

(And so the reason why we need estimators to be unbiased is that all the above involve deviations of a random variable from its expected value)

Performance guarantees for expected values

The Hoeffding inequality gives the sharpest performance guarantees: but it applies only for the averages of random variables that have *bounded range*

The Chebyshev inequality is more general (at the price of being less powerful): it gives nontrivial results whenever the *variance of the estimator* - its crucial component - is finite

When the variance, or its *upper bound*, is calculated exactly, then the performance guarantee has the highest level of exactness

In practice, we may often know the variance, or a bound in the Hoeffding inequality, only approximately - for instance

- we compute them only numerically, up to some precision
- or, which can be seen as a special case of the previous, we estimate them via Monte Carlo

Then we have (very) slightly less exact performance guarantees

Even less exact performance guarantee is obtained if we use a central limit theorem to approximate the distribution of the estimator

The outreach to numerical analysis

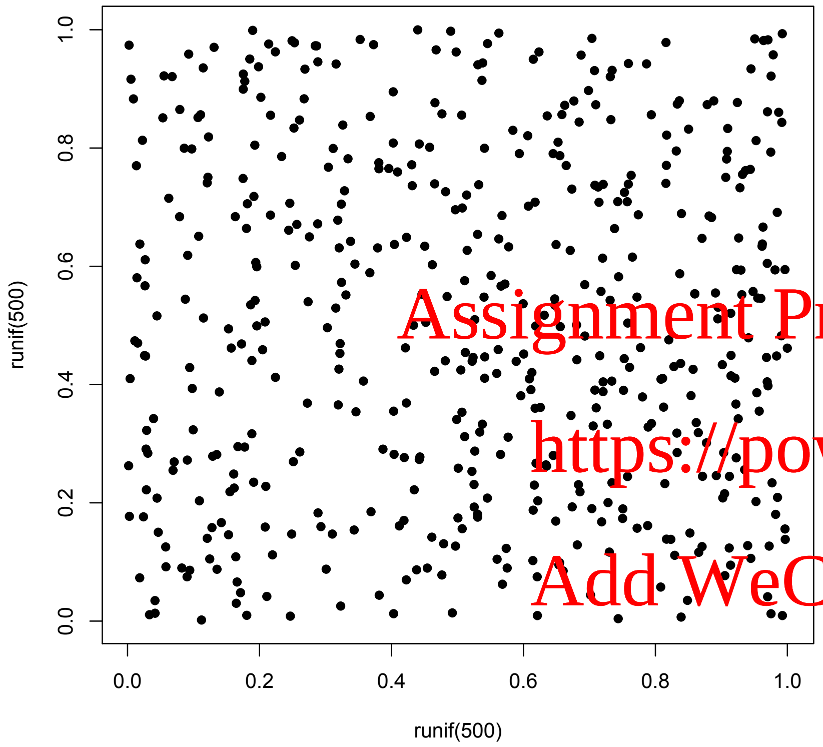
As expected values are integrals, Monte Carlo technology is useful not only for computing statistical and probabilistic quantities, but also as a purely numerical method to compute integrals

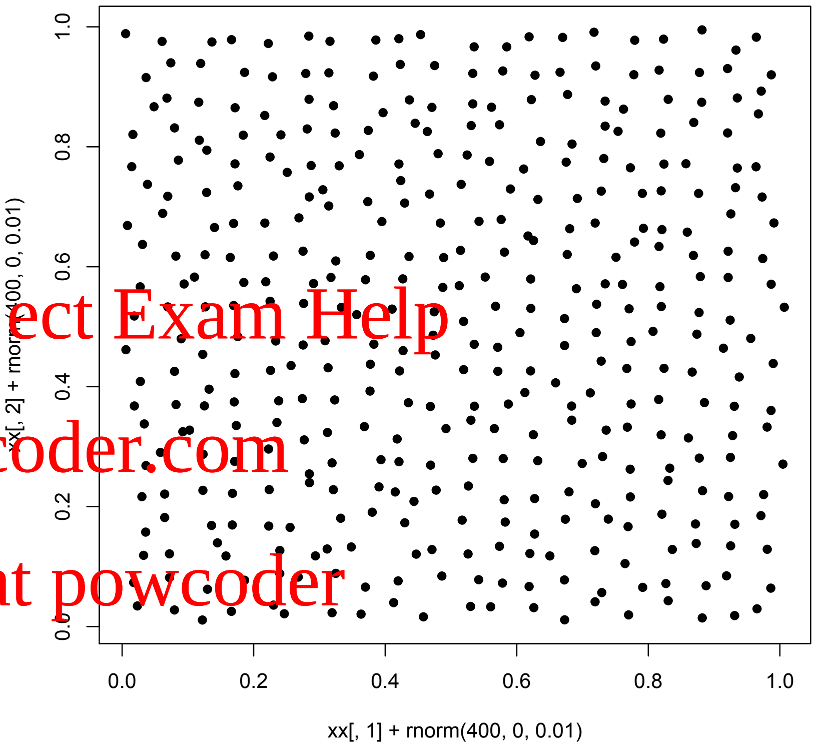
in particular high-dimensional ones

While it can be successful in high-dimensional context compared to classical methods of numerical analysis, they have their limitations too

The success of integration methods based on the evaluation of a function at scattered point depends not only on their probability distribution, but also how evenly the N are spread over the integration domain

Right panel has more evenly spread points





Quasi Monte-Carlo methods

Points that behave as results of *independent* random variables with the same distributions cannot overcome ceratin limits regarding this

Recall: after all, every probability inequality told us that in order to get performance guarantee for another digit of the result - increase the precision by factor $1/10$ - we have to multiply the number of repetitions by 100.

There are methods that perform better - by providing collections of points in higher dimensions that are more evenly spread as those obtained by random sampling: *quasi Monte-Carlo methods*

but these are subject of specialized monographs

(and beyond the scope of this course)