Lecture Notes 4

Numerical differentiation and integration

Numerical integration and differentiation is a key step is a lot of economic applications, among which optimization of utility functions or profits, computation of expectations, incentive problems ....

4.1 Numerical differentiation

In a lot of economic problems and most of all in most of numerical problems we will encounter we will have to compute either *Jacobian* of *Hessian* matrices, in particular in optimization problems or when we will solve non–linear equations problems.

4.1.1 Computation of derivatives

A direct approach

Let us recall that the derivative of a function is given by

$$F'(x) = \lim_{\varepsilon \longrightarrow 0} \frac{F(x+\varepsilon) - F(x)}{\varepsilon}$$

which suggests as an approximation of $F'(x)$

$$F'(x) \simeq \frac{F(x+\Delta_x) - F(x)}{\Delta_x} \tag{4.1}$$

The problem is then: *how big should* Δ_x *be?* It is obvious that Δ_x should be small, in order to be as close as possible to the limit. The problem is that it cannot be too small because of the numerical precision of the computer. Assume for a while that the computer can only deliver a precision of `1e-2` and that we select $\Delta_x = 0.00001$, then $F(x + \Delta_x) - F(x)$ would be 0 for the computer as it would round Δ_x to 0! Theory actually delivers an answer to this problem. Assume that $F(x)$ is computed with accuracy e that is

$$|F(x) - \widehat{F}(x)| \leqslant e$$

where $\widehat{F}(x)$ is the computed value of F. If we compute the derivative using formula (4.1), the computation error is given by

$$\left| \frac{\widehat{F}(x + \Delta_x) - \widehat{F}(x)}{\Delta_x} - \frac{F(x + \Delta_x) - F(x)}{\Delta_x} \right| \leqslant \frac{2e}{\Delta_x}$$

Further, Taylor's expansion theorem states

$$F(x + \Delta_x) = F(x) + F'(x)\Delta_x + \frac{F''(\zeta)}{2}\Delta_x^2$$

for $\zeta \in [x; x + \Delta_x]$. Therefore,[1]

$$\frac{F(x + \Delta_x) - F(x)}{\Delta_x} = F'(x) + \frac{F''(\zeta)}{2}\Delta_x$$

such that the approximation error is

$$\left| \frac{\widehat{F}(x + \Delta_x) - \widehat{F}(x)}{\Delta_x} - F'(x) - \frac{F''(\zeta)}{2}\Delta_x \right| \leqslant \frac{2e}{\Delta_x}$$

suppose now that $\mathscr{M} > 0$ is an upper bound on $|F''|$ in a neighborhood of x, then we have

$$\left| \frac{\widehat{F}(x + \Delta_x) - \widehat{F}(x)}{\Delta_x} - F'(x) \right| \leqslant \frac{2e}{\Delta_x} + \Delta_x \frac{\mathscr{M}}{2}$$

that is the approximation error is bounded above by

$$\frac{2e}{\Delta_x} + \Delta_x \frac{\mathscr{M}}{2}$$

[1] Note that this also indicates that this approximation is $\mathscr{O}(\Delta_x)$.

If we minimize this quantity with respect to Δ_x, we obtain

$$\Delta_x^\star = 2\sqrt{\frac{e}{\mathscr{M}}}$$

such that the upper bound is $2\sqrt{e\mathscr{M}}$. One problem here is that we usually do not know $\mathscr{M}$ However, from a practical point of view, most people use the following scheme for Δ_x

$$\Delta_x = 1e-5.\max(|x|, 1e-8)$$

which essentially amounts to work at the machine precision.

Ajouter ici une discussion de Δ_x

Similarly, rather than taking a forward difference, we may also take the backward difference

$$F'(x) \simeq \frac{F(x) - F(x-\Delta_x)}{\Delta_x} \tag{4.2}$$

Central difference

There are a number of situations where one–sided differences are not accurate enough, one potential solution is then to use the central difference — or two–sided differences — approach that essentially amounts to compute the derivative using the backward–forward formula

$$F'(x) \simeq \frac{F(x+\Delta_x) - F(x-\Delta_x)}{2\Delta_x} \tag{4.3}$$

What do we gain from using this formula? To see this, let us consider the Taylor series expansion of $F(x+\Delta_x)$ and $F(x-\Delta_x)$

$$F(x+\Delta_x) = F(x) + F'(x)\Delta_x + \frac{1}{2}F''(x)\Delta_x^2 + \frac{1}{6}F^{(3)}(\eta_1)\Delta_x^3 \tag{4.4}$$

$$F(x-\Delta_x) = F(x) - F'(x)\Delta_x + \frac{1}{2}F''(x)\Delta_x^2 - \frac{1}{6}F^{(3)}(\eta_2)\Delta_x^3 \tag{4.5}$$

where $\eta_1 \in [x, x+\Delta_x]$ and $\eta_2 \in [x-\Delta_x; x]$. Then, although the error term involves the third derivative at two unknown points on two intervals, assuming

that F is at least $\mathscr{C}^3$, the central difference formula rewrites

$$F'(x) = \frac{F(x+\Delta_x) - F(x-\Delta_x)}{2\Delta_x} + \frac{\Delta_x^2}{6} F^{(3)}(\eta)$$

with $\eta \in [x-\Delta_x; x+\Delta_x]$. A nice feature of this formula is therefore that it is now $\mathscr{O}(\Delta_x^2)$ rather than $\mathscr{O}(\Delta_x)$.

Further improvement: Richardson extrapolation

Basic idea of Richardson Extrapolation There are many approximation procedures in which one first picks a step size h and then generates an approximation $A(h)$ to some desired quantity A. Often the order of the error generated by the procedure is known. This means that the quantity A writes

$$A = A(h) + \Phi h^k + \Phi' h^{k+1} + \Phi'' h^{k+2} + \dots$$

where k is some known constant, called the *order of the error*, and Φ, Φ', Φ', ... are some other (usually unknown) constants. For example, A may be the derivative of a function, $A(h)$ will be the approximation of the derivative when we use a step size of h, and k will be set to 2.

The notation $\mathscr{O}(h^{k+1})$ is conventionally used to stand for "a sum of terms of order h^{k+1} and higher". So the above equation may be written

$$A = A(h) + \Phi h^k + \mathscr{O}(h^{k+1}) \tag{4.6}$$

Dropping the, hopefully tiny, term $\mathscr{O}(h^{k+1})$ from this equation, we obtain a linear equation, $A = A(h) + \Phi h^k$, in the two unknowns A and Φ. But this really gives a different equation for each possible value of h. We can therefore get two different equations to identify both A and Φ by just using two different step sizes. Then doing this , using step sizes h and $h/2$, for any h, and taking 2^k times

$$A = A(h/2) + \Phi(h/2)^k + \mathscr{O}(h^{k+1}) \tag{4.7}$$

(note that, in equations (4.6) and (4.7), the symbol $\mathscr{O}(h^{k+1})$ is used to stand for two different sums of terms of order h^{k+1} and higher) and subtracting

equation (4.6) yields

$$(2^k - 1)A = 2^k A(h/2) - A(h) + \mathscr{O}(h^{k+1})$$

where $\mathscr{O}(h^{k+1})$ stands for a new sum of terms of order h^{k+1} and higher. We then get

$$A = \frac{2^k A(h/2) - A(h)}{2^k - 1} + \mathscr{O}(h^{k+1})$$

where, once again, $\mathscr{O}(h^{k+1})$ stands for a new sum of terms of order h^{k+1} and higher. Denoting

$$B(h) = \frac{2^k A(h/2) - A(h)}{2^k - 1}$$

then

$$A = B(h) + \mathscr{O}(h^{k+1})$$

What have we done so far? We have defined an approximation $B(h)$ whose error is of order $k+1$ rather than k, such that it is a better one than A(h)'s. The generation of a "new improved" approximation for A from two A(h)'s with different values of h is called *Richardson Extrapolation*. We can then continue the process with $B(h)$ to get a new better approximation. This method is widely used when computing numerical integration or numerical differentiation.

Numerical differentiation with Richardson Extrapolation Assume we want to compute the first order derivative of the function $F \in \mathscr{C}^{2n}\mathbb{R}$ at point $x^\star$. We may first compute the approximate quantity:

$$D_0^0(F) = \frac{F(x^\star + h_0) - F(x^\star - h_0)}{2h_0}$$

let us define $h_1 = h_0/2$ and compute

$$D_0^1(F) = \frac{F(x^\star + h_1) - F(x^\star - h_1)}{2h_1}$$

Then according to the previous section, we may compute a better approximation as (since $k = 2$ in the case of numerical differentiation)

$$D_0^1(F) = \frac{4D_0^1(F) - D_0^0(F)}{3}$$

which may actually be rewritten as

$$D_0^1(F) = D_0^1(F) + \frac{D_0^1(F) - D_0^0(F)}{3}$$

We then see that a recursive algorithm occurs as

$$D_j^\ell(F) = D_{\ell-1}^{j+1}(F) + \frac{D_{j+1}^{\ell-1}(F) - D_j^{\ell-1}(F)}{4^k - 1}$$

note that since F is at most $\mathscr{C}^{2n}$, then $k \leqslant n$ such that

$$F'(x^\star) = D_j^k(F) + \mathscr{O}(h_j^{2(k+1)})$$

Hence, $D_j^k(F)$ yields an approximate value for $F'(x^\star)$ with an approximation error proportional to $h_j^{2(k+1)}$. The recursive scheme is carried out until

$$|D_0^m - D_1^{m-1}| < \varepsilon$$

in which case, D_0^m is used as an approximate value for $F'(x^\star)$

MATLAB CODE: RICHARDON EXTRAPOLATION

```
Function D = richardson(f,x,varargin)
%
% f -> function to differentiate
% x -> point at which the function is to be differentiated
% varargin -> parameters of the function
%
delta = 1e-12;   % error goal
toler = 1e-12;   % relative error goal
err   = 1;       % error bound
rerr  = 1;       % relative error
h     = 1;       % initialize step size
j     = 1;       % initialize j
%
% First, compute the first derivative
%
fs    = feval(f,x+h,varargin{:});
fm    = feval(f,x-h,varargin{:});
D(1,1)= (fs-fm)/(2*h);

while (rerr>toler) & (err>delta) & (j<12)

  h  = h/2;   % update the step size
  fs       = feval(f,x+h,varargin{:});
```

```
  fm       = feval(f,x-h,varargin{:});
  D(j+1,1) = (fs-fm)/(2*h);   % derivative with updated step size
  %
  % recursion
  %
  for k = 1:j,
    D(j+1,k+1) = D(j+1,k-1+1) + (D(j+1,k-1+1)-D(j-1+1,k-1+1))/(4^k -1);
  end
  %
  % compute errors
  %
  err  = abs(D(j+1,j+1)-D(j-1+1,j-1+1));
  rerr = 2*err/(abs(D(j+1,j+1))+abs(D(j-1+1,j-1+1))+eps);
  j = j+1;

end

n = size(D,1);
D = D(n,n);
```

4.1.2 Partial Derivatives

Let us now consider that rather than having a single variable function, the problem is multidimensional, such that $F : \mathbb{R}^n \longrightarrow \mathbb{R}$ and that we now want to compute the first order partial derivative

$$F_i(x) = \frac{\partial F(x)}{\partial x_i}$$

This may be achieved extremely easily by computing, for example in the case of central difference formula

$$\frac{F(x + e_i\Delta_x) - F(x - e_i\Delta_x)}{2\Delta_x}$$

where e_i is a vector which $i - -th$ component is 1 and all other elements are 0.

