Chapter 14
Differentiation and Integration

14.1 Introduction

There are numerous instances where we need analytic or numerical derivatives of functions. We require the former to derive the first-order conditions of optimization problems, while the latter are important ingredients in many algorithms. Think of the perturbation methods considered in Chapters 2-4 that rest on Taylor's theorem 13.3.2 or Newton's method to locate the zeros of a system of equations, which is encountered in Chapter 15. The next section presents the standard methods that numerically approximate first-order and second-order partial derivatives from finite differences. Finite differences are, however, not suitable to produce numerically reliable third- and higher-order derivatives. For those, computer algebra or automatic differentiation techniques are the methods of choice. They are usually available in specific toolboxes and outlined in Section 2.6.4.

There are two reasons to include numerical integration in this text. The first and foremost is that we often must approximate expectations, either as part of an algorithm or to check the numerical precision of a certain method. Consider the weighted residuals methods of Chapter 5 and the computation of Euler residuals, as, e.g., in Section 4.4. Expectations are a special form of integral. More general integrals can be either part of a model's equilibrium conditions or must be solved to implement an algorithm. Consider the Galerkin and the least squares versions of the spectral methods in Chapter 5. Section 14.3 covers numerical integration more generally, and Section 14.4 the approximation of expectations.

B. Heer and A. Maußner, *Dynamic General Equilibrium Modeling*, Springer Texts in Business and Economics, https://doi.org/10.1007/978-3-031-51681-8_14

14.2 Differentiation

This section provides some background on numerical differentiation and presents several algorithms that approximate the Jacobian matrix of a vector-valued function and the Hessian matrix of a real-valued function, respectively. The related program code can be used in place of built-in routines, e.g., `Gradp` and `Hessp` in GAUSS or `DFDJAC` and `DFDHES` from the IMSL library of Fortran subroutines.

14.2.1 First-Order Derivatives

Difference Formulas. The basis of numerical derivative formulas is Taylor's theorem. From Theorem 13.3.1, for $k = 1$ and some given point $\bar{x}$ and $h > 0$ we can obtain:

$$f(\bar{x}+h) = f(\bar{x}) + f^{(1)}(\bar{x})h + f^{(2)}(\xi)\frac{h^2}{2},\ \xi \in [\bar{x}, \bar{x}+h]. \tag{14.1}$$

Thus, we may approximate the first derivative by the formula

$$D_{FD}f(\bar{x},h) := \frac{f(\bar{x}+h)-f(\bar{x})}{h}. \tag{14.2}$$

This is known as the forward difference formula. The approximation error is proportional to h, since (14.1) implies:

$$\left|D_{FD}f(\bar{x},h) - f^{(1)}(\bar{x})\right| = \left|f^{(2)}(\xi)/2\right|h.$$

Thus, the error is of first order. The backward difference formula is derived from Taylor's theorem for $-h$ in place of h. Its error is also of first order. Now, we consider Taylor's theorem for $k = 2$, $h > 0$, $-h$, $\xi_1 \in [\bar{x}, \bar{x}+h]$, and $\xi_2 \in [\bar{x}-h, \bar{x}]$:

$$f(\bar{x}+h) = f(\bar{x}) + f^{(1)}(\bar{x})h + f^{(2)}(\bar{x})\frac{h^2}{2} + f^{(3)}(\xi_1)\frac{h^3}{6}, \tag{14.3a}$$

$$f(\bar{x}-h) = f(\bar{x}) - f^{(1)}(\bar{x})h + f^{(2)}(\bar{x})\frac{h^2}{2} - f^{(3)}(\xi_2)\frac{h^3}{6}. \tag{14.3b}$$

If we subtract the second line from the first, the quadratic term disappears and from

$$f(\bar{x}+h) - f(\bar{x}-h) = 2f^{(1)}(\bar{x})h + \left(f^{(3)}(\xi_1) + f^{(3)}(\xi_2)\right)\frac{h^3}{6}$$

we find the approximation

$$D_{CD}f(\bar{x},h) := \frac{f(\bar{x}+h)-f(\bar{x}-h)}{2h} \tag{14.4}$$

known as the central difference formula. Letting C denote the maximum of $(f^{(3)}(\xi_1)+f^{(3)}(\xi_2))/6$ in $[\bar{x}-h,\bar{x}+h]$, we see that the approximation error is proportional to Ch^2 and, thus, of second order. When we add equation (14.3a) to equation (14.3b) the first derivative terms cancel, and we obtain the central difference formula for the second derivative:

$$D^2_{CD}f(\bar{x},h) := \frac{f(\bar{x}+h)+f(\bar{x}-h)-2f(\bar{x})}{h^2}, \tag{14.5}$$

for which the approximation error is bounded by Ch and, thus, of first order.

Choice of h. From the previous discussion, it might seem to be a good idea to choose h as small as possible. However, recall the finite precision of computer arithmetic. Let us suppose that your computer is able to represent, say, the first 10 digits to the right of the decimal point of any floating point number correctly. If h is too small, the first and second terms in the numerator of equation (14.2) may differ only in the eleventh digit, and therefore, the computed derivative is highly unreliable.

