Chapter 14

Numerical Integration

14.1 Introduction

Integrals of the form $\int_a^b g(x)dx$ arise often in economic models. One example is the aggregating of consumption amounts of a continuum of differentiated goods $c_t(i)$,

$$C_t = \left(\int_0^1 \alpha_i^{\frac{1}{\varepsilon}} c_t(i)^{\frac{\varepsilon-1}{\varepsilon}} di \right)^{\frac{\varepsilon}{\varepsilon-1}} \tag{14.1}$$

where C_t is aggregate consumption, α_i is a weight on the particular amount of consumption of good i, ε is the constant elasticity of substitution between different goods i, and the measure of goods is normalized to be between 0 and 1, without loss of generality.[1] Another key example of an integral that often occurs in macroeconomics is the expectations operator on the right-hand-side of the standard intertermporal Euler equation,

$$\begin{aligned} & u'(c_t) = \beta E_{z_{t+1}|z_t} \Big[(1 + r_{t+1} - \delta) u'(c_{t+1}) \Big] \\ \Rightarrow \quad & u'(c_t) = \beta \int_a^b \Big(1 + r_{t+1}(z_{t+1}) - \delta \Big) u' \Big(c_{t+1}(z_{t+1}) \Big) f(z_{t+1}|z_t) dz_{t+1} \end{aligned} \tag{14.2}$$

where a and b are the bounds of the support of z_{t+1} and $f(z_{t+1}|z_t)$ is the pdf of z_{t+1} that could potentially be conditional on z_t.

[1] The consumption aggregator in (**??**) is often called the Armington aggregator as it was first proposed in **?**. It is also known as a Dixit-Stiglitz aggregator after its use in **?**.

It is a rare convenience when these integrals can be evaluated analytically. However, it does not take much richness in functional form to render analytical solutions impossible for many integrals in economic models. In these cases, the integral must be computed numerically. The following discussion of numerical integration draws from the great treatments of the subject in **?**, pp. 598-603, **?**, Ch. 7, and **?**, pp.55-60.

14.2 Newton-Cotes Quadrature

Newton-Cotes quadrature forumalas approximate the integral of a function $\int_a^b g(x)dx$ by evaluating the function at N equally spaced nodes $\{x_1, x_2, ...x_N\}$ and weighting those nodes with N weights $\{\omega_1, \omega_2, ...\omega_N\}$. The general form of Newton-Cotes quadrature forumulas is

$$\int_a^b g(x)dx \approx \sum_{n=1}^{N} \omega_n g(x_n) \tag{14.3}$$

14.2.1 Midpoint rule (1 node)

The midpoint rule is the simplest Newton-Cotes formula and uses only one node or evaluation of the function. The midpoint formula simply evaluates the function at the midpoint of the domain of $x = \frac{a+b}{2}$ and assumes that the function is a constant at that level over the entire domain of $x \in [a, b]$.

$$\int_a^b g(x)dx \approx (b-a)g\left(\frac{a+b}{2}\right) \tag{14.4}$$

A more sophisticated midpoint rule is the composite midpoint rule, which breaks up the domain of the function $g(x)$ into N intervals and applies the midpoint rule to each interval. For nodes $x_0, x_1, \ldots, x_{N-1}$ with $x_i = a + \frac{(2i+1)(b-a)}{2N}$, the composite midpoint rule is given by

$$\int_a^b g(x)dx \approx \frac{b-a}{N} \sum_{i=0}^{N-1} g(x_i) \tag{14.5}$$

14.2.2 Trapezoid rule (2 nodes)

The trapezoid rule estimates the integral as the area under a line that connects the function $g(x)$ at the two endpoints a and b.

$$\int_a^b g(x)dx \approx \frac{b-a}{2}\left[g(a)+g(b)\right] \tag{14.6}$$

A more sophisticated trapezoid rule is the composite trapezoid rule, which breaks up the domain of the function $g(x)$ into N intervals and applies the trapezoid rule to each interval. For nodes $x_0, x_1, \ldots, x_N$ with $x_i = a + i(b-a)/N$, the composite trapezoid rule is given by

$$\int_a^b g(x)dx \approx \frac{b-a}{2N}\left[g(x_0) + 2\sum_{i=1}^{N-1} g(x_i) + g(x_N)\right] \tag{14.7}$$