MATLAB CODE: JACOBIAN MATRIX

```
function J=jacobian(func,x0,method,varargin);
%
% [df]=numgrad(func,x0,method,param)
%
% method = 'c' -> centered difference
```

```
%        = 'l' -> left difference
%        = 'r' -> right difference
%
x0      = x0(:);
f       = feval(func,x0,varargin{:});
m       = length(x0);
n       = length(f);
J      = zeros(n,m);
dev = diag(.00001*max(abs(x0),1e-8*ones(size(x0))));
if (lower(method)=='l');
   for i=1:m;
      ff= feval(func,x0+dev(:,i),varargin{:});
      J(:,i)   = (ff-f)/dev(i,i);
   end;
elseif (lower(method)=='r')
   for i=1:m;
      fb= feval(func,x0-dev(:,i),varargin{:});
      J(:,i)   = (f-fb)/dev(i,i);
   end;
elseif (lower(method)=='c')
   for i=1:m;
      ff= feval(func,x0+dev(:,i),varargin{:});
      fb= feval(func,x0-dev(:,i),varargin{:});
      J(:,i)   = (ff-fb)/(2*dev(i,i));
   end;
else
   error('Bad method specified')
end
```

4.1.3 Hessian

Hessian matrix can be computed relying on the same approach as for the Jacobian matrix. Let us consider for example that we want to compute the second order derivative of function $F : \mathbb{R} \longrightarrow \mathbb{R}$ using a central difference approach, as we have seen it delivers higher accuracy. Let us write first write the Taylor's expansion of $F(x + \Delta_x)$ and $F(x - \Delta_x)$ up to order 3

$$
\begin{aligned}
F(x + \Delta_x) &= F(x) + F'(x)\Delta_x + \frac{\Delta_x^2}{2}F''(x) + \frac{\Delta_x^3}{6}F^{(3)}(x) + \frac{\Delta_x^4}{4!}F^{(4)}(\eta_1) \\
F(x - \Delta_x) &= F(x) - F'(x)\Delta_x + \frac{\Delta_x^2}{2}F''(x) - \frac{\Delta_x^3}{6}F^{(3)}(x) + \frac{\Delta_x^4}{4!}F^{(4)}(\eta_2)
\end{aligned}
$$

with $\eta_1 \in [x; x+\Delta_x]$ and $\eta_2 \in [x-\Delta_x; x]$. We then get

$$F(x+\Delta_x) + F(x-\Delta_x) = 2F(x) + \Delta_x^2 F''(x) + \frac{\Delta_x^4}{4!}[F^{(4)}(\eta_1) + F^{(4)}(\eta_2)]$$

such that as long as F is at least $\mathscr{C}^4$, we have

$$F''(x) = \frac{F(x+\Delta_x) - 2F(x) + F(x-\Delta_x)}{\Delta_x^2} - \frac{\Delta_x^2}{12}F^{(4)}(\eta)$$

with $\eta \in [x-\Delta_x; x+\Delta_x]$. Note then that the approximate second order derivative is $\mathscr{O}(\Delta_x^2)$.

4.2 Numerical Integration

Numerical integration is a widely encountered problem in economics. For example, if wa are to compute the welfare function in a continuous time model, we will face an equation of the form

$$W = \int_0^\infty e^{-\rho t} u(c_t) \mathrm{d}t$$

Likewise in rational expectations models, we will have to compute conditional expectations such that — assuming that the innovations of the shocks are gaussian — we will quite often encounter an equation of the form

$$\frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{\infty} f(X, \varepsilon) e^{-\frac{1}{2}\frac{\varepsilon^2}{\sigma^2}} \mathrm{d}\varepsilon$$

In general, numerical integration formulas approximate a definite integral by a weighted sum of function values at points within the interval of integration. In other words, a numerical integration rule takes the typical form

$$\int_a^b F(x)\mathrm{d}x \simeq \sum_{i=0}^{n} \omega_i F(x_i)$$

where the coefficients ω_i depend on the method chosen to compute the integral. This approach to numerical integration is known as the *quadrature* problem. These method essentially differ by *(i)* the weights that are assigned to each function evaluation and *(ii)* the nodes at which the function is evaluated. In fact basic quadrature methods may be categorized in two wide class:

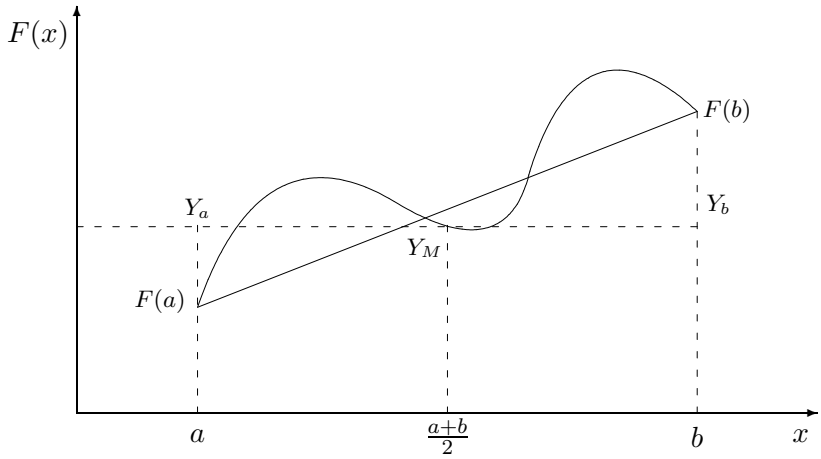
1. The methods that are based on equally spaced data points: these are Newton–cotes formulas: the mid–point rule, the trapezoid rule and Simpson rule.

2. The methods that are based on data points which are not equally spaced: these are Gaussian quadrature formulas.

4.2.1 Newton–Cotes formulas

Newton–Cotes formulas evaluate the function F at a finite number of points and uses this point to build an interpolation between these points — typically a linear approximation in most of the cases. Then this interpolant is integrated to get an approximate value of the integral.

Figure 4.1: Newton–Cotes integration



The mid–point rule

The mid–point rule essentially amounts to compute the area of the rectangle formed by the four points $P_0 = (a, 0)$, $P_1 = (b, 0)$, $P_2 = (a, f(\zeta))$, $P_3 =$

$(b, f(\zeta))$ where $\zeta = (a+b)/2$ as an approximation of the integral, such that

$$\int_a^b F(x)\mathrm{d}x = (b-a)F\left(\frac{a+b}{2}\right) + \frac{(b-a)^3}{4!}F''(\xi)$$

where $\xi \in [a;b]$, such that the approximate integral is given by

$$\widehat{I} = (b-a)F\left(\frac{a+b}{2}\right)$$

Note that this rule does not make any use of the end points. It is noteworthy that this approximation is far too coarse to be accurate, such that what is usually done is to break the interval $[a;b]$ into smaller intervals and compute the approximation on each sub–interval. The integral is then given by cumulating the sub–integrals, we therefore end up with a *composite rule*. Hence, assume that the interval $[a;b]$ is broken into $n \geqslant 1$ sub–intervals of size $h = (b-a)/n$, we have $n+1$ data points $x_i = a + \left(i - \frac{1}{2}\right) h$ with $i = 1, \ldots, n$. The approximate integral is then given by

$$\widehat{I}_n = h \sum_{i=1}^{n} f(x_i)$$

MATLAB CODE: MID–POINT RULE INTEGRATION

```
function mpr=midpoint(func,a,b,n,varargin);
%
% function mpr=midpoint(func,a,b,n,P1,...,Pn);
%
% func          : Function to be integrated
% a             : lower bound of the interval
% b             : upper bound of the interval
% n             : number of sub-intervals => n+1 points
% P1,...,Pn : parameters of the function
%
h   = (b-a)/n;
x   = a+([1:n]'-0.5)*h;
y   = feval(func,x,varargin{:});
mpr = h*(ones(1,n)*y);
```

Trapezoid rule

The trapezoid rule essentially amounts to use a linear approximation of the function to be integrated between the two end points of the interval. This

then defines the trapezoid $\{(a,0),(a,F(a)),(b,F(b)),(b,0)\}$ which area — and consequently the approximate integral — is given by

$$\widehat{I} = \frac{(b-a)}{2}(F(a)+F(b))$$

This may be derived appealing to the Lagrange approximation for function F over the interval $[a;b]$, which is given by

$$\mathscr{L}(x) = \frac{x-b}{a-b}F(a) + \frac{x-a}{b-a}F(b)$$

then

$$\begin{aligned}
\int_a^b F(x)\mathrm{d}x &\simeq \int_a^b \frac{x-b}{a-b}F(a) + \frac{x-a}{b-a}F(b)\mathrm{d}x \\
&\simeq \frac{1}{b-a}\int_a^b (b-x)F(a) + (x-a)F(b)\mathrm{d}x \\
&\simeq \frac{1}{b-a}\int_a^b (bF(a) - aF(b)) + x(F(b)-F(a))\mathrm{d}x \\
&\simeq bF(a) - aF(b) + \frac{1}{b-a}\int_a^b x(F(b)-F(a))\mathrm{d}x \\
&\simeq bF(a) - aF(b) + \frac{b^2-a^2}{2(b-a)}(F(b)-F(a)) \\
&\simeq bF(a) - aF(b) + \frac{b+a}{2}(F(b)-F(a)) \\
&\simeq \frac{(b-a)}{2}(F(a)+F(b))
\end{aligned}$$

Obviously, this approximation may be poor, as in the example reported in figure 4.1, such that as in the mid–point rule we should break the $[a;b]$ interval in $n \geqslant$ sub–intervals of size $h = (b-a)/n$, we have $n+1$ data points $x_i = a+ih$ and their corresponding function evaluations $F(x_i)$ with $i = 0,\ldots,n$. The approximate integral is then given by

$$\widehat{I}_n = \frac{h}{2}\left[F(x_0) + F(x_n) + 2\sum_{i=1}^{n-1} F(x_i)\right]$$

MATLAB CODE: TRAPEZOID RULE INTEGRATION

```
function trap=trapezoid(func,a,b,n,varargin);
%
% function trap=trapezoid(func,a,b,n,P1,...,Pn);
%
% func          : Function to be integrated
% a             : lower bound of the interval
% b             : upper bound of the interval
% n             : number of sub-intervals => n+1 points
% P1,...,Pn : parameters of the function
%
h   = (b-a)/n;
x   = a+[0:n]'*h;
y   = feval(func,x,varargin{:});
trap= 0.5*h*(2*sum(y(2:n))+y(1)+y(n+1));
```

Simpson's rule

The simpson's rule attempts to circumvent an inefficiency of the trapezoid rule: a composite trapezoid rule may be far too coarse if F is smooth. An alternative is then to use a piecewise quadratic approximation of F that uses the values of F at a, b and $(b+a)/2$ as interpolating nodes. Figure 4.2 illustrates the rule. The thick line is the function F to be integrated and the thin line is the quadratic interpolant for this function. A quadratic interpolation may be obtained by the Lagrange interpolation formula, where $\zeta = (b+a)/2$

$$\mathscr{L}(x) = \frac{(x-\zeta)(x-b)}{(a-\zeta)(a-b)}F(a) + \frac{(x-a)(x-b)}{(\zeta-a)(\zeta-b)}F(\zeta) + \frac{(x-a)(x-\zeta)}{(b-a)(b-\zeta)}F(b)$$

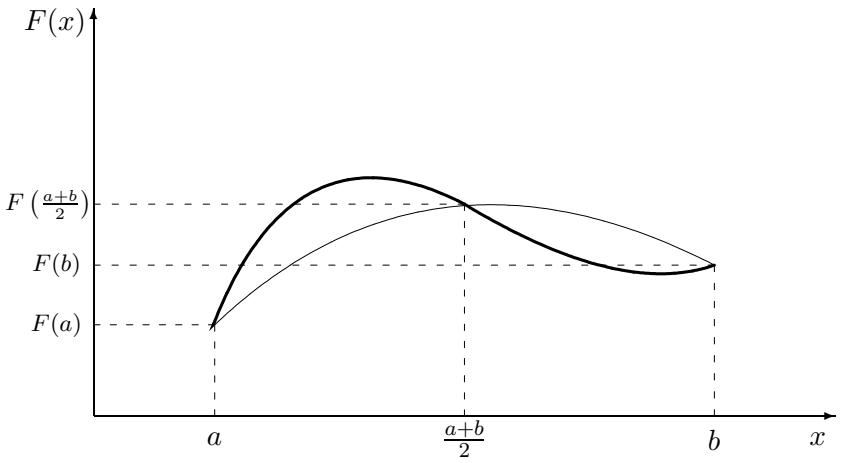
Setting $h = (b-a)/2$ we can approximate the integral by

$$\begin{aligned}\int_a^b F(x)\mathrm{d}x &\simeq \int_a^b \frac{(x-\zeta)(x-b)}{2h^2}F(a) - \frac{(x-a)(x-b)}{h^2}F(\zeta) + \frac{(x-a)(x-\zeta)}{2h^2}F(b)\mathrm{d}x \\ &\simeq I_1 - I_2 + I_3\end{aligned}$$