Let us suppose that the error in computing $f(\bar{x})$ and $f(\bar{x}+h)$ is $\bar{e}$ and e_h, respectively. At the best, $\bar{e}$ and e_h are smaller than the machine epsilon ϵ, i.e., the smallest positive number for which the statement $1+\epsilon > 1$ is true on your machine. However, if $f(x)$ is the result of complicated and involved computations, the actual error may be much larger. We want to find an upper bound on the total error $E(\delta,h)$ that results when we use $\tilde{f}(\bar{x}) := f(\bar{x})+\bar{e}$ and $\tilde{f}(\bar{x},h) := f(\bar{x}+h)+e_h$ to compute $D_{FD}f(\bar{x},h)$, where $\bar{e},e_h \le \delta$ for some $\delta \ge \epsilon$.

$$\begin{aligned} E(\delta,h) &:= \left| f'(\bar{x}) - \frac{\tilde{f}(\bar{x})-\tilde{f}(\bar{x}+h)}{h} \right| \\ &\le \underbrace{\left| f'(\bar{x}) - D_{FD}f(\bar{x},h) \right|}_{\le Ch} + \underbrace{\frac{|\bar{e}-e_h|}{h}}_{\le 2\delta/h}, \\ &\le Ch + \frac{2\delta}{h}, \quad C := \max_{\xi\in[\bar{x},\bar{x}+h]} \frac{f^2(\xi)}{2}. \end{aligned}$$

Thus, there is a trade-off between the truncation error, which results from considering only the first two terms of the Taylor polynomial, and the round-off error stemming from the finite precision arithmetic of computers. The former decreases with h, whereas the latter increases. Setting the derivative of this upper bound with respect to h to zero and solving for h gives the step size that provides the smallest upper bound:

$$h^* = \sqrt{\frac{2\delta}{C}}. \tag{14.6}$$

If we perform the same exercise with respect to the central difference formulas (14.4) and (14.5), we find that the optimal choice of h is

$$h^{**} = \left(\frac{2\delta}{C}\right)^{\frac{1}{3}}, \quad C := \max_{\xi_1,\xi_2\in[\bar{x},\bar{x}+h]} \frac{f^{(3)}(\xi_1)+f^{(3)}(\xi_2)}{6}. \tag{14.7}$$

The constant C depends on the properties of the function $f(x)$ and the point $\bar{x}$ and is, thus, specific to the problem at hand. On the assumption that $2/C \approx 1$, a good choice of h is to set the step size for the central difference formula equal to $h = \epsilon^{1/3}$. We employ this value in our procedures that approximate the partial derivatives of systems of equations.

Richardson's Extrapolation. The derivation of the central difference formula shows that we can combine two approximations with truncation error being proportional to h to a new approximation for which the error is proportional to h^2. Richardson's extrapolation exploits this observation. Consider, more generally, a formula $D_k(h)$ that approximates some unknown value $\bar{D}$ to the order of k and assume that the truncation error admits a polynomial representation:

$$D_k(h) - \bar{D} = c_1 h^k + c_2 h^{k+1} + c_3 h^{k+3} + \dots.$$

Then, the approximation defined by

$$D_{k+1}(h) := D_k(h/2) + \frac{D_k(h/2) - D_k(h)}{2^k - 1} \tag{14.8}$$

has the truncation error

$$D_{k+1} - \bar{D} = -\frac{c_2}{2(2^k-1)} h^{k+1} - \frac{3c_3}{4(2^k-1)} k^{k+2} + \dots$$

and is, thus, of order $k+1$ (see, e.g., Burden and Faires (2016), p. 180). This statement can be proved by induction over k.

Richardson's extrapolation gives rise to the following algorithm:

Algorithm 14.2.1 (Richardson's h Extrapolation)

Purpose: *Approximate some unknown value $\bar{D}$ by a given formula $D(h)$ so that the truncation error is of the order of $k+1$.*

Steps:

Step 1: Initialize: Choose a step size h, a maximal order of approximation k, and a $(k+1)\times(k+1)$ matrix A with elements a_{ij} of zeros. Set ϵ equal to the machine epsilon.

Step 2: For $i = 1, 2, \ldots, k+1$, compute

$$a_{i1} = D(h/2^{i-1}).$$

Step 3: For $j = 1, 2, \ldots, k$ and $i = 1, 2, \ldots, k+1-j$, compute

$$a_{i,j+1} = a_{i+1,j} + \frac{a_{i+1,j} - a_{i,j}}{2^j - 1}.$$

Step 4: Return the element $a_{1,k+1}$ as the estimate of $\bar{D}$

Alternatively, one may estimate the relative truncation error after each step j from

$$relerr_j := \frac{2|a_{k+1-j,j+1} - a_{k+2-j,j}|}{|a_{k+1-j,j+1}| + |a_{k+2-j,j}| + \epsilon}$$

and stop the loop over $j = 2, \ldots, k$ whenever this error starts to increase between $j-1$ and j.

The truncation error of the central difference formula (14.4) has the polynomial representation

$$D(h) - \bar{D} = c_1 h^2 + c_2 h^4 + c_3 h^6 + \ldots.$$

To obtain improved estimates for the first derivative from this formula, we must replace $2^j - 1$ in Step 3 of the algorithm with $4^j - 1$.

Computation of the Jacobian. It is easy to apply the above results in computing the Jacobian matrix (13.10) of a vector-valued function $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$. By using the central difference formula (14.4), we may approximate the element $\partial f^i(\bar{\mathbf{x}})/\partial x_j$ by

$$\frac{\partial f^i(\bar{\mathbf{x}})}{\partial x_j} \simeq \frac{f(\bar{x} + \mathbf{e}_j h) - f(\bar{x} - \mathbf{e}_j h)}{2h}, \tag{14.9}$$

where $\mathbf{e}_j$ is the unit (row) vector with one in the j–th position and zeros elsewhere.

If the arguments x_j, $j = 1, \ldots, n$ differ considerably in size, we set h proportional to x_j using

$$h_j = \bar{h} \max\{|x_j|, 1\}. \tag{14.10}$$

The GAUSS procedure `CDJac` and the MATLAB® function `CDJac.m` as well as the Fortran subroutine `CDJac` employ equation (14.9) together with this choice of h_j (and $\bar{h} = \sqrt[3]{\epsilon}$, ϵ the machine epsilon as default) to compute the Jacobian of a user-supplied routine that evaluates $f^i(\bar{x}), i = 1, 2, \ldots, m$ at $\bar{x}$. The MATLAB® function `CDJacRE.m` combines (14.9) and Algorithm 14.2.1 to compute Jacobian matrices with a smaller truncation error.

