

Integration

Zhentao Shi

Feb 14, 2018

Integration

In their mathematical definitions, integration and differentiation involve limits. However, our computer is a finite-precision machine that can handle neither arbitrarily small nor arbitrarily large numbers; it can, at best, approximate the limiting behavior. In this lecture, we first briefly talk about numerical differentiation and integration, and then we discuss stochastic methods with examples.

Numerical Methods

Numerical differentiation and integration are fundamental topics and of great practical importance. However, how the computer works out these operations has nothing to do with economics or econometrics; it is the content of a numerical analysis course. Here we quickly go over the numerical methods. @judd1998numerical (Chapter 7) is an authoritative reference.

In undergraduate calculus, we have learned the analytic differentiation of many common functions. However, there are cases in which analytic forms are unavailable or too cumbersome to program. For instance, to find the optimum for the objective function $f : R^K \mapsto R$ by Newton's method, in principle we need to code the K -dimensional gradient and the $K \times K$ -dimensional Hessian matrix. Programming up the gradient and the Hessian manually is a time-consuming and error-prone job. What is worse, whenever we change the objective function, which happens often at the experimental stage, we have to redo the gradient and Hessian. Therefore, it is more efficient to use numerical differentiation instead of coding up the analytical expressions, in particular in the trial-and-error stage.

The partial derivative of a multivariate function $f : R^K \mapsto R$ at a point $x_0 \in R^K$ is

$$\left. \frac{\partial f(x)}{\partial x_k} \right|_{x=x_0} = \lim_{\epsilon \rightarrow 0} \frac{f(x_0 + \epsilon \cdot e_k) - f(x_0 - \epsilon \cdot e_k)}{2\epsilon},$$

where $e_k = (0, \dots, 0, 1, 0, \dots, 0)$ is the identifier of the k -th coordinate. The numerical execution in a computer follows the basic definition to evaluate $f(x_0 \pm \epsilon \cdot e_k)$ with a small ϵ . But how small is small? Usually it tries a sequence of ϵ 's until the numerical derivative is stable. There are also more sophisticated algorithms.

In R, the package `numDeriv` conducts numerical differentiation, in which

- `grad` for a scalar-valued function;
- `jacobian` for a real-vector-valued function;
- `hessian` for a scalar-valued function;
- `genD` for a real-vector-valued function.

Integration is, in general, more difficult than differentiation. In R, `integrate` carries out one-dimensional quadrature, and `adaptIntegrate` in the package `cubature` deals with multi-dimensional quadrature. The reader is referred to the documentation for the algorithm behind numerical integrations.

Numerical methods are not panacea. Not all functions are differentiable or integrable. Before turning to numerical methods, it is always imperative to try to understand the behavior of the function at the first place. Some symbolic software, say `Mathematica` or `Wolfram Alpha`, is a powerful tool for this purpose. R is weak in symbolic calculation, if not completely useless.

Stochastic Methods

An alternative to numerical integration is the stochastic methods. The underlying principle of stochastic integration is the law of large numbers. Let $\int h(x)dF(x)$ be an integral where $F(x)$ is a probability distribution. We can approximate the integral by $\int h(x)dF(x) \approx S^{-1} \sum_{s=1}^S h(x_s)$, where x_s is randomly generated from $F(x)$. When S is large, a law of large numbers gives

$$S^{-1} \sum_{s=1}^S h(x_s) \xrightarrow{P} E[h(x)] = \int h(x)dF(x).$$

If the integration is carried out not in the entire support of $F(x)$ but on a subset A , then

$$\int_A h(x)dF(x) \approx S^{-1} \sum_{s=1}^S h(x_s) \cdot 1\{x_s \in A\},$$

where $1\{\cdot\}$ is the indicator function.

In theory, we want to use an S as large as possible. In reality, we are constrained by computing time and power. There is no clear guidance of the size of S in practice. Preliminary experiment can help decide an S that produces stable results.

Stochastic integration is popular in econometrics and statistics, thanks to its convenience in execution.

Example

Structural econometric estimation starts from economic principles. In an economic model, some elements unobservable to the econometrician dictate an economic agent's decision. @roy1951some proposes such a structural model with latent variables, and now the Roy model is the foundation of self-selection in labor economics.