We then compute each sub–integral

$$\begin{aligned}I_1 &= \int_a^b \frac{(x-\zeta)(x-b)}{2h^2}F(a)\mathrm{d}x \\ &= \frac{F(a)}{2h^2}\int_a^b x^2 - (b+\zeta)x + b\zeta\mathrm{d}x\end{aligned}$$

Figure 4.2: Simpson's rule



$$= \frac{F(a)}{2h^2}\left[\frac{b^3-a^3}{3} - (b+\zeta)\frac{b^2-a^2}{2} + b\zeta(b-a)\right]$$
$$= \frac{F(a)}{12h}(b^2 - 2ba + a^2) = \frac{h}{3}F(a)$$

$$I_2 = \int_a^b \frac{(x-a)(x-b)}{h^2}F(\zeta)\mathrm{d}x$$
$$= \frac{F(\zeta)}{h^2}\int_a^b x^2 - (b+a)x + ab\mathrm{d}x$$
$$= \frac{F(\zeta)}{h^2}\left[\frac{b^3-a^3}{3} - (b+a)\frac{b^2-a^2}{2} + ba(b-a)\right]$$
$$= -\frac{F(\zeta)}{3h}(b-a)^2 = -\frac{4h}{3}F(\zeta)$$

$$I_3 = \int_a^b \frac{(x-\zeta)(x-a)}{2h^2}F(b)\mathrm{d}x$$
$$= \frac{F(b)}{2h^2}\int_a^b x^2 - (a+\zeta)x + a\zeta\mathrm{d}x$$
$$= \frac{F(b)}{2h^2}\left[\frac{b^3-a^3}{3} - (a+\zeta)\frac{b^2-a^2}{2} + a\zeta(b-a)\right]$$

$$= \frac{F(b)}{12h}(b^2 - 2ba + a^2) = \frac{h}{3}F(b)$$

Then, summing the 3 components, we get an approximation of the integral given by

$$\widehat{I} = \frac{b-a}{6}\left[F(a) + 4F\left(\frac{b+a}{2}\right) + F(b)\right]$$

If, like in the mid–point rule and the trapezoid rules we want to compute a better approximation of the integral by breaking $[a;b]$ into $n \geqslant 2$ even number of sub–intervals, we set $h = (b-a)/n$, $x_i = x + ih$, $i = 0, \ldots, n$. Then the composite Simpson's rule is given by

$$\widehat{I}_n = \frac{h}{3}\left[F(x_0) + 4F(x_1) + 2F(x_2) + 4F(x_3) + \ldots + 2F(x_{n-2}) + 4F(x_{n-1}) + F(x_n)\right]$$

MATLAB CODE: SIMPSON'S RULE INTEGRATION

```
function simp=simpson(func,a,b,n,varargin);
%
% function simp=simpson(func,a,b,n,P1,...,Pn);
%
% func          : Function to be integrated
% a             : lower bound of the interval
% b             : upper bound of the interval
% n             : even number of sub-intervals => n+1 points
% P1,...,Pn : parameters of the function
%
h   = (b-a)/n;
x   = a+[0:n]'*h;
y   = feval(func,x,varargin{:});
simp= h*(2*(1+rem(1:n-1,2))*y(2:n)+y(1)+y(n+1))/3;
```

Infinite domains and improper integrals

The methods we presented so far were defined over finite domains, but it will be often the case — at least when we will be dealing with economic problems — that the domain of integration is infinite. We will now investigate how we can transform the problem to be able to use standard methods to compute the integrals. Nevertheless, we have to be sure that the integral is well defined.

For example, let us consider the integral

$$\int_{-\infty}^{\infty} F(x)\mathrm{d}x$$

it may not exist because of either divergence — if $\lim_{x\to\pm\infty} F(x) = \pm\infty$ — or because of oscillations as in $\int_{-\infty}^{\infty} \sin(x)\mathrm{d}x$. Let us restrict ourselves to the case where the integral exists. In this case, we can approximate

$$\int_{-\infty}^{\infty} F(x)\mathrm{d}x$$

by

$$\int_{a}^{b} F(x)\mathrm{d}x$$

setting a and b too large enough negative and positive values. However, this may be a particularly slow way of approximating the integral, and the next theorem provides a indirect way to achieve higher efficiency.

Theorem 1 *If $\Phi : \mathbb{R} \longrightarrow \mathbb{R}$ is a monotonically increasing, $\mathscr{C}^1$, function on the interval $[a;b]$ then for any integrable function $F(x)$ on $[a;b]$ we have*

$$\int_{a}^{b} F(x)\,dx = \int_{\Phi^{-1}(a)}^{\Phi^{-1}(b)} F(\Phi(y))\Phi'(y)\,dy$$

This theorem is just what we usually call a change of variables, and convert a problem where we want to integrate a function in the variable x into a perfectly equivalent problem where we integrate with regards to y, with y and x being related by the non–linear relation: $x = \Phi(y)$.

As an example, let us assume that we want to compute the average of the transformation of a gaussian random variable $x \rightsquigarrow \mathscr{N}(0,1)$. This is given by

$$\frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} G(x) e^{-\frac{x^2}{2}} \mathrm{d}x$$

such that $F(x) = G(x)e^{-\frac{x^2}{2}}$. As a first change of variable, that will leave the interval unchanged, we will apply the transformation $z = x/\sqrt{2}$ such that the integral rewrites

$$\frac{1}{\sqrt{\pi}} \int_{-\infty}^{\infty} G(\sqrt{2}z) e^{-z^2} \mathrm{d}z$$

We would like to transform this problem since it would be quite difficult to compute this integral on the interval $[a, b]$ and set both a and b to large negative and positive values. Another possibility is to assume that we compute the integral of a transformed problem over the interval $[a; b]$. We therefore look for a $\mathscr{C}^1$, monotonically increasing transformation Φ that would insure that $\lim_{y\to a} \Phi(y) = -\infty$ and $\lim_{y\to b} \Phi(y) = \infty$. Let us assume that $a = 0$ and $b = 1$, a possible candidate for $\Phi(y)$ is

$$\Phi(y) = \log\left(\frac{y}{1-y}\right) \text{ such that } \Phi'(y) = \frac{1}{y(1-y)}$$

In this case, the integral rewrites

$$\frac{1}{\sqrt{\pi}} \int_0^1 F\left(\sqrt{2}\log\left(\frac{y}{1-y}\right)\right) \frac{1}{y(1-y)} \mathrm{d}y$$

or

$$\frac{1}{\sqrt{\pi}} \int_0^1 \left(\frac{1-y}{y}\right)^{\log((1-y)/y)} G\left(\sqrt{2}\log\left(\frac{y}{1-y}\right)\right) \frac{1}{y(1-y)} \mathrm{d}y$$

which is now equivalent to compute a simple integral of the form

$$\int_0^1 h(y)\mathrm{d}y$$

with

$$h(y) \equiv \frac{1}{\sqrt{\pi}} \left(\frac{1-y}{y}\right)^{\log(y/(1-y))} G\left(\sqrt{2}\log\left(\frac{y}{1-y}\right)\right) \frac{1}{y(1-y)}$$

Table 4.1 reports the results for the different methods we have seen so far. As can be seen, the mid–point and the trapezoid rule perform pretty well with 20 sub–intervals as the error is less than `1e-4`, while the Simpson rule is less efficient as we need 40 sub–intervals to be able to reach a reasonable accuracy. We will see in the next section that there exist more efficient methods to deal with this type of problem.

Note that not all change of variable are admissible. Indeed, in this case we might have used $\Phi(y) = \log(y/(1-y))^{1/4}$, which also maps $[0; 1]$ into $\mathbb{R}$ in a monotonically increasing way. But this would not have been an admissible

Table 4.1: Integration with a change in variables: True value=exp(0.5)

n	Mid–point	Trapezoid	Simpson
2	2.2232	1.1284	1.5045
	(-0.574451)	(0.520344)	(0.144219)
4	1.6399	1.6758	1.8582
	(0.0087836)	(-0.0270535)	(-0.209519)
8	1.6397	1.6579	1.6519
	(0.00900982)	(-0.00913495)	(-0.0031621)
10	1.6453	1.6520	1.6427
	(0.00342031)	(-0.00332232)	(0.00604608)
20	1.6488	1.6487	1.6475
	(-4.31809e-005)	(4.89979e-005)	(0.00117277)
40	1.6487	1.6487	1.6487
	(-2.92988e-006)	(2.90848e-006)	(-1.24547e-005)

transformation. *Why?* Remember that any approximate integration has an associated error bound that depends on the derivatives of the function to be integrated (the overall $h(.)$ function). If the derivatives of $h(.)$ are well defined when y tends towards 0 or 1 in the case we considered in our experiments, this is not the case for the latter case. In particular, the derivatives are found to diverge as y tends to 1, such that the error bound does not converge. In others, we always have to make sure that the derivatives of $F(\Phi(y))\Phi'(y)$ have to be defined over the interval.

4.2.2 Gaussian quadrature

As we have seen from the earlier examples, Newton–Cotes formulas actually derives from piecewise interpolation theory, as they just use a collection of low order polynomials to get an approximation for the function to be integrated and then integrate this approximation — which is in general far easier. These formulas also write

$$\int_a^b F(x)\mathrm{d}x \simeq \sum_{i=1}^{n} \omega_i F(x_i)$$

for some *quadrature nodes* $x_i \in [a;b]$ and *quadrature weights* ω_i. All x_is are arbitrarily set in Newton–Cotes formulas, as we have seen we just imposed a equally spaced grid over the interval $[a;b]$. Then the weights ω_i follow from the fact that we want the approximation to be equal for a polynomial of order lower or equal to the degree of the polynomials used to approximate the function. The question raised by Gaussian Quadrature is then *Isn't there a more efficient way to set the nodes and the weights?* The answer is clearly **Yes**. The key point is then to try to get a good approximation to $\int F(x)\mathrm{d}x$. The problem is *what is a good approximation?* Gaussian quadrature sets the nodes and the weights in such a way that the approximation is exact when F is a low order polynomial.