14.2.2 Second-Order Derivatives

Now, suppose that we want to compute the elements of the Hessian matrix (13.5) of $f : \mathbb{R}^n \to \mathbb{R}$. There are two possibilities. Note that the Hessian matrix equals the Jacobian matrix of the gradient of f. Thus, if an analytic expression for the gradient of f is easy to program, one can use this as an input to a procedure that approximates the Jacobian. This gives a better approximation than the use of difference formulas for second partial derivatives.[1] If this is not an option, one can apply the central difference formula for the second derivative of a function in one variable to compute the diagonal elements of H.[2] This gives:

$$\frac{\partial^2 f(\bar{\mathbf{x}})}{\partial x_i \partial x_i} \simeq \frac{f(\bar{\mathbf{x}} + \mathbf{e}_i h_i) + f(\bar{\mathbf{x}} - \mathbf{e}_i h_i) - 2f(\bar{x})}{h_i^2}. \tag{14.11}$$

We can improve upon this estimate by using Richardson's extrapolation as defined in Algorithm 14.2.1.

There are several choices for the off-diagonal elements of the Hessian matrix. We now show that a four-point formula provides a second-order truncation error. Let f_i, f_{ij}, and f_{ijk} abbreviate the first-, second-, and third-order partial derivatives at the point $\bar{\mathbf{x}}$, and let $h_i, h_j \in \mathbb{R}_{>0}$ for some $i, j \in \{1, 2, \ldots, n\}$. Using Theorem (13.3.2) for $k = 3$ yields:

[1] As we demonstrated above, the error of the central difference formula for the first derivative is of second order, whereas the error from the central difference formula for the second derivative is of first order.

[2] In the following we use h_i proportional to $\max\{|x_i|, 1\}$ as in (14.10).

$$f(\bar{\mathbf{x}}+\mathbf{e}_i h_i+\mathbf{e}_j h_j)=f(\bar{\mathbf{x}})+f_i h_i+f_j h_j+\frac{1}{2}\sum_{i_1\in\{i,j\}}\sum_{i_2\in\{i,j\}} f_{i_1 i_2}h_{i_1}h_{i_2}$$
$$+\frac{1}{6}\sum_{i_1\in\{i,j\}}\sum_{i_2\in\{i,j\}}\sum_{i_3\in\{i,j\}} f_{i_1 i_2 i_3}h_{i_1}h_{i_2}h_{i_3}+C_1$$
$$f(\bar{\mathbf{x}}-\mathbf{e}_i h_i-\mathbf{e}_j h_j)=f(\bar{\mathbf{x}})-f_i h_i-f_j h_j+\frac{1}{2}\sum_{i_1\in\{i,j\}}\sum_{i_2\in\{i,j\}} f_{i_1 i_2}h_{i_1}h_{i_2}$$
$$-\frac{1}{6}\sum_{i_1\in\{i,j\}}\sum_{i_2\in\{i,j\}}\sum_{i_3\in\{i,j\}} f_{i_1 i_2 i_3}h_{i_1}h_{i_2}h_{i_3}+C_2,$$

where C_1 and C_2 are truncation errors that involve fourth-order terms. Adding both equations gives

$$\begin{aligned} &f(\bar{\mathbf{x}}+\mathbf{e}_i h_i+\mathbf{e}_j h_j)+f(\bar{\mathbf{x}}-\mathbf{e}_i h_i-\mathbf{e}_j h_j)\\ &=2f(\bar{\mathbf{x}})+f_{ii}h_i^2+f_{jj}h_j^2+(f_{ij}+f_{ji})h_i h_j+C_1+C_2. \end{aligned} \tag{14.12}$$

so that the third-order terms drop out. Using the same expansions for $f(\bar{\mathbf{x}}-\mathbf{e}_i h_i+\mathbf{e}_j h_j)$ and $f(\bar{\mathbf{x}}+\mathbf{e}_i h_i-\mathbf{e}_j h_j)$ yields

$$\begin{aligned} &f(\bar{\mathbf{x}}-\mathbf{e}_i h_i+\mathbf{e}_j h_j)+f(\bar{\mathbf{x}}+\mathbf{e}_i h_i-\mathbf{e}_j h_j)\\ &=2f(\bar{\mathbf{x}})+f_{ii}h_i^2+f_{jj}h_j^2-(f_{ij}+f_{ji})h_i h_j+C_3+C_4. \end{aligned} \tag{14.13}$$

Subtracting (14.13) from (14.12) and employing Young's theorem[3], i.e. $f_{ij}=f_{ji}$, finally produces

$$\begin{aligned} f_{ij}\simeq\frac{1}{4h_i h_j}\Big[&f(\bar{\mathbf{x}}+\mathbf{e}_i h_i+\mathbf{e}_j h_j)+f(\bar{\mathbf{x}}-\mathbf{e}_i h_i-\mathbf{e}_j h_j)\\ &-f(\bar{\mathbf{x}}-\mathbf{e}_i h_i+\mathbf{e}_j h_j)-f(\bar{\mathbf{x}}-\mathbf{e}_i h_i+\mathbf{e}_j h_j)\Big], \end{aligned} \tag{14.14}$$

with truncation error

$$C:=\frac{C_1+C_2-C_3-C_4}{4h_i h_j}.$$

Since the terms C_i, $i=1,\ldots,4$ are of order four, this error is of order two. We can therefore employ Algorithm 14.2.1 with 4^j-1 to reduce the truncation error further.

We implement the formulas (14.11) and (14.14) in our GAUSS procedure `CDHesse`, the Fortran subroutine `CDHesse`, and the MATLAB® function `CDHesse.m` The MATLAB® function `CDHesseRE` combines formulas (14.11) and (14.14) with Algorithm 14.2.1 to reduce the truncation error.

[3] See, e.g., Theorem 1.1 on p. 372 of Lang (1997).