14.2.3 Simpson's rule (3 nodes)

Simpson's rule offers a smooth nonlinear (quadratic) alternative the linear approximations of the midpoint and trapezoid rules. Simpson's rule finds the unique quadratic function in x that passes through the end points and the midpoint of the function $g(a)$, $g\left(\frac{a+b}{2}\right)$, and $g(b)$, which produces the following weights and values.

$$\int_a^b g(x)dx \approx \frac{b-a}{6}\left[g(a) + 4g\left(\frac{a+b}{2}\right) + g(b)\right] \tag{14.8}$$

Again, a more sophisticated Simpson's rule is the composite Simpson's rule, which breaks up the domain of the function $g(x)$ into N intervals and applies the Simpson's rule to each interval. For nodes $x_0, x_1, \ldots, x_{2N}$ with $x_i = a + i(b-a)/(2N)$, the composite Simpson's rule is given by

$$\int_a^b g(x)dx \approx \frac{b-a}{3(N+1)}\left[g(x_0) + 4\sum_{i=1,3,\ldots}^{2N-1} g(x_i) + 2\sum_{i=2,4,\ldots}^{2N-2} g(x_i) + g(x_{2N})\right] \tag{14.9}$$

Exercise 14.1. You can verify that the analytical solution to the integral of the function

$$g(x) = 0.1x^4 - 1.5x^3 + 0.53x^2 + 2x + 1$$

between $x = -10$ and $x = 10$ is $\int_{-10}^{10} g(x)dx = 4,373.3\bar{3}$. Write a Python function that will take as arguments an anonymous function that the user specifies representing $g(x)$, integration bounds a and b, the number of intervals N, and `method = {'midpoint', 'trapezoid', 'Simpsons'}`. Using the composite methods, evaluate the numerical approximations of the integral $\int_a^b g(x)dx$ using all three Newton-Cotes methods in your function and compare the difference between the values of these integrals to the true analytical value of the integral.

Exercise 14.2. Write a Python function that makes a Newton-Cotes discrete approximation of the distribution of the normally distributed variable $Z \sim N(\mu, \sigma)$. Let this function take as arguments the mean μ, the standard deviation σ, the number of equally spaced nodes N to estimate the distribution, and the number of standard deviations k away from μ to make the furthest nodes on either side of μ. Use the `scipy.stats.norm.cdf` command for the cdf of the normal distribution to compute the weights ω_n for the nodes x_n. Have this function return a vector of nodes of $[Z_1, Z_2, ...Z_N]$ and a vector of weights $[\omega_1, \omega_2, ...\omega_N]$ such that ω_i is given by the integral under the normal distribution between the midpoints of the two closest nodes. Define $f(Z; \mu, \sigma)$ as the pdf of the normal distribution and $F(Z; \mu, \sigma)$ as the cdf.

$$\omega_i = \begin{cases} F\left(\frac{Z_1+Z_2}{2}; \mu, \sigma\right) & \text{if} \quad i = 1 \\ \int_{Z_{min}}^{Z_{max}} f(Z; \mu, \sigma)dZ & \text{if} \quad 1 < i < N \\ 1 - F\left(\frac{Z_{N-1}+Z_N}{2}; \mu, \sigma\right) & \text{if} \quad i = N \end{cases}$$

$$\text{where} \quad Z_{min} = \frac{Z_{i-1} + Z_i}{2} \quad \text{and} \quad Z_{max} = \frac{Z_i + Z_{i+1}}{2}$$

What are the weights and nodes $\{\omega_n, Z_n\}_{n=1}^N$ for $N = 11$?

Exercise 14.3. If $Z \sim N(\mu, \sigma)$, then $A \equiv e^Z \sim LN(\mu, \sigma)$ is distributed lognormally and $\log(A) \sim N(\mu, \sigma)$. Use your knowledge that $A \equiv e^Z$, $\log(A) \sim N(\mu, \sigma)$, and your function from Exercise **??** to write a function that gives a discrete approximation to the lognormal distribution. Note: You will not end up with evenly spaced nodes $[A_1, A_2, ...A_N]$, but your weights should be the same as in Exercise **??**.