In fact, Gaussian quadrature is a much more general than simple integration, it actually computes an approximation to the weighted integral

$$\int_a^b F(x)w(x)\mathrm{d}x \simeq \sum_{i=1}^{n} \omega_i F(x_i)$$

Gaussian quadrature imposes that the last approximation is exact when F is a polynomial of order $2n-1$. Further, the nodes and the weights are contingent on the weighting function. Then orthogonal polynomials are expected to come back in the whole story. This is stated in the following theorem by Davis and Rabinowitz [1984]

Theorem 2 *Assume $\{\varphi_\ell(x)\}_{\ell=0}^{\infty}$ is an orthonormal family of polynomials with respect to the weighting function $w(x)$ on the interval $[a;b]$, and define α_ℓ so that $\varphi_\ell(x) = \alpha_k x^k + \ldots$. Let x_i, $i = 1, \ldots, n$ be the roots of the polynomial $\varphi_n(x)$. If $a < x_1 < \ldots < x_n < b$ and if $F \in \mathscr{C}^{2n}[a;b]$, then*

$$\int_a^b w(x)F(x)dx = \sum_{i=1}^{n} \omega_i F(x_i) + \frac{F^{(2n)}(\zeta)}{\alpha_n^2 (2n)!}$$

for $\zeta \in [a;b]$ with

$$\omega_i = -\frac{\alpha_{n+1}/\alpha_n}{\varphi_n'(x)\varphi_{n+1}(x)} > 0$$

This theorem is of direct applicability, as it gives for any weighting function a general formula for both the nodes and the weights. Fortunately, most of the job has already been done, and there exist Gaussian quadrature formulas for a wide spectrum of weighting function, and the values of the nodes and the weights are given in tables. Assume we have a family of orthogonal polynomials, $\{\varphi_\ell(x)\}_{\ell=0}^n$, we know that for any $i \neq j$

$$\langle \varphi_i(x), \varphi_j(x) \rangle = 0$$

In particular, we have

$$\langle \varphi_i(x), \varphi_0(x) \rangle = \int_a^b \varphi_k(x)\varphi_0(x)w(x)\mathrm{d}x = 0 \text{ for } i > 0$$

but since the orthogonal polynomial of order 0 is 1, this reduces to

$$\int_a^b \varphi_k(x)w(x)\mathrm{d}x = 0 \text{ for } i > 0$$

We will take advantage of this property. The nodes will be the roots of the orthogonal polynomial of order n, while the weights will be chosen such that the gaussian formulas is exact for lower order polynomials

$$\int_a^b \varphi_k(x)w(x)\mathrm{d}x = \sum_{i=1}^n \omega_i\varphi_k(x_i) \text{ for } k = 0, \ldots, n-1$$

This implies that the weights can be recovered by solving a linear system of the form

$$\left\{ \begin{array}{rcl} \omega_1\varphi_0(x_1) + \ldots + \omega_n\varphi_0(x_n) & = & \int_a^b w(x)\mathrm{d}x \\ \omega_1\varphi_1(x_1) + \ldots + \omega_n\varphi_1(x_n) & = & 0 \\ & \vdots & \\ \omega_1\varphi_{n-1}(x_1) + \ldots + \omega_{n-1}\varphi_n(x_n) & = & 0 \end{array} \right.$$

which rewrites

$$\Phi\omega = \Psi$$

with

$$\Phi = \begin{pmatrix} \varphi_0(x_1) & \cdots & \varphi_0(x_n) \\ \vdots & \ddots & \vdots \\ \varphi_{n-1}(x_1) & \cdots & \varphi_{n-1}(x_n) \end{pmatrix}, \omega = \begin{pmatrix} \omega_1 \\ \vdots \\ \omega_n \end{pmatrix} \text{ and } \Psi = \begin{pmatrix} \int_a^b w(x)\mathrm{d}x \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

Note that the orthogonality property of the polynomials imply that the Φ matrix is invertible, such that $\omega = \Phi^{-1}\Psi$. We now review the most commonly used Gaussian quadrature formulas.

Gauss–Chebychev quadrature

This particular quadrature can be applied to problems that takes the form

$$\int_{-1}^{1} F(x)(1-x^2)^{-\frac{1}{2}}\mathrm{d}x$$

such that in this case $w(x) = (1-x^2)^{-\frac{1}{2}}$ and $a = -1$, $b = 1$. The very attractive feature of this gaussian quadrature is that the weight is constant and equal to $\omega_i = \omega = \pi/n$, where n is the number of nodes, such that

$$\int_{-1}^{1} F(x)(1-x^2)^{-\frac{1}{2}}\mathrm{d}x = \frac{\pi}{n}\sum_{i=1}^{n} F(x_i) + \frac{\pi}{2^{2n-1}}\frac{F^{(2n)}(\zeta)}{(2n)!}$$

for $\zeta \in [-1;1]$ and where the nodes are given by the roots of the Chebychev polynomial of order n:

$$x_i = \cos\left(\frac{2i-1}{2n}\pi\right) \quad i = 1,\ldots,n$$

It is obviously the case that we rarely have to compute an integral that exactly takes the form this quadrature imposes, and we are rather likely to compute

$$\int_{a}^{b} F(x)\mathrm{d}x$$

Concerning the bounds of integration, we may use the change of variable

$$y = 2\frac{x-a}{b-a} - 1 \text{ implying } \mathrm{d}y = \frac{2\mathrm{d}x}{b-a}$$

such that the problem rewrites

$$\frac{b-a}{2}\int_{-1}^{1} F\left(a + \frac{(y+1)(b-a)}{2}\right)\mathrm{d}y$$

The weighting matrix is still missing, nevertheless multiplying and dividing the integrand by $(1-y^2)^{-\frac{1}{2}}$, we get

$$\frac{b-a}{2}\int_{-1}^{1} G(y)\frac{1}{\sqrt{1-y^2}}\mathrm{d}y$$

with

$$G(y) \equiv F\left(a+\frac{(y+1)(b-a)}{2}\right)\sqrt{1-y^2}$$

such that

$$\int_a^b F(x)\mathrm{d}x \simeq \frac{\pi(b-a)}{2n}\sum_{i=1}^{n} F\left(a+\frac{(y_i+1)(b-a)}{2}\right)\sqrt{1-y_i^2}$$

where y_i, $i=1,\ldots,n$ are the n Gauss–Chebychev quadrature nodes.

Gauss–Legendre quadrature

This particular quadrature can be applied to problems that takes the form

$$\int_{-1}^{1} F(x)\mathrm{d}x$$

such that in this case $w(x)=1$ and $a=-1$, $b=1$. We are therefore back to a standard integration problem as the weighting function is constant and equal to 1. We then have

$$\int_{-1}^{1} F(x)\mathrm{d}x = \sum_{i=1}^{n}\omega_i F(x_i) + \frac{2^{2n+1}(n!)^4}{(2n+1)!(2n)!}\frac{F^{(2n)}(\zeta)}{(2n)!}$$

for $\zeta \in [-1;1]$. In this case, both the nodes and the weights are non trivial to compute. Nevertheless, we can generate the nodes using any root finding procedure, and the weights can be computed as explained earlier, noting that $\int_{-1}^{1} w(x)\mathrm{d}x = 2$.

Like in the case of Gauss–Chebychev quadrature, we may use the linear transformation

$$y = 2\frac{x-a}{b-a} - 1 \text{ implying } \mathrm{d}y = \frac{2\mathrm{d}x}{b-a}$$

to be able to compute integrals of the form

$$\int_a^b F(x)\mathrm{d}x$$

which is then approximated by

$$\int_a^b F(x)\mathrm{d}x \simeq \frac{b-a}{2}\sum_{i=1}^{n}\omega_i F\left(a+\frac{(y_i+1)(b-a)}{2}\right)$$

where y_i and ω_i are the Gauss–Legendre nodes and weights over the interval $[a;b]$.

Such a simple formula has a direct implication when we want to compute the discounted value of a an asset, the welfare of an agent or the discounted sum of profits in a finite horizon problem, as it can be computed solving the integral

$$\int_0^T e^{-\rho t}u(c(t)) \text{ with } T<\infty$$

in the case of the welfare of an individual or

$$\int_0^T e^{-rt}\pi(x(t)) \text{ with } T<\infty$$

in the case of a profit function. However, it will be often the case that we will want to compute such quantities in an infinite horizon model, something that this quadrature method cannot achieve unless considering a change of variable of the kind we studied earlier. Nevertheless, there exists a specific Gaussian quadrature that can achieve this task.

As an example of the potential of Gauss–Legendre quadrature formula, we compute the welfare function of an individual that lives an infinite number of period. Time is continuous and the welfare function takes the form

$$W=\int_0^T e^{-\rho t}\frac{c(t)^\theta}{\theta}\mathrm{d}t$$

where we assume that $c(t)=c^\star e^{-\alpha t}$. Results for n=2, 4, 8 and 12 and T=10, 50, 100 and 1000 (as an approximation to ∞) are reported in table 4.3, where

we set $\alpha = 0.01$, $\rho = 0.05$ and $c^\star = 1$. As can be seen from the table, the integral converges pretty fast to the true value as the absolute error is almost zero for $n \geqslant 8$, except for T=1000. Note that even with $n = 4$ a quite high level of accuracy can be achieved in most of the cases.

Gauss–Laguerre quadrature

This particular quadrature can be applied to problems that takes the form

$$\int_0^\infty F(x)e^{-x}\mathrm{d}x$$

such that in this case $w(x) = e^{-x}$ and $a = 0$, $b = \infty$. The quadrature formula is then given by

$$\int_0^\infty F(x)e^{-x}\mathrm{d}x = \sum_{i=1}^{n} \omega_i F(x_i) + \frac{(n!)^2}{(2n+1)!(2n)!}\frac{F^{(2n)}(\zeta)}{(2n)!}$$

for $\zeta \in [0;\infty)$. In this case, like in the Gauss–Legendre quadrature, both the nodes and the weights are non trivial to compute. Nevertheless, we can generate the nodes using any root finding procedure, and the weights can be computed as explained earlier, noting that $\int_0^\infty w(x)\mathrm{d}x = 1$.

A direct application of this formula is that it can be used to to compute the discounted sum of any quantity in an infinite horizon problem. Consider for instance the welfare of an individual, as it can be computed solving the integral once we know the function $c(t)$

$$\int_0^\infty e^{-\rho t}u(c(t))\mathrm{d}t$$

The problem involves a discount rate that should be eliminated to stick to the exact formulation of the Gauss–Laguerre problem. Let us consider the linear map $y = \rho t$, the problem rewrites

$$\int_0^\infty e^{-y}u\left(c\left(\frac{y}{\rho}\right)\right)\frac{\mathrm{d}y}{\rho}$$

and can be approximated by

$$\frac{1}{\rho}\sum_{i=1}^{n}\omega_i F\left(\frac{y_i}{\rho}\right)$$

Table 4.2: Welfare in finite horizon

n	$\theta = -2.5$	$\theta = -1$	$\theta = 0.5$	$\theta = 0.9$
		T=10		
2	-3.5392	-8.2420	15.3833	8.3929
	(-3.19388e-006)	(-4.85944e-005)	(0.000322752)	(0.000232844)
4	-3.5392	-8.2420	15.3836	8.3931
	(-3.10862e-014)	(-3.01981e-012)	(7.1676e-011)	(6.8459e-011)
8	-3.5392	-8.2420	15.3836	8.3931
	(0)	(1.77636e-015)	(1.77636e-015)	(-1.77636e-015)
12	-3.5392	-8.2420	15.3836	8.3931
	(-4.44089e-016)	(0)	(3.55271e-015)	(1.77636e-015)
		T=50		
2	-11.4098	-21.5457	33.6783	17.6039
	(-0.00614435)	(-0.0708747)	(0.360647)	(0.242766)
4	-11.4159	-21.6166	34.0389	17.8467
	(-3.62327e-008)	(-2.71432e-006)	(4.87265e-005)	(4.32532e-005)
8	-11.4159	-21.6166	34.0390	17.8467
	(3.55271e-015)	(3.55271e-015)	(7.10543e-015)	(3.55271e-015)
12	-11.4159	-21.6166	34.0390	17.8467
	(-3.55271e-015)	(-7.10543e-015)	(1.42109e-014)	(7.10543e-015)
		T=100		
2	-14.5764	-23.6040	32.5837	16.4972
	(-0.110221)	(-0.938113)	(3.63138)	(2.28361)
4	-14.6866	-24.5416	36.2078	18.7749
	(-1.02204e-005)	(-0.000550308)	(0.00724483)	(0.00594034)
8	-14.6866	-24.5421	36.2150	18.7808
	(3.55271e-015)	(-1.03739e-012)	(1.68896e-010)	(2.39957e-010)
12	-14.6866	-24.5421	36.2150	18.7808
	(-5.32907e-015)	(-1.77636e-014)	(2.84217e-014)	(1.77636e-014)
		T=1000		
2	-1.0153	-0.1066	0.0090	0.0021
	(-14.9847)	(-24.8934)	(36.3547)	(18.8303)
4	-12.2966	-10.8203	7.6372	3.2140
	(-3.70336)	(-14.1797)	(28.7264)	(15.6184)
8	-15.9954	-24.7917	34.7956	17.7361
	(-0.00459599)	(-0.208262)	(1.56803)	(1.09634)
12	-16.0000	-24.9998	36.3557	18.8245
	(-2.01256e-007)	(-0.000188532)	(0.00798507)	(0.00784393)

where y_i and ω_i are the Gauss–Laguerre nodes and weights over the interval $[0; \infty)$.