In the original Roy model, a man can work on one of the two occupations: farming or a fishing. The utility of being a farmer is $U_1^* = x'\beta_1 + e_1$ and that of being a fisher is $U_2^* = x'\beta_2 + e_2$, where U_1^* and U_2^* are latent (unobservable). The econometrician observes the binary outcome $y = \mathbf{1}\{U_1^* > U_2^*\}$. If (e_1, e_2) is independent of x , and

$$\begin{bmatrix} e_1 \\ e_2 \end{bmatrix} \sim N \left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} \sigma_1^2 & \sigma_{12} \\ \sigma_{12} & 1 \end{bmatrix} \right)$$

where σ_2 is normalized to be 1, we can write down the log-likelihood as

$$L(\theta) = \sum_{i=1}^n \{y_i \log P(U_{i1}^* > U_{i2}^*) + (1 - y_i) \log P(U_{i1}^* \leq U_{i2}^*)\}.$$

Let $\theta = (\beta_1, \beta_2, \sigma_1, \sigma_{12})$. Given a trial value θ , we can compute

$$p(\theta|x_i) = P(U_{i1}^*(\theta) > U_{i2}^*(\theta)) = P(x_i'(\beta_1 - \beta_2) > e_{i2} - e_{i1}).$$

Under the joint normal assumption, $e_{i2} - e_{i1} \sim N(0, \sigma_1^2 - 2\sigma_{12} + 1)$ so that

$$p(\theta|x_i) = \Phi\left(\frac{x_i'(\beta_1 - \beta_2)}{\sqrt{\sigma_1^2 - 2\sigma_{12} + 1}}\right)$$

where $\Phi(\cdot)$ is the CDF of the standard normal.

However, notice that the analytical form depends on the joint normal assumption and cannot be easily extended to other distributions. As long as the joint distribution of e_{i1}, e_{i2} can be generated from the computer, we can use the stochastic method. We estimate

$$\hat{p}(\theta|x_i) = \frac{1}{S} \sum_{i=1}^S \mathbf{1}(U_{i1}^{s*}(\theta) > U_{i2}^{s*}(\theta)),$$

where $s = 1, \dots, S$ is the index of simulation and S is the total number of simulation replications.

This above example can be viewed as an application of simulated maximum likelihood. In parallel, we can simulate a moment condition if its explicit form is unavailable. @pakes1989simulation provide the theoretical foundation of the simulated method of moments (SMM).

Indirect Inference

Indirect inference [Gourieroux 1993] is yet another simulated-based estimation method. Indirect inference is widely used in estimating structural models. The basic idea of indirect inference is to inversely recover the structural parameter from an *auxiliary model*—usually an reduced-form regression. The reduced-form regression ignores the underlying economic structure and is a purely statistical procedure; thus the reduced-form regression is relatively easier to implement. A *binding function* is a one-to-one mapping from the parameter space of the reduced-form to that of the structural form. Once the reduced-form parameter is estimated, we can recover the structural parameter via the binding function.

The choice of the auxiliary model is not unique. In the Roy model example where θ is the structural parameter, a sensible starting point to construct the auxiliary model is the linear regression between y_i and x_i . A set of reduced-form parameters can be chosen as $\hat{b} = (\hat{b}_1, \hat{b}_2, \hat{b}_3)'$, where

$$\begin{aligned}\hat{b}_1 &= (X'X)^{-1}X'y \\ \hat{b}_2 &= n^{-1} \sum_{y_i=1} (y_i - x_i'b_1)^2 = n^{-1} \sum_{y_i=1} (1 - x_i'b_1)^2 \\ \hat{b}_3 &= n^{-1} \sum_{y_i=0} (y_i - x_i'b_1)^2 = n^{-1} \sum_{y_i=0} (x_i'b_1)^2.\end{aligned}$$

Here \hat{b}_1 is associated with β , and (\hat{b}_2, \hat{b}_3) are associated with (σ_1, σ_{12}) .