14.3 Numerical Integration

14.3.1 Newton-Cotes Formulas

Basically, there are two different approaches to computing an integral $\int_a^b f(x)\,\mathrm{d}\,x$ numerically.[4] The first idea is to approximate the function $f(x)$ by piecewise polynomials and integrate the polynomials over subintervals of $[a,b]$. For example, the trapezoid rule evaluates the function $f(x)$ at the end points $x=a$ and $x=b$ and uses the linear Lagrange polynomial

$$P_1(x)=\frac{x-b}{a-b}f(a)+\frac{x-a}{b-a}f(b) \tag{14.15}$$

to approximate $f(x)$. Integration of P_1 over $[a,b]$ results in the formula

$$\int_a^b f(x)\,\mathrm{d}\,x\approx\frac{b-a}{2}\left[f(a)+f(b)\right]. \tag{14.16}$$

More generally, we can divide the interval $[a,b]$ into n equidistant subintervals of length $h=(b-a)/n$, $n\in\mathbb{N}$ and approximate $f(x)$ on each of them with the Lagrange polynomial of order n. This gives

$$\int_a^b f(x)\,\mathrm{d}\,x\approx\int_a^b\sum_{k=0}^{n}f(a+kh)L_k(x)\,\mathrm{d}\,x.$$

Defining $x=a+th$, the Lagrange polynomial $L_k(x)$ presented in (13.17) becomes

$$L_k(t)=\prod_{\substack{i=0\\k\neq i}}^{n}\frac{t-i}{k-i}.$$

Using the change of variable formula presented below in (14.21) to replace x with t yields the Newton-Cotes formula

$$\int_a^b f(x)\,\mathrm{d}\,x\approx h\sum_{k=0}^{n}w_k f(a+kh),\ w_k:=\int_0^n\prod_{\substack{i=0\\k\neq i}}^{n}\frac{t-i}{k-i}\,\mathrm{d}\,t. \tag{14.17}$$

[4] In fact, there is a third approach that we do not pursue here. It considers the related problem to solve an ordinary differential equation. See, e.g., Walter (2014), Section 6.2.4.

For $n = 2$, we obtain Simpson's rule[5]

$$\int_a^b f(x)\,\mathrm{d}x \approx \frac{h}{3}[f(a) + 4f(a+h) + f(b)]. \tag{14.18}$$

14.3.2 Gaussian Formulas

The weights w_k of the Newton-Cotes formula (14.17) ensure that the local approximation of $f(x)$ is correct and the nodes x_k follow from the arbitrary choice of n. The second approach, which we pursue in all quadrature applications of this book, is to choose both the weights w_k and the nodes x_k optimally to provide a good approximation of $\int_a^b f(x)\,\mathrm{d}x$. It is obvious that we increase the degrees of freedom at our disposal if we choose both the nodes x_k and the weights w_k simultaneously rather than just the weights w_k in order to obtain a good approximation. Essentially, the resulting Gaussian quadrature formulas have twice the order than that of the Newton-Cotes formulas for the same number of functions evaluations.[6]

Gauss-Chebyshev Integration. For the Chebyshev polynomials considered in Section 13.8, the following theorem provides the integration weights and nodes:

Theorem 14.3.1 *Let z_k^0, $k = 1, \ldots, K$ denote the zeros of $T_K(z)$ and*

$$\omega_k = \frac{\pi}{K}.$$

The approximation

$$I := \int_{-1}^{1} w(z) f(z)\,\mathrm{d}z \simeq \sum_{i=1}^{K} \omega_k f(z_k^0) =: I_K \tag{14.19}$$

is exact for all polynomials of degree $2K-1$ or less.

Now, since the degree of the polynomial $T_l(z) := T_i(z)T_j(z)$, $i, j < K-1$ from equation (13.31) is $l = i + j < 2K-2$, Theorem 14.3.1 implies that (13.32) equals the exact integral (13.31).

[5] For the computation of w_k, $k = 0, 1, 2$ see Stoer and Bulirsch (2002), p. 147.

[6] Notice, however, that higher order does not always translate into higher accuracy.

For functions that are $2K$ times continuously differentiable on $[-1,1]$, the approximation error is given by:[7]

$$I - I_n = \frac{\pi}{2^{2K-1}} \frac{f^{2K}(\xi)}{(2K)!},\ \xi \in [-1,1].$$

The Gauss-Chebyshev quadrature formula based on the extrema of $T_n(z)$ is[8]

$$I_n = \frac{\pi}{K}\left[\tfrac{1}{2}f(\bar{z}_0) + f(\bar{z}_1) + \cdots + f(\bar{z}_{K-1}) + \tfrac{1}{2}f(\bar{z}_K)\right]. \tag{14.20}$$

Change of Variable Technique. Gaussian quadrature formulas, such as the Gauss-Chebyshev formulas from the previous subsection, rest on specific weight functions and specific domains. Therefore, we must adapt those rules to approximate the integral of a specific function f on a given domain. The respective tool is the change of variable technique. We present it here for the general case of a function $f : X \to \mathbb{R}$ that maps some bounded subset $X \subset \mathbb{R}^n$ of Euclidean n space to the real line. Let

$$\mathbf{h} : X \to Y \subset \mathbb{R}^n,\ \mathbf{x} \mapsto \mathbf{y} := \mathbf{h}(\mathbf{x})$$

define a one-to-one map between the bounded sets X and Y so that

$$y_i = h^i(x_1, \ldots, x_n).$$

Furthermore, let

$$J(\mathbf{h})(\mathbf{x}) := \begin{bmatrix} \frac{\partial h^1(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial h^1(\mathbf{x})}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial h^n(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial h^n(\mathbf{x})}{\partial x_n} \end{bmatrix}$$

denote the Jacobian matrix of $\mathbf{h}$ and assume that this matrix is nonsingular on X. If f is integrable on X and $\mathbf{h}$ differentiable on X, then the change of variables formula asserts:[9]

$$I(\mathbf{y}) := \int_Y f(\mathbf{y})\,\mathrm{d}\,\mathbf{y} = \int_X f(\mathbf{h}(\mathbf{x}))|J(\mathbf{h})(\mathbf{x})|\,\mathrm{d}\,\mathbf{x}. \tag{14.21}$$

[7] See, e.g., Chawla (1968), equation (2) or Judd (1998), equation (7.2.4) on p. 259.