As an example of the potential of Gauss–Laguerre quadrature formula, we compute the welfare function of an individual that lives an infinite number of period. Time is continuous and the welfare function takes the form

$$W = \int_0^\infty e^{-\rho t} \frac{c(t)^\theta}{\theta} \mathrm{d}t$$

where we assume that $c(t) = c^\star e^{-\alpha t}$. Results for n=2, 4, 8 and 12 are reported in table 4.3, where we set $\alpha = 0.01$, $\rho = 0.05$ and $c^\star = 1$. As can be seen from the table, the integral converges pretty fast to the true value as the absolute error is almost zero for $n \geqslant 8$. It is worth noting that the method performs far better than the Gauss–Legendre quadrature method with T=1000. Note that even with $n = 4$ a quite high level of accuracy can be achieved in some cases.

Table 4.3: Welfare in infinite horizon

n	$\theta = -2.5$	$\theta = -1$	$\theta = 0.5$	$\theta = 0.9$
2	-15.6110	-24.9907	36.3631	18.8299
	(0.388994)	(0.00925028)	(0.000517411)	(0.00248525)
4	-15.9938	-25.0000	36.3636	18.8324
	(0.00622584)	(1.90929e-006)	(3.66246e-009)	(1.59375e-007)
8	-16.0000	-25.0000	36.3636	18.8324
	(1.26797e-006)	(6.03961e-014)	(0)	(0)
12	-16.0000	-25.0000	36.3636	18.8324
	(2.33914e-010)	(0)	(0)	(3.55271e-015)

Gauss–Hermite quadrature

This type of quadrature will be particularly useful when we will consider stochastic processes with gaussian distributions as they approximate integrals of the type

$$\int_{-\infty}^{\infty} F(x) e^{-x^2} \mathrm{d}x$$

such that in this case $w(x) = e^{-x^2}$ and $a = -\infty$, $b = \infty$. The quadrature formula is then given by

$$\int_{-\infty}^{\infty} F(x)e^{-x^2}\mathrm{d}x = \sum_{i=1}^{n} \omega_i F(x_i) + \frac{n!\sqrt{\pi}}{2^n}\frac{F^{(2n)}(\zeta)}{(2n)!}$$

for $\zeta \in (-\infty; \infty)$. In this case, like in the two last particular quadratures, both the nodes and the weights are non trivial to compute. The nodes can be computed using any root finding procedure, and the weights can be computed as explained earlier, noting that $\int_{-\infty}^{\infty} w(x)\mathrm{d}x = \sqrt{\pi}$.

As aforementioned, this type of quadrature is particularly useful when we want to compute the moments of a normal distribution. Let us assume that $x \rightsquigarrow \mathscr{N}(\mu, \sigma^2)$ and that we want to compute

$$\frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{\infty} F(x)e^{-\frac{(x-\mu)^2}{2\sigma^2}}\mathrm{d}x$$

in order to stick to the problem this type of approach can explicitly solve, we need to transform the variable using the linear map

$$y = \frac{x - \mu}{\sigma\sqrt{2}}$$

such that the problem rewrites

$$\frac{1}{\sqrt{\pi}} \int_{-\infty}^{\infty} F(\sigma\sqrt{2}y + \mu)e^{-y^2}\mathrm{d}y$$

and can therefore be approximated by

$$\frac{1}{\sqrt{\pi}} \sum_{i=1}^{n} \omega_i F(\sigma\sqrt{2}y_i + \mu)$$

where y_i and ω_i are the Gauss–Hermite nodes and weights over the interval $(-\infty; \infty)$.

As a first example, let us compute the average of a log–normal distribution, that is $\log(X) \rightsquigarrow \mathscr{N}(\mu, \sigma^2)$ We then know that $E(X) = \exp(\mu + \sigma^2/2)$. This

is particularly important as we will often rely in macroeconomics on shocks that follow a log–normal distribution. Table 4.4 reports the results as well as the approximation error into parenthesis for $\mu = 0$ and different values of σ. Another direct application of this method in economics is related to the

Table 4.4: Gauss–Hermite quadrature

n	0.01	0.1	0.5	1.0	2.0
2	1.00005	1.00500	1.12763	1.54308	3.76219
	(8.33353e-10)	(8.35280e-06)	(0.00552249)	(0.105641)	(3.62686)
4	1.00005	1.00501	1.13315	1.64797	6.99531
	(2.22045e-16)	(5.96634e-12)	(2.46494e-06)	(0.000752311)	(0.393743)
8	1.00005	1.00501	1.13315	1.64872	7.38873
	(2.22045e-16)	(4.44089e-16)	(3.06422e-14)	(2.44652e-09)	(0.00032857)
12	1.00005	1.00501	1.13315	1.64872	7.38906
	(3.55271e-15)	(3.55271e-15)	(4.88498e-15)	(1.35447e-14)	(3.4044e-08)

discretization of shocks that we will face when we will deal with methods for solving rational expectations models. In fact, we will often face shocks that follow Gaussian AR(1) processes

$$x_{t+1} = \rho x_t + (1-\rho)\overline{x} + \varepsilon_{t+1}$$

where $\varepsilon_{t+1} \rightsquigarrow \mathcal{N}(0, \sigma^2)$. This implies that

$$\frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{\infty} \exp\left\{ -\frac{1}{2} \left(\frac{x_{t+1} - \rho x_t - (1-\rho)\overline{x}}{\sigma} \right)^2 \right\} \mathrm{d}x_{t+1} = \int f(x_{t+1}|x_t)\mathrm{d}x_{t+1} = 1$$

which illustrates the fact that x is a continuous random variable. The question we now ask is: *does there exist a discrete representation to x which is equivalent to its continuous representation?* The answer to this question is **yes** as shown in Tauchen and Hussey [1991][2] Tauchen and Hussey propose to replace the integral by

$$\int \Phi(x_{t+1}; x_t, \overline{x}) f(x_{t+1}|\overline{x})\mathrm{d}x_{t+1} \equiv \int \frac{f(x_{t+1}|x_t)}{f(x_{t+1}|\overline{x})} f(x_{t+1}|\overline{x})\mathrm{d}x_{t+1} = 1$$

[2]This is actually a direct application of gaussian quadrature.

where $f(x_{t+1}|\overline{x})$ denotes the density of x_{t+1} conditional on the fact that $x_t = \overline{x}$ (therefore the unconditional density), which in our case implies that

$$\Phi(x_{t+1}; x_t, \overline{x}) \equiv \frac{f(x_{t+1}|x_t)}{f(x_{t+1}|\overline{x})} = \exp\left\{-\frac{1}{2}\left[\left(\frac{x_{t+1} - \rho x_t - (1-\rho)\overline{x}}{\sigma}\right)^2 - \left(\frac{x_{t+1} - \overline{x}}{\sigma}\right)^2\right]\right\}$$

then we can use the standard linear transformation and impose $y_t = (x_t - \overline{x})/(\sigma\sqrt{2})$ to get

$$\frac{1}{\sqrt{\pi}} \int_{-\infty}^{\infty} \exp\left\{-\left((y_{t+1} - \rho y_t)^2 - y_{t+1}^2\right)\right\} \exp\left(-y_{t+1}^2\right) \mathrm{d}y_{t+1}$$

for which we can use a Gauss–Hermite quadrature. Assume then that we have the quadrature nodes y_i and weights ω_i, $i = 1, \ldots, n$, the quadrature leads to the formula

$$\frac{1}{\sqrt{\pi}} \sum_{j=1}^{n} \omega_j \Phi(y_j; y_i; \overline{x}) \simeq 1$$

in other words we might interpret the quantity $\omega_j \Phi(y_j; y_i; \overline{x})$ as an "estimate" $\widehat{\pi}_{ij}$ of the transition probability from state i to state j, but remember that the quadrature is just an approximation such that it will generally be the case that $\sum_{j=1}^{n} \widehat{\pi}_{ij} = 1$ will not hold exactly. Tauchen and Hussey therefore propose the following modification:

$$\widehat{\pi}_{ij} = \frac{\omega_j \Phi(y_j; y_i; \overline{x})}{\sqrt{\pi} s_i}$$

where $s_i = \frac{1}{\sqrt{\pi}} \sum_{j=1}^{n} \omega_j \Phi(y_j; y_i; \overline{x})$. We then end up with a markov chain with nodes $x_i = \sqrt{2}\sigma y_i + \mu$ and transition probability π_{ij} given by the previous equation. The matlab code to generate such an approximation is then straightforward. It yields the following 4 states approximation to an AR(1) process with persistence $\rho = 0.9$ and $\sigma = 0.01$ with $\overline{x} = 0$

$$x^d = \{-0.0233, -0.0074, 0.0074, 0.0233\}$$

and

$$\Pi = \begin{pmatrix} 0.7330 & 0.2557 & 0.0113 & 0.0000 \\ 0.1745 & 0.5964 & 0.2214 & 0.0077 \\ 0.0077 & 0.2214 & 0.5964 & 0.1745 \\ 0.0000 & 0.0113 & 0.2557 & 0.7330 \end{pmatrix}$$

meaning for instance that we stay in state 1 with probability 0.7330, but will transit from state 2 to state 3 with probability 0.2214.

MATLAB CODE: DISCRETIZATION OF AN AR(1)

```
n       = 2;                    % number of nodes
xbar    = 0;                    % mean of the x process
rho     = 0.95;                 % persistence parameter
sigma   = 0.01;                 % volatility

[xx,wx] = gauss_herm(n);        % nodes and weights for x
x_d     = sqrt(2)*s*xx+mx;      % discrete states
x=xx(:,ones(n,1));
y=x';
w=wx(:,ones(n,1))';
%
% computation
%
px  = (exp(y.*y-(y-rx*x).*(y-rx*x)).*w)./sqrt(pi);
sx  = sum(px')';
px  = px./sx(:,ones(n,1));
```

4.2.3 Potential problems

In all the cases we dealt with in the previous sections, the integral were definite or at least existed (up to some examples), but there may exist some singularities in the function such that the integral may not be definite. For instance think of integrating $x^{-\alpha}$ over $[0;1]$, the function diverges in 0. How will perform the methods we presented in the previous section. The following theorem by Davis and Rabinowitz [1984] states that standard method can still be used.

Theorem 3 *Assume that there exists a continuous monotonically increasing function* $G : [0;1] \longrightarrow \mathbb{R}$ *such that* $\int_0^1 G(x)dx < \infty$ *and* $|F(x)| \leqslant |G(x)|$ *on* $[0;1]$*, the the Newton–Cotes rule (with* $F(1) = 0$ *to avoid the singularity in 1) and the Gauss–Legendre quadrature rule converge to* $\int_0^1 F(x)dx$ *as* n *increases to* ∞*.*

Therefore, we can still apply standard methods to compute such integrals, but convergence is much slower and the error formulas cannot be used anymore as

$\|F^{(k)}(x)\|_\infty$ is infinite for $k \geqslant 1$. Then, if we still want to use error bounds, we need to accommodate the rules to handle singularities. There are several ways of dealing with singularities

- develop a specific quadrature method to deal with the singularity
- Use a change of variable

Another potential problem is *how much intervals or nodes should we use?* Usually there is no clear answer to that question, and we therefore have to adapt the method. This is the so–called adaptive quadrature method. The idea is to increase the number of nodes up to the point where increases in the number of nodes do not yield any significant change in the numerical integral. The disadvantage of this approach is the computational cost it involves.

4.2.4 Multivariate integration

There will be situations where we would like to compute multivariate integrals. This will in particular be the case when we will deal with models in which the economic environment is hit by stochastic shocks, or in incentives problems where the principal has to reveal multiple characteristics.... In such a case, numerical integration is on order. There are several ways of obtaining multivariate integration, among which product rules that I will describe the most, non–product rules which are extremely specific to the problem we handle or Monte–Carlo and Quasi Monte–Carlo methods.