Now we consider the structural parameter. Given a trial value θ , the model is parametric and we can simulate artificial error (e_{i1}^*, e_{i2}^*) conditional on x_i . In each simulation experiment, we can decide y_i^* , and we can further estimate the reduced-form parameter $\hat{b}^* = \hat{b}^*(\theta)$ given the artificial data. $b(\theta)$ is the binding function. Conducting such simulation for S times, we measure the distance between \hat{b} and \hat{b}^* as

$$Q(\theta) = \left(\hat{b} - S^{-1} \sum_{s=1}^S \hat{b}^*(\theta)^s \right)' W \left(\hat{b} - S^{-1} \sum_{s=1}^S \hat{b}^*(\theta)^s \right)$$

where s indexes the simulation and W is a positive definite weighting matrix. The indirect inference estimator is $\hat{\theta} = \arg \min_{\theta} Q(\theta)$. That is, we seek the θ that minimizes the distance between the reduced-form parameter from the real data and that from the simulated artificial data.

Markov Chain Monte Carlo

If the CDF $F(X)$ is known, it is easy to generate random variables that follow such a distribution. We can simply compute $X = F^{-1}(U)$, where U is a random draw from $\text{Uniform}(0, 1)$. This X follows the distribution $F(X)$.

If the pdf $f(X)$ is known, we can generate a sample with such a distribution by *importance sampling*. Metropolis-Hastings algorithm (MH algorithm) is such a method. MH is one of the Markov Chain Monte Carlo methods. It can be implemented in the R package `mcmc`.

Metropolis-Hastings Algorithm

The underlying theory of the MH is obscure, but implementation is straightforward. Here we use MH to generate a sample of normally distributed observations. To generate a random variable with other pdf $f(x)$, we just need to replace the line `h = function(x){ y = -x^2 / 2 }` below by `h = function(x){ y = log(f(x)) }`.

```
library(mcmc)
h = function(x){ y = -x^2 / 2 } # the log, non-normalized function

out = metrop( obj = h, initial = 0, nbatch = 100, nspac = 1 )

par(mfrow = c(3,1))
par(mar = c(2,2,1,1))
plot(out$batch, type = "l") # a time series with flat steps

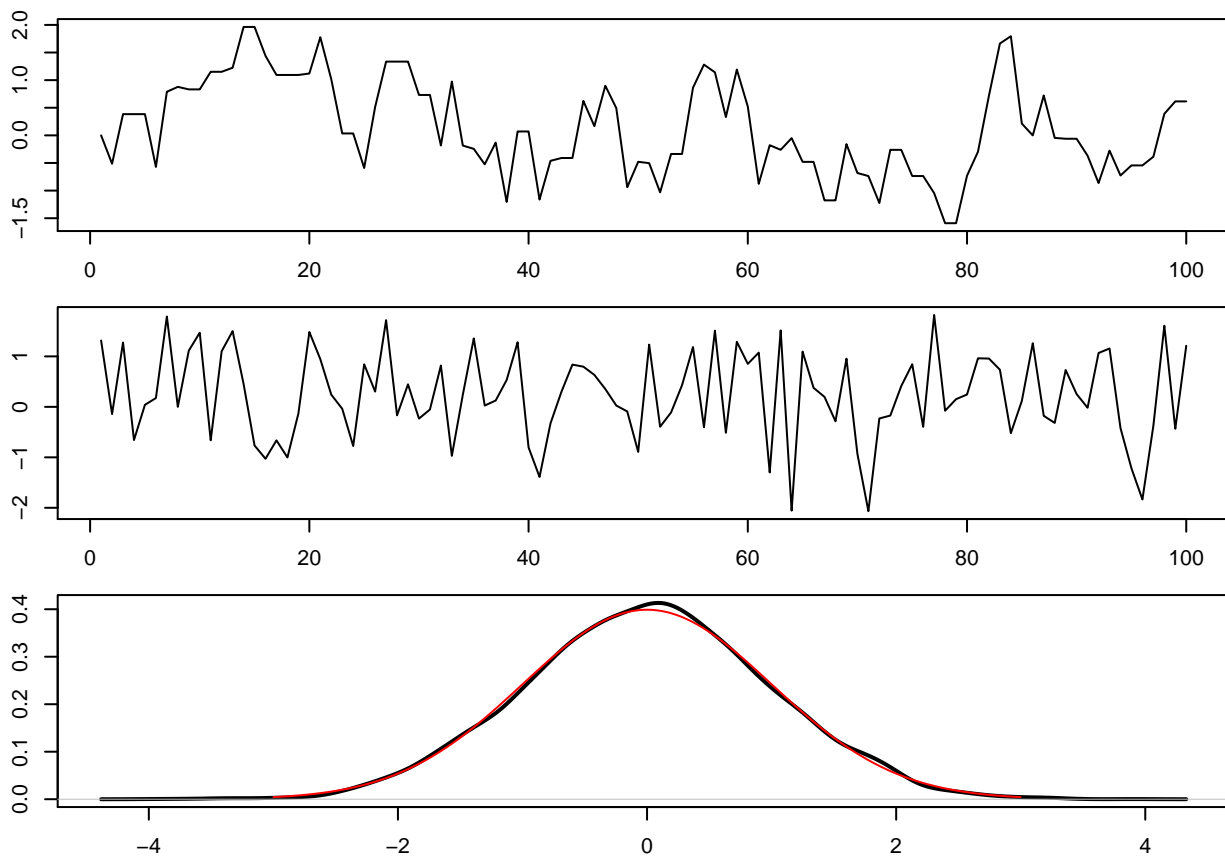
out = metrop( obj = h, initial = 0, nbatch = 100, nspac = 10 )
plot(out$batch, type = "l") # a time series looks like a white noise

out = metrop( obj = h, initial = 0, nbatch = 10000, nspac = 10 )

plot(density(out$batch), main = "", lwd = 2)

xbase = seq(-3, 3, 0.01)
```

```
ynorm = dnorm(xbase)
lines( x = xbase, y = ynorm, type = "l", col = "red")
```



```
summary(out)
```