[8] See, e.g., Mason and Handscomb (2003), equation (8.32) on p. 208).

[9] See, e.g., Judd (1998), Theorem 7.5.3, p. 275 or Sydsæter et al. (1999), formula 9.73, p. 54.

Note that we replace $\mathbf{y}$ with $\mathbf{h}(\mathbf{x})$ and additionally adjust the integral by the determinant of the Jacobian matrix $|J(\mathbf{h})(\mathbf{x})|$.[10]

In the univariate case considered in the previous subsection, the relation between $z \in [-1,1]$ and $x \in [a,b]$ is given by the bijection (13.28a), which has the Jacobian matrix

$$\psi'(z) = \frac{b-a}{2}.$$

Accordingly, the integral of $f : [a,b] \to \mathbb{R}$ equals

$$\int_a^b f(x)\,\mathrm{d}\,x = \int_{-1}^{1} \frac{f(\psi(z))}{\sqrt{1-z^2}} \sqrt{1-z^2}\,\frac{b-a}{2}\,\mathrm{d}\,z.$$

Applying the formula (14.19) to the function $g(z) := f(\psi(z))\sqrt{1-z^2}$ gives

$$\int_a^b f(x)\,\mathrm{d}\,x \approx \frac{\pi(b-a)}{2n} \sum_{k=1}^{K} f(\psi(z_k^0))\sqrt{1-(z_k^0)^2}. \tag{14.22}$$

In the same way, we can use the formula (14.20) to approximate the integral of $f(x)$ on the extrema of $T_K(z)$.

The most natural way to extend this formula to d dimensions it through product rules. They approximate multiple integrals by multiple sums. For example, consider the integral of $f : X \to \mathbb{R}$, $X \subset \mathbb{R}^d$ over the d-dimensional rectangle $[a_1,b_1] \times \cdots \times [a_d,b_d]$. Its approximation by a product rule based on the Gauss-Chebyshev formula (14.22) is:

$$\begin{aligned}
&\int_{a_1}^{b_1} \cdots \int_{a_d}^{b_d} f(x_1,\ldots,x_d)\,\mathrm{d}\,x_1 \cdots \mathrm{d}\,x_d \\
&\approx \frac{\pi^d (b_1-a_1)\cdots(b_d-a_d)}{(2m)^d} \sum_{k_1}^{m} \cdots \sum_{k_d}^{m} f(\psi(z_{k_1}^0),\ldots,\psi(z_{k_d}^0)) \\
&\times \sqrt{1-(z_{k_1}^0)^2} \cdots \sqrt{1-(z_{k_d}^0)^2}.
\end{aligned} \tag{14.23}$$

14.3.3 Monomial Integration Formula

Notation. Stroud (1971) provides quadrature formulas for Riemann integrals of a function $f : X \to \mathbb{R}$, $X \subset \mathbb{R}^n$ over various regions and for

[10] For an intuitive explanation, see Judge et al. (1988), p. 31.

various weighting functions $w(\mathbf{x})$. They take the form

$$\int_{a_1}^{b_1} \cdots \int_{a_n}^{b_n} f(\mathbf{x}) w(\mathbf{x}) \, \mathrm{d}\, x_1 \cdots \mathrm{d}\, x_n \approx \sum_{j=1}^{J} B_j f(v_{j1}, v_{j2}, \ldots, v_{jn}), \quad \mathbf{x} \in X \subset \mathbb{R}^n, B_j \in \mathbb{R}, \tag{14.24}$$

where the vector $\mathbf{v}_j := (v_{j1}, v_{j2}, \ldots, v_{jn}) \in \mathbb{R}^n$ denotes the jth integration node. The rule (14.24) has *degree* d if it is exact for any combination of monomials

$$g(\mathbf{z}) := z_1^{\alpha_1} z_2^{\alpha_2} \cdots z_n^{\alpha_n}, \quad \alpha_1, \alpha_2, \ldots, \alpha_n \geq 0, 0 \leq \sum_{i=1}^{n} \alpha_i \leq d$$

but not exact for at least one polynomial of degree $d+1$ (Stroud (1971), p. 3).

Integration Formula. For the integrals of f over the hypercube $C_n := [-1,1]^n$ (the n-fold Cartesian product of the closed interval $[-1,1]$) and the weight function $w(\mathbf{x}) = 1$,

$$I_{C_n}(f) := \int_{-1}^{1} \cdots \int_{-1}^{1} f(x_1, x_2, \ldots, x_n) \, \mathrm{d}\, x_1 \cdots \mathrm{d}\, x_n$$

we reproduce two formulas. The degree $d = 3$ formula uses $J = 2n+1$ integration nodes (see Stroud (1971), p. 230):

$$I_{C_n}(f) \approx B_0 f(0,0,\ldots,0) + B_1 \sum_{i=1}^{n} (f(\mathbf{e}_i) + f(-\mathbf{e}_i)),$$
$$B_0 = \frac{3-n}{3} 2^n, \; B_1 = \frac{1}{6} 2^n. \tag{14.25}$$

The second formula is of degree $d = 5$, employing $J = 2^n + 2n + 1$ integration (see Stroud (1971), p. 233):

$$I_{C_n}(f) \approx B_0 f(0,0,\ldots,0) + B_1 \sum_{i=1}^{n} (f(r\mathbf{e}_i) + f(-r\mathbf{e}_i)) + B_2 \sum_{\mathbf{v} \in V} f(\mathbf{v}),$$
$$r = \sqrt{\frac{2}{5}}, \; B_0 = \frac{8-5n}{9} 2^n, \; B_1 = \frac{5}{18} 2^n, \; B_2 = \frac{1}{9}. \tag{14.26}$$