Product rules

Let us assume that we want to compute the integral

$$\int_{a_1}^{b_1} \dots \int_{a_s}^{b_s} F(x_1, \dots, x_s) w_1(x_1) \dots w_s(x_s) \mathrm{d}x_1 \dots \mathrm{d}x_s$$

for the function $F : \mathbb{R}^s \longrightarrow \mathbb{R}$ and where w_k is a weighting function. The idea of product rules is just to extend the standard one–dimensional quadrature

approach to higher dimensions by multiplying sums. For instance, let x_i^k and ω_i^k, $k = 1, \ldots, n_k$ be the quadrature nodes and weights of the one dimensional problem along dimension $k \in \{1, \ldots, s\}$, which can be obtained either from a Newton–Cotes formula or a Gaussian quadrature formula. The product rule will approximate the integral by

$$\sum_{i_1=1}^{n_1} \ldots \sum_{i_s=1}^{n_s} \omega_{i_1}^1 \ldots \omega_{i_s}^s F(x_{i_1}^1, \ldots, x_{i_s}^s)$$

A potential difficulty with this approach is that when the dimension of the space increases, the computational cost increases exponentially — this is the so–called "curse of dimensionality". Therefore, this approach should be restricted for low dimensions problems.

As an example of use of this type of method, let us assume that we want to compute the first order moment of the 2 dimensional function $F(x_1, x_2)$, where

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \rightsquigarrow \mathscr{N}\left(\begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix}, \begin{pmatrix} \sigma_{11} & \sigma_{12} \\ \sigma_{12} & \sigma_{22} \end{pmatrix} \right)$$

We therefore have to compute the integral

$$|\Sigma|^{-\frac{1}{2}} (2\pi)^{-1} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} F(x_1, x_2) \exp\left(-\frac{1}{2} (x-\mu)' \Sigma^{-1} (x-\mu) \right) \mathrm{d}x_1 \mathrm{d}x_2$$

where $x = (x_1, x_2)'$, $\mu = (\mu_1, \mu_2)'$, $\Sigma = \begin{pmatrix} \sigma_{11} & \sigma_{12} \\ \sigma_{12} & \sigma_{22} \end{pmatrix}$. Let Φ be the Cholesky decomposition of Σ such that $\Sigma = \Phi\Phi'$, and let us make the change of variable

$$y = \Phi^{-1}(x - \mu)/\sqrt{2} \Longleftrightarrow x = \sqrt{2}\Phi y + \mu$$

then, the integral rewrites

$$\pi^{-1} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} F(\sqrt{2}\Phi y + \mu) \exp\left(-\sum_{i=1}^{s} y_i^2 \right) \mathrm{d}y_1 \mathrm{d}y_2$$

We then use the product rule relying on one–dimensional Gauss–Hermite quadrature, such that we approximate the integral by

$$\frac{1}{\pi} \sum_{i_1=1}^{n_1} \sum_{i_2=1}^{n_2} \omega_{i_1}^1 \omega_{i_2}^2 F(\sqrt{2}\varphi_{11} y_1 + \mu_1, \sqrt{2}(\varphi_{21} y_1 + \varphi_{22} y_2) + \mu_2)$$

As an example (see the matlab code) we set

$$F(x_1, x_2) = (e^{x_1} - e^{\mu_1})\,(e^{x_2} - e^{\mu_2})$$

with $\mu = (0.1, 0.2)'$ and

$$\Sigma = \begin{pmatrix} 0.0100 & 0.0075 \\ 0.0075 & 0.0200 \end{pmatrix}$$

The results are reported in table 4.5, where we consider different values for n_1 and n_2. It appears that the method performs well pretty fast, as the true value for the integral is 0.01038358129717, which is attained for $n_1 \geqslant 8$ and $n_2 \geqslant 8$.

Table 4.5: 2D Gauss-Hermite quadrature

$n_x \backslash n_y$	2	4	8	12
2	0.01029112845254	0.01029142086814	0.01029142086857	0.01029142086857
4	0.01038328639869	0.01038358058862	0.01038358058906	0.01038358058906
8	0.01038328710679	0.01038358129674	0.01038358129717	0.01038358129717
12	0.01038328710679	0.01038358129674	0.01038358129717	0.01038358129717

MATLAB CODE: 2D GAUSS–HERMITE QUADRATURE (PRODUCT RULE)

```
n       = 2;                    % dimension of the problem
n1      = 8;                    % # of nodes for x1
[x1,w1] = gauss_herm(n1);       % nodes and weights for x1
n2      = 8;                    % # of nodes for x2
[x2,w2] = gauss_herm(n2);       % nodes and weights for x2

Sigma   = 0.01*[1 0.75;0.75 2];
Omega   = chol(Sigma)';
mu1     = 0.1;
mu2     = 0.2;

int=0;
for i=1:n1;
   for j=1:n2;
      x12   = sqrt(2)*Omega*[x1(i);x2(j)]+[mu1;mu2];
      f     = (exp(x12(1))-exp(mu1))*(exp(x12(2))-exp(mu2));
```

```
      int   = int+w1(i)*w2(j)*f
   end
end
int=int/sqrt(pi^n);
```

The problem is that whenever the dimension of the problem increases or as the function becomes complicated these procedures will not perform well, and relying on stochastic approximation may be a good idea.

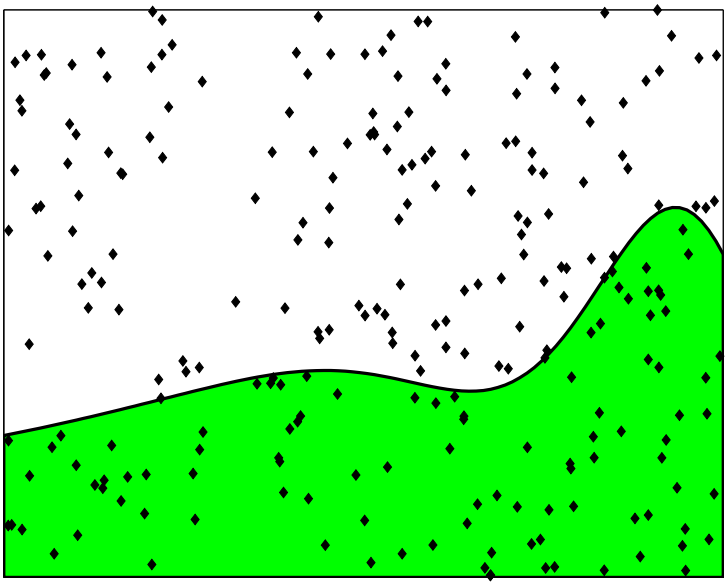
4.2.5 Monte–Carlo integration

Monte–Carlo integration methods are sampling methods that are based on probability theory, and rely on several trials to reveal information. From an intuitive point of view, Monte carlo methods rest on the central limit theorem and the law of large number and are capable of handling quite complicated and large problems. These two features make Monte–Carlo method particularly worth learning.

A very important feature of Monte–Carlo methods is that they appeal to probability theory, therefore any result of a Monte–Carlo experiment is a random variable. This is precisely a very nice feature of Monte–Carlo methods as by their probabilistic nature, they put a lot of structure on the error of approximation which has a probabilistic distribution. Finally, by adjusting the size of the sample we can always increase the accuracy of the approximation. This is just a consequence of the central limit theorem.

The basic intuition that lies behind Monte–Carlo integration may be found in figure 4.3. The dark curve is the univariate function we want to integrate and the shaded area under this curve is the integral. Then the evaluation of an integral using Monte–Carlo simulations amounts to draw random numbers in the x–y plan (the dots in the graph), then the integral of the function f is approximately given by the total area times the fraction of points that fall under the curve $f(x)$. It is then obvious that the greater the number of points — the more information we get — the more accurate is the evaluation of this area. Further, this method will prove competitive only for complicated

Figure 4.3: Basic idea of Monte–Carlo integration



and/or multidimensional functions. Note that the integral evaluation will be better if the points are uniformly scattered in the entire area — that is if the information is spread all over the area.

Another way to think of it is just to realize that

$$\int_a^b f(x)\mathrm{d}x = (b-a)E_{\mathscr{U}_{[a;b]}}(f(x))$$

such that if we draw n random numbers, x_i, $i = 1, \ldots, n$, from a $\mathscr{U}_{[b;a]}$, an approximation of the integral of $f(x)$ over the interval $[a; b]$ is given by

$$\frac{(b-a)}{n}\sum_{i=1}^{n} f(x_i)$$

The key point here is the way we get random numbers.

Not so random numbers!

Monte–Carlo methods are usually associated to stochastic simulations and therefore rely on random numbers. But such numbers cannot be generated

by computers.[3] Computers are only — and this is already a great thing — capable of generating *pseudo–random numbers* — that is numbers that look like random numbers because they look unpredictable. However it should be clear to you that all these numbers are just generated with deterministic algorithms — explaining the term *pseudo* — whose implementation is said to be of the *volatile type* in the sense that the *seed* — the initial value of a sequence depends on an external unpredictable feeder like the computer clock. Two important properties are usually demanded to such generators:

1. zero serial correlation: we want iid sequences.

2. correct frequency of runs: we do not want to generate predictable sequences

The most well–known and the simplest random number generator relies on the so–called *linear congruential method* which obeys the equation

$$x_{k+1} = ax_k + c \,(\text{mod } m)$$

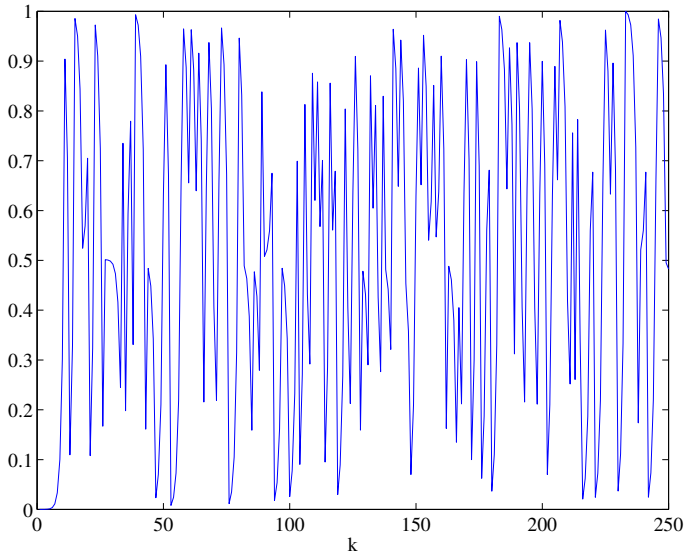
One big advantage of this method is that it is pretty fast and cheap. The most popular implementation of this scheme assumes that $a = \pm 3(\text{mod}8)$, $c = 0$ and $m = 2^b$ where b is the number of significant bits available on the computer (these days 32 or 64). Using this scheme we then generates sequences that ressemble random numbers.[4] For example figure 4.4 reports a sequence of 250 random numbers generated by this pseudo random numbers generator, and as can be seen it looks like random numbers, it smells randomness, it tastes randomness but this is not randomness! In fact, linear congruential methods are not immune from serial correlation on successive calls: if k random numbers generators at a time are used to plot points in k dimensional space, then the points will not fill up the k–dimensional space but they will tend to lie on

[3] There have been attempts to build truly random number generators, but these technics were far too costly and awkward.

[4] Generating a 2 dimensional sequence may be done extracting sub–sequences: $y_k = (x_{2k+1}, x_{2k+2})$.

$(k-1)$–dimensional planes. This can easily be seen as soon as we plot x_{k+1} against x_k, as done in figure 4.5. This too pronounced non–random pattern

Figure 4.4: A pseudo random numbers draw (linear congruential generator)

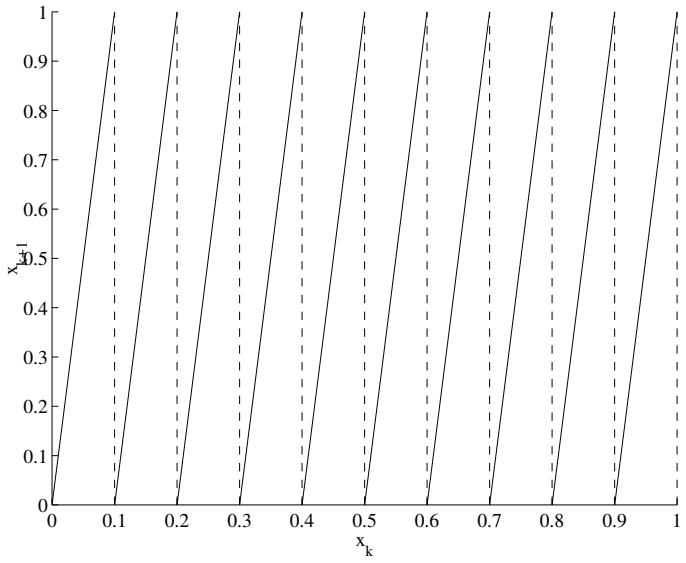


for these numbers led to push linear congruential methods into disfavor, the solution has been to design more complicated generators. An example of those generators quoted by Judd [1998] is the multiple prime random number generator for which we report the matlab code. This pseudo random numbers generator proposed by Haas [1987] generates integers between 0 and 99999, such that dividing the sequence by 100,000 returns numbers that approximate a uniform random variable over [0;1] with 5 digits precision. If higher precision is needed, the sequence may just be concatenated using the scheme (for 8 digits precision) $100,000 \times x_{2k} + x_{2k+1}$. The advantage of this generator is that its period is over 85 trillions!