##	Length	Class	Mode
## accept	1	-none-	numeric
## batch	10000	-none-	numeric
## initial	1	-none-	numeric
## final	1	-none-	numeric
## accept.batch	10000	-none-	numeric
## initial.seed	626	-none-	numeric
## final.seed	626	-none-	numeric
## time	5	proc_time	numeric
## lud	1	-none-	function
## nbatch	1	-none-	numeric
## blen	1	-none-	numeric
## nspac	1	-none-	numeric
## scale	1	-none-	numeric
## debug	1	-none-	logical

Laplace-type Estimator: An Application of MCMC

For some econometric estimators, finding the global optimizer is known to be difficult, because of irregular behavior of the objective function. @chernozhukov2003mcmc's *Laplace-type estimator* (LTE) is an alternative to circumvent the challenge in optimization. LTE transforms the value of the criterion function of an extremum estimator into a probability weight

$$f_n(\theta) = \frac{\exp(-L_n(\theta))\pi(\theta)}{\int_{\Theta} \exp(-L_n(\theta))\pi(\theta)}$$

where $L_n(\theta)$ is an criterion function (say, OLS criterion, (negative) log likelihood criterion, or GMM criterion), and $\pi(\theta)$ is the density of a prior distribution. The smaller is the value of the objective function, the larger it weighs.

We use MCMC to simulate the distribution of θ . From a Bayesian's viewpoint, $f_n(\theta)$ is the posterior distribution. However, @chernozhukov2003mcmc use this distribution for classical estimation and inference, and they justify the procedure via frequentist asymptotic theory. Once $f_n(\theta)$ is known, then *asymptotically* the point estimator equals its mean under the quadratic loss function, and equals its median under the absolute-value loss function.

The code block below compares the OLS estimator with the LTE estimator in a linear regression model.

```
library(mcmc)

# DGP
n = 100
b0 = c(1,2)
X = cbind(1, rnorm(n))
Y = X %*% b0 + rnorm(n)

# Laplace-type estimator
L = function(b) -sum( (Y - X %*% b )^2 )
# criterion function with a flat prior
# notice the "minus" sign of the OLS objective function

out = metrop( obj = L, initial = c(0,0), nbatch = 10000, nspac = 10 )

# summarize the estimation
bhat2 = out$batch[,2]
bhat2_point = mean(bhat2)
bhat2_sd = sd(bhat2)
bhat2_CI = quantile(bhat2, c(.025, .975) )

# compare with OLS
b_OLS = summary( lm(Y~-1+X) )
print(coef( b_OLS ) )

##      Estimate Std. Error  t value    Pr(>|t|)
## X1 0.9764064 0.10281864  9.496395 1.500221e-15
## X2 1.8778976 0.09094901 20.647806 1.813532e-37
```

```
print(c(bhat2_point, bhat2_sd))
```

```
## [1] 1.87862453 0.06407781
```

References