In this formula, the symbol $\mathbf{e}_i$ denotes the ith unit vector, i.e., the n-dimensional vector with all elements equal to zero except its ith element,

which is equal to one. The set V consists of the 2^n elements constructed from n draws of the set $\{-1, 1\}$. For instance, if $n = 3$ this set consists of the eight points

$$\begin{aligned}
\mathbf{v}_1 &:= (1,1,1), & \mathbf{v}_5 &:= (-1,1,1),\\
\mathbf{v}_2 &:= (1,1,-1), & \mathbf{v}_6 &:= (-1,1,-1),\\
\mathbf{v}_3 &:= (1,-1,1), & \mathbf{v}_7 &:= (-1,-1,1),\\
\mathbf{v}_4 &:= (1,-1,-1), & \mathbf{v}_8 &:= (-1,-1,-1).
\end{aligned}$$

If the domain of the function f is not C_n but $X := [a_1, b_1] \times \cdots \times [a_n, b_n]$, we can employ the same change of variable formula used in the previous subsection to map points in $[-1, 1]$ to $[a_i, b_i]$.

14.4 Approximation of Expectations

This section explains the numerical approximation of expectations with Gauss-Hermite and monomial integration or, interchangeably, quadrature formulas. We focus on Gaussian random variables, as explained in the next subsection. Afterwards, we introduce Gauss-Hermite integration. To overcome the curse of dimensionality in cases with many random variables, we present several monomial integration formulas in Subsection 14.3.3.

14.4.1 Expectation of a Function of Gaussian Random Variables

Let us suppose that the vector of random variables $\mathbf{x} \in \mathbb{R}^n$ is normally distributed with mean $\mathbb{E}(\mathbf{x}) = \boldsymbol{\mu}$ and covariance matrix $\text{var}(\mathbf{x}) := \mathbb{E}\left(\mathbf{x}\mathbf{x}^T\right) = \Sigma$ so that the probability density function is given by

$$f(\mathbf{x}) = \pi^{-\frac{n}{2}} |\Sigma|^{-\frac{1}{2}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x}-\boldsymbol{\mu})}. \tag{14.27}$$

Let $g(\mathbf{x})$ denote some function of this random vector. The problem that we want to solve is to numerically approximate the expectation

$$\mathbb{E}(g(\mathbf{x})) := \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} g(\mathbf{x}) f(\mathbf{x}) \, \mathrm{d}\, x_1 \cdots \mathrm{d}\, x_n. \tag{14.28}$$

We transform this problem to a simpler one that replaces the exponential in (14.27) with the weight function

$$w(\mathbf{x}) = e^{-z_1^2 - z_2^2 - \cdots - z_n^2}$$

for which there exist quadrature formulas. Let Ω denote the $n \times n$ square matrix with the property

$$\Sigma = \Omega\Omega^T.$$

Since Σ is the covariance matrix of the random variables $x_1, \ldots, x_N$, it is symmetric and positive definite. Accordingly, we have two choices for the matrix Ω. The first derives from the Jordan factorization of Σ and the second from the Cholesky factorization . The former is given $\Omega := P\Lambda^{1/2}$ (see equation (12.32)) and the latter by $\Omega := L$ (see equation 12.37). Given the matrix Ω, we can define the vector $\mathbf{z}$ as:

$$\mathbf{z} = \frac{\Omega^{-1}(\mathbf{x} - \boldsymbol{\mu})}{\sqrt{2}}$$

so that the exponent in (14.27) is given by

$$\begin{aligned} -\tfrac{1}{2}\left(\sqrt{2}\mathbf{z}^T\Omega^T\right)\Sigma^{-1}\left(\sqrt{2}\Omega\mathbf{z}\right) &= -\mathbf{z}^T\Omega^T\left(\Omega\Omega^T\right)^{-1}\Omega\mathbf{z} \\ &= -\mathbf{z}^T\Omega^T(\Omega^T)^{-1}\Omega^{-1}\Omega\mathbf{z} = -\mathbf{z}^T\mathbf{z}. \end{aligned}$$

We can now apply the change of variable technique. The Jacobian matrix of $\mathbf{x} = \boldsymbol{\mu} + \sqrt{2}\Omega\mathbf{z}$ is $\sqrt{2}\Omega$ with determinant:[11]

$$|\Sigma|^{-\frac{1}{2}} = |\Omega\Omega^T|^{-\frac{1}{2}} = (|\Omega||\Omega^T|)^{-\frac{1}{2}} = (|\Omega|^2)^{-\frac{1}{2}} = |\Omega|^{-1}.$$

By using this in the formula (14.21), we can write expression (14.28) in terms of the vector of uncorrelated random variables $\mathbf{z}$ as:

$$\mathbb{E}(g(h(\mathbf{z}))) := \pi^{-\frac{n}{2}} \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} g\left(\sqrt{2}\Omega\mathbf{z} + \boldsymbol{\mu}\right) e^{-\mathbf{z}^T\mathbf{z}}\, \mathrm{d}z_1 \cdots \mathrm{d}z_n. \quad (14.29)$$

14.4.2 Gauss-Hermite Integration

Univariate Integration Formula. Gauss-Hermite integration derives its weights ω_i and integration nodes z_k, $k = 1, 2, \ldots, m$ from the zeros of the Hermite polynomials introduced in Section 13.7.4. In the single

[11] See (12.10) for these derivations.

variable case, let $z \in \mathbb{R}$. The integral of $g(z)$ with weight function e^{-z^2} equals

$$I(z) := \int_{-\infty}^{\infty} g(z)e^{-z^2}\,\mathrm{d}\,z = \sum_{k=1}^{m} \omega_k g(z_k) + \frac{(m!)\sqrt{\pi}}{2^m(2m)!} g^{2m}(\xi), \xi \in \mathbb{R}, \tag{14.30}$$

where $g^k(z)$ denotes the kth derivative of g at the point z. Accordingly, if g is a polynomial of degree $2m-1$ (such that $g^{2m}(z) = 0$), the remainder term vanishes and the formula is said to be exact of degree $2m-1$.