MATLAB CODE: PRIME RANDOM NUMBER GENERATOR

```
long = 10000;          % length of the sample
m    = 971;
ia   = 11113;
```

Figure 4.5: The linear congruential generator




```
ib   = 104322;
x    = zeros(long,1);
x(1) = 481;
for i= 2:long;
   m = m+7;
   ia= ia+1907;
   ib= ib+73939;
   if m>=9973;m=m-9871;end
   if ia>=99991;ia=ia-89989;end
   if ib>=224729;ib=ib-96233;end
   x(i)=mod(x(i-1)*m+ia+ib,100000)/10;
end
```

Over generators may be designed and can be non–linear as

$$x_{k+1} = f(x_k) \text{mod } m$$

or may take rather strange formulations as the one reported by Judd [1998], that begins with a sequence of 55 odd numbers and computes

$$x_k = (x_{k-24} x_{k-55}) \text{mod } 2^{32}$$

which has a period length of 10^25, such that it passes a lot of randomness tests.

A key feature of all these random number generators is that they attempt to draw numbers from a uniform distribution over the interval [0;1]. There however may be some cases where we would like to draw numbers from another distribution — mainly the normal distribution. The way to handle this problem is then to invert the cumulative density function of the distribution we want to generate to get a random draw from this particular distribution. More formally, assume we want numbers generated from the distribution $F(.)$, and we have a draw $\{x_i\}_{i=1}^N$ from the uniform distribution, then the draw $\{y_i\}_{i=1}^N$ from the f distribution may be obtained solving

$$\int_a^{y_i} F(s)\mathrm{d}s = x_i \text{ for } i = 1, \ldots, N$$

Inverting this function may be trivial in some cases (say the uniform over [a;b]) but it may require approximation as in the case of a normal distribution.

Monte–Carlo integration

The underlying idea of Monte–Carlo integration may be found in the *Law of Large Numbers*

Theorem 4 (Law of Large Numbers) *If X_i is a collection of i.i.d. random variables with density $\mu(x)$ then*

$$\lim_{N\to\infty} \frac{1}{N}\sum_{i=1}^{N} X_i = \int x\mu(x)dx \text{ almost surely.}$$

Further, we know that in this case

$$\mathrm{var}\left(\frac{1}{N}\sum_{i=1}^{N} X_i\right) = \frac{\sigma^2}{N} \text{ where } \sigma^2 = \mathrm{var}(X_i)$$

If σ^2 is not known it can be estimated by

$$\widehat{\sigma}^2 = \frac{1}{N-1}\sum_{i=1}^{N}\left(X_i - \overline{X}\right)^2 \text{ with } \overline{X} = \frac{1}{N}\sum_{i=1}^{N} X_i$$

With this in mind, we understand the potential of Monte–Carlo methods for numerical integration. Integrating a function $F(x)$ over [0;1] is nothing else than computing the mean of $F(x)$ assuming that $x \rightsquigarrow \mathscr{U}_{[0;1]}$ therefore a crude application of Monte–Carlo method to compute the integral $\int_0^1 F(x)\mathrm{d}x$ is to draw N numbers, x_i, from a $\mathscr{U}_{[0;1]}$ distribution and take

$$\widehat{I}_F = \frac{1}{N}\sum_{i=1}^{N} F(x_i)$$

as an approximation to the integral. Further, as this is just an estimate of the integral, it is a random variable that has variance

$$\sigma^2_{\widehat{I}_F} = \frac{1}{N}\int_0^1 (F(x) - I_F)^2\,\mathrm{d}x = \frac{\sigma^2_F}{N}$$

where σ^2_F may be estimated by

$$\widehat{\sigma}^2_F = \frac{1}{N-1}\sum_{i=1}^{N}\left(F(x_i) - \widehat{I}_F\right)^2$$

such that the standard error of the Monte–Carlo integral is $\sigma_{\widehat{I}_f} = \widehat{\sigma}_f/\sqrt{N}$.

As an example of a crude application of Monte–Carlo integration, we report in table 4.6 the results obtained integrating the exponential function over [0;1]. This table illustrates why Monte–Carlo integration is seldom used *(i)*

Table 4.6: Crude Monte–Carlo example: $\int_0^1 e^x\mathrm{d}x$.

N	$\widehat{I}_f$	$\widehat{\sigma}_{\widehat{I}_f}$
10	1.54903750	0.13529216
100	1.69945455	0.05408852
1000	1.72543465	0.01625793
10000	1.72454262	0.00494992
100000	1.72139292	0.00156246
1000000	1.71853252	0.00049203

True value: 1.71828182

for univariate integration and *(ii)* without modification. Indeed, as can be

seen a huge number of data points is needed to achieve, on average, a good enough approximation as 1000000 points are needed to get an error lower than `0.5e-4` and the standard deviation associated to each experiment is far too high as even with only 10 data points a Student test would lead us to accept the approximation despite its evident lack of accuracy! Therefore several modifications are usually proposed in order to circumvent these drawbacks.

- *Antithetic variates*: This acceleration method lies on the idea that if f is monotonically increasing then $f(x)$ and $f(1-x)$ are negatively correlated. Then estimating the integral as

$$\widehat{I}_f^A = \frac{1}{2N}\sum_{i=1}^{N}(F(x_i) + F(1-x_i))$$

  will still furnish an unbiased estimator of the integral while delivering a lower variance of the estimator because of the negative correlation between $F(x)$ and $F(1-x)$:

$$\begin{aligned}\text{var}(\widehat{I}_f^A) &= \frac{\text{var}(F(x)) + \text{var}(F(1-x)) + 2\,\text{cov}(F(x), F(1-x))}{4N} \\ &= \frac{\sigma_F^2 + \text{cov}(F(x), F(1-x))}{2N} \leqslant \frac{\sigma_F^2}{N}\end{aligned}$$

  This method is particularly recommended when F is monotone. Table 4.7 illustrates the potential of the approach for the previous example. As can be seen the gains in terms of volatility are particularly important but these are also important in terms of average even in small sample.[5]

- *Stratified sampling:* Stratified sampling rests on the basic and quite appealing idea that the variance of f over a subinterval of [0;1] should be lower than the variance over the whole interval. The underlying idea is to prevent draws from clustering in a particular region of the interval, and therefore we force the procedure to visit each sub–interval, and by this we enlarge the information set used by the algorithm.

[5]Note that we used the same seed when generating this integral and the one we generate using crude Monte–Carlo.

Table 4.7: Antithetic variates example: $\int_0^1 e^x \mathrm{d}x$.

N	$\widehat{I}_f$	$\widehat{\sigma}_{\widehat{I}_f}$
10	1.71170096	0.02061231
100	1.73211884	0.00908890
1000	1.72472178	0.00282691
10000	1.71917393	0.00088709
100000	1.71874441	0.00027981
1000000	1.71827383	0.00008845

True value: 1.71828182

The stratified sampling approach works as follows. We set $\lambda \in (0,1)$ and we draw $N_a = \lambda N$ data points over $[0;\lambda]$ and $N_b = N - N_a = (1-\lambda)N$ over $[\lambda;1]$. Then the integral can be evaluated by

$$\widehat{I}_f^s = \frac{1}{N_a}\sum_{i=1}^{N_a} F(x_i^a) + \frac{1}{N_b}\sum_{i=1}^{N_b} F(x_i^b)$$

where $x_i^a \in [0;\lambda]$ and $x_i^b \in [\lambda;1]$. Then the variance of this estimator is given by

$$\frac{\lambda^2}{N_a}\mathrm{var}_a(F(x)) + \frac{(1-\lambda)^2}{N_b}\mathrm{var}_b(F(x))$$

which equals

$$\frac{\lambda}{N}\mathrm{var}_a(F(x)) + \frac{(1-\lambda)}{N}\mathrm{var}_b(F(x))$$

Table 4.8 reports results for the exponential function for $\lambda = 0.25$. As can be seen from the table, up to the 10 points example,[6] there is hopefully no differences between the crude Monte–Carlo method and the stratified sampling approach in the evaluation of the integral and we find potential gain in the use of this approach in the variance of the estimates. The potential problem that remains to be fixed is *How should λ be selected?* In fact we would like to select λ such that we minimize the volatility,

[6]This is related to the very small sample in this case.

Table 4.8: Stratified sampling example: $\int_0^1 e^x \mathrm{d}x$.

N	$\widehat{I}_f$	$\widehat{\sigma}_{\widehat{I}_f}$
10	1.52182534	0.11224567
100	1.69945455	0.04137204
1000	1.72543465	0.01187637
10000	1.72454262	0.00359030
100000	1.72139292	0.00114040

True value: 1.71828182

which amounts to set λ such that

$$\mathrm{var}_a(F(x)) = \mathrm{var}_b(F(x))$$

which drives the overall variance to

$$\frac{\mathrm{var}_b(F(x))}{N}$$

- *Control variates*: The method of control variates tries to extract information from a function that approximates the function to be integrated arbitrarily well, while begin easy to integrate. Hence, assume there exists a function φ that is similar to F, but that can be easily integrated, the identity

$$\int F(x)\mathrm{d}x = \int (F(x) - \varphi(x))\mathrm{d}x + \int \varphi(x)\mathrm{d}x$$

restates the problem as the Monte–Carlo integration of $(F - \varphi)$ plus the known integral of φ. The variance of $(F - \varphi)$ is given by $\sigma_F^2 + \sigma_\varphi^2 - 2\mathrm{cov}(F, \varphi)$ which is lower than the variance of σ_F^2 provided the covariance between F and φ is high enough.

In our example, we may use as the φ function: $1+x$ since $\exp(x) \simeq 1+x$ in a neighborhood of zero. $\int_0^1 (1+x)\mathrm{d}x$ is simple to compute and equal to 1.5. Table 4.9 reports the results. As can be seen the method performs a little worse than the antithetic variates, but far better than the crude Monte–Carlo.

Table 4.9: Control variates example: $\int_0^1 e^x \mathrm{d}x$.