Using this formula and the change of variable formula (14.21) with $x = \mu + \sqrt{2}\sigma z$, the expectation (14.28) can be approximated by

$$\begin{aligned} \mathbb{E}(g(x)) &= \frac{1}{\sqrt{2\pi}\sigma} \int_{-\infty}^{\infty} g(x) e^{-\frac{(x-\mu)^2}{2\sigma^2}}\,\mathrm{d}\,x \\ &\approx \frac{1}{\sqrt{2\pi}\sigma} \sum_{k=1}^{m} \omega_k g(\mu + \sqrt{2}\sigma z_k)\sqrt{2}\sigma, \\ &= \sum_{k=1}^{m} \tilde{\omega}_k g(\mu + \sqrt{2}\sigma z_k),\ \tilde{\omega}_k := \frac{\omega_k}{\sqrt{\pi}}. \end{aligned} \tag{14.31}$$

Integration Nodes and Weights. There exist tables with nodes and weights (see, e.g., Judd (1998), p. 262, Table 7.4). Instead, one can follow Golub and Welsch (1969) and compute any number of nodes and weights. They show that the nodes associated with Gaussian quadrature rules based on orthogonal polynomials $\varphi_k(x)$ are equal to the eigenvalues of the tridiagonal matrix

$$J := \begin{bmatrix} \alpha_1 & \beta_1 & 0 & 0 & \dots & 0 & 0 \\ \beta_1 & \alpha_2 & \beta_2 & 0 & \dots & 0 & 0 \\ 0 & \beta_2 & \alpha_3 & \beta_3 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \\ 0 & 0 & 0 & 0 & \dots & \alpha_{m-1} & \beta_{m-1} \\ 0 & 0 & 0 & 0 & \dots & \beta_{m-1} & \alpha_m \end{bmatrix}, \quad \alpha_k := -\frac{b_k}{a_k}, \beta_k := \left(\frac{c_{k+1}}{a_k a_{k+1}}\right)^{\frac{1}{2}},$$

where a_k, b_k, and c_k are the coefficients introduced in equation (13.22). It follows from equation (13.23) that $a_i = 2$, $b_i = 0$, and $c_i = 2(i-1)$ so that the matrix J simplifies to

$$J := \begin{bmatrix} 0 & \beta_1 & 0 & 0 & \dots & 0 & 0 \\ \beta_1 & 0 & \beta_2 & 0 & \dots & 0 & 0 \\ 0 & \beta_2 & 0 & \beta_3 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \\ 0 & 0 & 0 & 0 & \dots & 0 & \beta_{n-1} \\ 0 & 0 & 0 & 0 & \dots & \beta_{n-1} & 0 \end{bmatrix}, \quad \beta_i := \left(\frac{i}{2}\right)^{\frac{1}{2}}. \tag{14.32}$$

The weights are equal to the squared elements of the first component of the orthonormal eigenvectors of J times the integral

$$\int_{-\infty}^{\infty} e^{-z^2}\,\mathrm{d}z = \sqrt{\pi}.$$

Accordingly, instead of using the weights ω_k in the quadrature rule (14.31), we need to compute only the weights $\tilde{\omega}_k$. It is easy to implement the computation of Gauss-Hermite nodes and weights in matrix oriented languages as our MATLAB® function `GH_NW.m` demonstrates.

Multivariate Extension. Since the elements of the vector $\mathbf{z}$ are uncorrelated, we can evaluate the n-fold integral element wise and approximate each of the partial integrals by a weighted sum with n_{GH} elements:

$$\int_{-\infty}^{\infty} g(z_1,\dots,z_i,\dots z_n)e^{-z_i^2}\,\mathrm{d}z_i \approx \sum_{k=1}^{n_{GH}} \omega_{ik}\, g(z_1,\dots,z_{ik},\dots,z_n).$$

The entire integral (14.29) is then approximately equal to the n-fold sum

$$\begin{aligned}\mathbb{E}_t(g(h(\mathbf{z}))) \approx \pi^{-\frac{n}{2}} \sum_{i_1=1}^{n_{GH}} \cdots \sum_{i_n=1}^{n_{GH}} \omega_{i_1}\omega_{i_2}\cdots\omega_{i_n} \\ \times\, g\left(\sqrt{2}\Omega(z_{1,i_1},\dots,z_{n,i_n})^T + \boldsymbol{\mu}\right).\end{aligned} \tag{14.33}$$

Our MATLAB® function `GH_quad.m` implements equation (14.33). For a given matrix root Ω and the desired number of integration nodes n_{GH}, it constructs the nodes $\sqrt{2}\Omega\mathbf{z}_k$ and the respective products of weights.

14.4.3 Monomial Rules for Expectations

For the weight function in (14.27),

$$w(\mathbf{z}) := e^{-z_1^2} e^{z_2^2} \cdots e^{-z_n^2} = e^{-\mathbf{z}^T \mathbf{z}}$$

and the related integral

$$I(\mathbf{z}) := \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} g(\mathbf{z}) e^{-\mathbf{z}^T \mathbf{z}} \, \mathrm{d}z_1 \cdots \mathrm{d}z_n$$

Stroud (1971), p. 315, presents several monomial formulas (see also Section 14.3.3). We reproduce three of them.

Degree 3, 2n Nodes. A formula of degree $d = 3$ with only $2n$ nodes is (Stroud (1971), p. 315):

$$I(\mathbf{z}) \approx \frac{\pi^{\frac{n}{2}}}{2n} \sum_{i=1}^{n} \Big[g(\sqrt{n/2}\mathbf{e}_i) + g(-\sqrt{n/2}\mathbf{e}_i) \Big], \tag{14.34}$$

with $\mathbf{e}_i$ as the ith unit vector. Accordingly, the expectation (14.29) is approximately equal to

$$\mathbb{E}(g(h(\mathbf{z}))) \approx \frac{1}{2n} \sum_{i=1}^{n} \Big[g\left(\sqrt{n}\Omega\mathbf{e}_j + \boldsymbol{\mu}\right) + g\left(-\sqrt{n}\Omega\mathbf{e}_j + \boldsymbol{\mu}\right) \Big]. \tag{14.35}$$

Degree 3, 2^n Nodes. The formula of degree $d = 3$ with 2^n nodes is (Stroud (1971), p. 316):

$$I(\mathbf{z}) \approx \frac{\pi^{\frac{n}{2}}}{2^n} \sum_{\mathbf{v} \in V} f\left(\mathbf{v}\sqrt{1/2}\right), \tag{14.36}$$

where V is the set already encountered in equation (14.26) so that the expectation (14.29) is approximately equal to

$$\mathbb{E}(g(h(\mathbf{z}))) \approx \frac{1}{2^n} \sum_{\mathbf{v} \in V} g\left(\Omega\mathbf{v} + \boldsymbol{\mu}\right). \tag{14.37}$$

Degree 5, $2n^2+1$ Nodes. Let (s_1, s_2) represent the result of two draws from the two-element set $\{s, -s\}$ with $s = \sqrt{(n+2)}/2$, and denote by S the set of points obtained from replacing all possible combinations of two elements of the zero vector $(0, 0, \ldots, 0)$ with (s_1, s_2). This set has $2n(n-1)$ elements. For example, if $n = 3$, this set consists of the twelve points

$$\begin{aligned}
&\mathbf{s}_1 := (s,s,0), \quad &&\mathbf{s}_5 := (s,0,s), \quad &&\mathbf{s}_9 := (0,s,s),\\
&\mathbf{s}_2 := (s,-s,0), \quad &&\mathbf{s}_6 := (s,0,-s), \quad &&\mathbf{s}_{10} := (0,s,-s),\\
&\mathbf{s}_3 := (-s,s,0), \quad &&\mathbf{s}_7 := (-s,0,s), \quad &&\mathbf{s}_{11} := (0,-s,s),\\
&\mathbf{s}_4 := (-s,-s,0), \quad &&\mathbf{s}_8 := (-s,0,-s), \quad &&\mathbf{s}_{12} := (0,-s,-s).
\end{aligned}$$

The formula of degree 5 with $2^n + 1$ nodes is (Stroud (1971), p. 317):

$$\begin{aligned}
I(\mathbf{x}) \approx &\frac{2\pi^{\frac{n}{2}}}{n+2} g(0,\ldots,0)\\
&+ \frac{(4-n)\pi^{\frac{n}{2}}}{2(n+2)^2} \sum_{j=1}^{n} \Big[g(\sqrt{(n+2)/2}\mathbf{e}_j) + g(-\sqrt{(n+2)/2}\mathbf{e}_j) \Big],\\
&+ \frac{\pi^{\frac{n}{2}}}{(n+2)^2} \sum_{\mathbf{s}\in S} g(\mathbf{s}).
\end{aligned} \tag{14.38}$$

With this rule, the expectation (14.29) is approximately equal to

$$\begin{aligned}
\mathbb{E}(g(h(\mathbf{z}))) \approx &\frac{2}{n+2} g(\boldsymbol{\mu})\\
&+ \frac{4-n}{2(n+2)^2} \sum_{j=1}^{n} g(\Omega\sqrt{n+2}\mathbf{e}_j + \boldsymbol{\mu})\\
&+ \frac{4-n}{2(n+2)^2} \sum_{j=1}^{n} g(-\Omega\sqrt{n+2}\mathbf{e}_j + \boldsymbol{\mu})\\
&+ \frac{1}{(n+2)^2} \sum_{\mathbf{s}\in S} g(\sqrt{2}\Omega\mathbf{s} + \boldsymbol{\mu}).
\end{aligned} \tag{14.39}$$

Degree 5, $2n^2+2n$ Nodes This rule combines the $2n$ nodes from (14.34) with the set V from (14.36):

$$\begin{aligned}
I(\mathbf{x}) \approx &\frac{4\pi^{\frac{n}{2}}}{(n+2)^2} \sum_{j=1}^{n} \Big[g\left(\sqrt{(n+2)/4}\mathbf{e}_j\right) + g\left(-\sqrt{(n+2)/4}\mathbf{e}_j\right) \Big]\\
&+ \frac{(n-2)^2\pi^{\frac{n}{2}}}{2^n(n+2)^2} \sum_{\mathbf{v}\in V} g\left(\mathbf{v}\sqrt{(n+2)/(2(n-2))}\right).
\end{aligned} \tag{14.40}$$

Accordingly, the rule approximates the expectation (14.29) with

$$\begin{aligned}\mathbb{E}(g(h(\mathbf{z}))) \approx & \frac{4}{(n+2)^2}\sum_{j=1}^{n} g\left(\sqrt{n+2}\Omega\mathbf{e}_j/\sqrt{2}+\boldsymbol{\mu}\right) \\ & + \frac{4}{(n+2)^2}\sum_{j=1}^{n} g\left(-\sqrt{n+21}\Omega\mathbf{e}_j/\sqrt{2}+\boldsymbol{\mu}\right) \\ & + \frac{n-2}{2^n(n+2)^2}\sum_{\mathbf{v}\in V} g\left(\Omega\mathbf{v}\sqrt{(n+2)/(n-2)}+\boldsymbol{\mu}\right). \quad (14.41)\end{aligned}$$

Our MATLAB® function `Mon_quad` implements the integration formulas (14.34), (14.36), (14.38), and (14.40). Its arguments are the matrix root Ω and an indicator for the desired rule, which takes the values 1, 2, 3, 4. The function returns the integration nodes, say, $\mathbf{z} = \mathbf{x} - \boldsymbol{\mu}$, and the weights of the respective formula. Added to the vector $\boldsymbol{\mu}$, these nodes can be passed to a function that computes $g(\boldsymbol{\mu}+\mathbf{z})$.