N	$\widehat{I}_f$	$\widehat{\sigma}_{\widehat{I}_f}$
10	1.64503465	0.05006855
100	1.71897083	0.02293349
1000	1.72499149	0.00688639
10000	1.72132486	0.00210111
100000	1.71983807	0.00066429
1000000	1.71838279	0.00020900

True value: 1.71828182

- *Importance sampling:* Importance sampling attempts to circumvent a insufficiency of crude Monte–Carlo method: by drawing numbers from a uniform distribution, information is spread all over the interval we are sampling over. But there are some cases where this is not the most efficient strategy. Further, it may exist a simple transformation of the problem for which Monte–Carlo integration can be improved to generate a far better result in terms of variance. More formally, assume you want to integrate F over a given interval

$$\int_{\mathscr{D}} F(x)\mathrm{d}x$$

now assume there exists a function G such that $H = F/G$ is almost constant over the domain of integration $\mathscr{D}$, the problem may be restated

$$\int_{\mathscr{D}} \frac{F(x)}{G(x)} G(x)\mathrm{d}x \equiv \int_{\mathscr{D}} H(x)G(x)\mathrm{d}x$$

Then we can easily integrate F by instead sampling H, but not by drawing numbers from a uniform density function but rather from a non uniform density $G(x)\mathrm{d}x$. Then the approximated integral is given by

$$\widehat{I}_F^{is} = \frac{1}{N}\sum_{i=1}^{N} \frac{F(x_i)}{G(x_i)}$$

and it has variance

$$\begin{aligned}\sigma^2_{\widehat{I}^{is}_F} &= \frac{\sigma^2_h}{N} = \frac{1}{N}\left(\int_{\mathscr{D}} \frac{F(x)^2}{G(x)^2} G(x)\mathrm{d}x - \left(\int_{\mathscr{D}} \frac{F(x)}{G(x)} G(x)\mathrm{d}x\right)^2\right) \\ &= \frac{1}{N}\left(\int_{\mathscr{D}} \frac{F(x)^2}{G(x)}\mathrm{d}x - \left(\int_{\mathscr{D}} F(x)\mathrm{d}x\right)^2\right)\end{aligned}$$

The problem we still have is *how should G be selected?* In fact, we see from the variance that if G were exactly F the variance would reduce to zero, but then what would be the gain? and it may be the case that G would not be a distribution or may be far too complicated to sample. In fact we would like to have G to display a shape close to that of F while being simple to sample.

In the example reported in table 4.10, we used $G(x) = (1+\alpha)x^\alpha$, with $\alpha = 1.5$. As can be seen the gains in terms of variance are particularly important, which render the method particularly attractive, nevertheless the selection of the G function requires a pretty good knowledge of the function to be integrated, which will not be the case in a number of economic problems.

Table 4.10: Importance sampling example: $\int_0^1 e^x \mathrm{d}x$.

N	$\widehat{I}_f$	$\widehat{\sigma}_{\widehat{I}_f}$
10	1.54903750	0.04278314
100	1.69945455	0.00540885
1000	1.72543465	0.00051412
10000	1.72454262	0.00004950
100000	1.72139292	0.00000494
1000000	1.71853252	0.00000049

True value: 1.71828182

4.2.6 Quasi–Monte Carlo methods

Quasi–Monte Carlo methods are fundamentally different from Monte–Carlo methods although they look very similar. Indeed, in contrast to Monte–Carlo methods that relied on probability theory, quasi–Monte Carlo methods rely on number theory (and Fourier analysis, but we will not explore this avenue here). In fact, as we have seen, Monte–Carlo methods use pseudo–random numbers generators, that are actually deterministic schemes. A first question that may then be addressed to such an approach is: *If the MC sequences are deterministic, how can I use probability theory to get theoretical results?* and in particular *What is the applicability of the Law of Large Numbers and the Central Limit Theorem?* This is however a bit unfair as many new random number generators pass the randomness tests. Nevertheless, why not acknowledging the deterministic nature of these sequences and try to use them? This is what is proposed by Quasi–Monte Carlo methods.

There is another nice feature of Quasi–Monte Carlo methods, which is related to the rate of convergence of the method. Indeed, we have seen that choosing N points uniformly in an n–dimensional space leads to an error in Monte–Carlo that diminishes as $1/\sqrt{N}$. From an intuitive point of view, this comes from the fact that each new point adds linearly to an accumulated sum that will become the function average, and also linearly to an accumulated sum of squares that will become the variance. Since the estimated error is the square root of the variance, the power is $N^{-1/2}$. But we can accelerate the convergence relying on some purely deterministic schemes, as quasi–Monte Carlo methods do.

Quasi–Monte Carlo methods rely on equi–distributed sequences, that is sequence that satisfy the following definition.

Definition 1 *A sequence* $\{x_i\}_{i=1}^{\infty} \subset \mathscr{D} \subset \mathbb{R}^n$ *is said to be equi–distributed*

over the domain $\mathscr{D}$ iff

$$\lim_{N\to\infty} \frac{\mu(\mathscr{D})}{N} \sum_{i=1}^{N} F(x_i) = \int_{\mathscr{D}} F(x)dx$$

for all Rieman–integrable function $F : \mathbb{R}^n \longrightarrow \mathbb{R}$, where $\mu(\mathscr{D})$ is the Lebesgue measure of $\mathscr{D}$.

In order to better understand what it exactly means, let us consider the uni–dimensional case the sequence $\{x_i\}_{i=1}^{\infty} \subset \mathbb{R}$ is equidistributed if for any Riemann–integrable function we have

$$\lim_{N\to\infty} \frac{(b-a)}{N} \sum_{i=1}^{N} F(x_i) = \int_a^b F(x)\mathrm{d}x$$

This is therefore just a formal statement of a uniform distribution, as it just states that if we sample "correctly" data points over the interval $[a; b]$ then these points should deliver a valid approximation to the integration problem. From an intuitive point of view, equi–distributed sequences are just deterministic sequences that mimic the uniform distribution, but since they are, by essence, deterministic, we can select their exact location and therefore we can avoid clustering or sampling twice the same point. This is why Quasi–Monte Carlo methods appear to be so attractive: they should be more efficient.

There exist different ways of selecting equi–distributed sequences. Judd [1998], chapter 9, reports different sequences that may be used, but they share the common feature of being generated by the scheme

$$x_{k+1} = (x_k + \theta) \bmod 1$$

which amounts to take the fractional part of $k\theta$.[7] θ should be an irrational number. These sequences are among others

[7]Remember that the fractional part is that part of a number that lies right after the dot. This is denoted by $\{.\}$, such that $\{2.5\} = 0.5$. This can be computed as

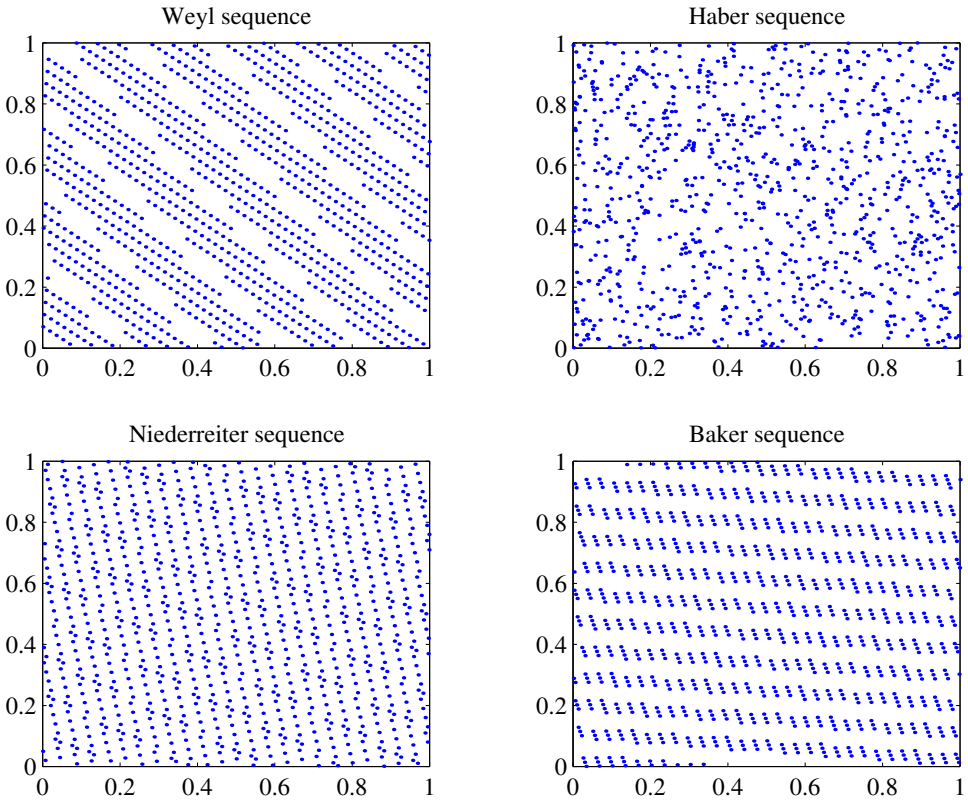
$$\{x\} = x - \max\{k \in \mathbb{Z} | k \leqslant x\}$$

The matlab function that return this component is `x-fix(x)`.

- Weyl: $(\{k\sqrt{p_1}\},\ldots,\{k\sqrt{p_n}\})$, where n is the dimension of the space.
- Haber: $\left(\{\frac{k(k+1)}{2}\sqrt{p_1}\},\ldots,\{\frac{k(k+1)}{2}\sqrt{p_n}\}\right)$
- Niederreiter: $\left(\{k\,2^{1/(1+n)}\},\ldots,\{k\,2^{n/(1+n)}\}\right)$
- Baker: $(\{k\,e^{r_1}\},\ldots,\{ke^{r_n}\})$, rs are rational and distinct numbers

In all these cases, the ps are usually prime numbers. Figure 4.6 reports a 2–dimensional sample of 1000 points for each type of sequence. There obviously

Figure 4.6: Quasi–Monte Carlo sequences



exist other ways of obtaining sequences for quasi–Monte carlo methods that rely on low discrepancy approaches, Fourier methods, or the so–called good lattice points approach. The interested reader may refer to chapter 9 in Judd

[1998], but we will not investigate this any further as this would bring us far away from our initial purpose.

MATLAB CODE: EQUIDISTRIBUTED SEQUENCES

```
n=2;                    % dimension of the space
nb=1000;                % number of data points
K=[1:nb]';              % k=1,...,nb
seq='NIEDERREITER';     % Type of sequence
switch upper(seq)
  case 'WEYL'                   % Weyl
    p=sqrt(primes(n+1));
    x=K*p;
    x=x-fix(x);
  case 'HABER'                  % Haber
    p=sqrt(primes(n+1));
    x=(K.*(K+1)./2)*p;
    x=x-fix(x);
  case 'NIEDERREITER'           % Niederreiter
    x=K*(2.^((1:n)/(1+n)));
    x=x-fix(x);
  case 'BAKER'                  % Baker
    x=K*exp(1./primes(n+1));
    x=x-fix(x);
  otherwise
     error('Unknown sequence requested')
end
```

As an example, we report in table 4.11 the results obtained integrating the exponential function over [0;1]. Once again the potential gain of this type of method will be found in approximating integral of multi–dimensional or complicated functions. Further, as for Monte–Carlo methods, this type of integration is not restricted to the $[0;1]^n$ hypercube. You may transform the function, or perform a change of variables to be able to use the method. Finally note, that we may apply all the acceleration methods applied to Monte–Carlo technics to the quasi–Monte Carlo approach too.

Table 4.11: Quasi Monte–Carlo example: $\int_0^1 e^x \mathrm{d}x$.

N	Weyl	Haber	Niederreiter	Baker
10	1.67548650	1.72014839	1.67548650	1.82322097
	(0.0427953)	(0.00186656)	(0.0427953)	(0.104939)
100	1.71386433	1.75678423	1.71386433	1.71871676
	(0.0044175)	(0.0385024)	(0.0044175)	(0.000434929)
1000	1.71803058	1.71480932	1.71803058	1.71817437
	(0.000251247)	(0.00347251)	(0.000251247)	(0.000107457)
10000	1.71830854	1.71495774	1.71830854	1.71829897
	(2.67146e-005)	(0.00332409)	(2.67146e-005)	(1.71431e-005)
100000	1.71829045	1.71890493	1.71829045	1.71827363
	(8.62217e-006)	(0.000623101)	(8.62217e-006)	(8.20223e-006)
1000000	1.71828227	1.71816697	1.71828227	1.71828124
	(4.36844e-007)	(0.000114855)	(4.36844e-007)	(5.9314e-007)

True value: 1.71828182, absolute error into parenthesis.

Bibliography

Davis, P.J. and P. Rabinowitz, *Methods of Numerical Integration*, New York: Academic Press, 1984.

Judd, K.L., *Numerical methods in economics*, Cambridge, Massachussets: MIT Press, 1998.

Tauchen, G. and R. Hussey, Quadrature Based Methods for Obtaining Approximate Solutions to Nonlinear Asset Pricing Models, *Econometrica*, 1991, *59* (2), 371–396.

Index

Contents

List of Figures

List of Tables