Exercise 14.4. Let Y_i represent the income of individual i in the United States for all individuals i. Assume that income Y_i is lognormally distributed in the U.S. according to $Y_i \sim LN(\mu, \sigma)$, where the mean of log income is $\mu = 10.5$ and the standard deviation of log income is $\sigma = 0.8$. Use your function from Exercise **??** to compute an approximation of the expected value of income or average income in the U.S. How does your approximation compare to the exact expected value of $E[Y] = e^{\mu + \frac{\sigma^2}{2}}$?

14.3 Gaussian Quadrature

In Newton-Cotes qudrature, the nodes are uniformly spaced. Gaussian quadrature formulas for approximating an integral take the same approximation form $\int_a^b g(x)dx \approx \sum_{n=1}^N \omega_n g(x_n)$ and optimally choose the weights ω_n and unevenly spaced nodes x_n given the total number of nodes N and some approximating polynomial class $h_i(x)$. The N weights and nodes are chosen to make an *exact integration* relationship hold. That is, for polynomials of order $2N-1$ the N weights and nodes must exactly satisfy

$$\int_a^b h_i(x)dx = \sum_{n=1}^N \omega_n h_i(x_n) \quad \text{for} \quad i = 0, 1, \ldots, 2N-1 \tag{14.10}$$

where $h_i(x)$ is an i-order polynomial in x. If the $h_i(x)$ form a basis, this means that every polynomial of degree less than or equal to $2N-1$ will be computed exactly using the N weights and N nodes.

As a simple example, suppose we want to approximate an arbitrary function $g(x)$ with Gaussian quadrature using a simple class of polynomials $h_i(x) = x^i$ and only $N = 2$ weights and nodes. (**??**) implies a system of four equations used to determine the four variables

$(\omega_1, \omega_2, x_1, x_2)$ to approximate the integral $\int_a^b g(x)dx \approx \sum_{n=1}^{N} \omega_n g(x_n)$.

$$\begin{aligned}
\int_a^b dx &= \omega_1 + \omega_2 \\
\int_a^b x dx &= \omega_1 x_1 + \omega_2 x_2 \\
\int_a^b x^2 dx &= \omega_1 x_1^2 + \omega_2 x_2^2 \\
\int_a^b x^3 dx &= \omega_1 x_1^3 + \omega_2 x_2^3
\end{aligned} \tag{14.11}$$

For $N = 2$, the optimal weights and nodes that solve the system (**??**) are $(\omega_1, \omega_2, x_1, x_2) = (1, 1, -0.578, 0.578)$. The Python code to solve this nonlinear system could be a simple root finder such as `scipy.optimize.root` or one of the constrained minimizers in `scipy.optimize.minimize`. In general, the spacing of the nodes will not be uniform.

The accuracy of the Gaussian quadrature approximation of the integral $\int_a^b g(x)dx$ increases in the number of nodes N. The accuracy of the approximation of the integral can also be improved by the choice of polynomial family $h_i(x)$. In particular, the families of orthonormal polynomials have multiple desirable properties. Because of the orthogonality of their coefficients, the system (**??**) is easier to solve due to the lack of collinearity.Also, the weights ω_n turn out to be the zeros of the orthogonal polynomial family. Lastly, these orthogonal families of polynomials can give very accurate solutions to integrals of the form $\int_a^b w(x)g(x)dx$, where $w(x)$ is the weighting function of an orthonormal family of polynomials.

For a more detailed discussion of the theory behind Gaussian quadrature, see **?**, pp. 257-265 and **?**, 599-601. The general applicability of Gaussian quadrature and its accuracy and efficiency advantage over Newton-Cotes formulas is summarized by **?**, p.265.

> "Even when the asymptotic rate of convergence for Gaussian quadrature is no better than the comparable Newton-Cotes formula, experience shows that Gaussian formulas often outperform the alternative Newton-Cotes formula [in terms of accuracy]."

Exercise 14.5. Approximate the integral of the function in Exercise **??** using Gaussian quadrature with $N = 3$, $(\omega_1, \omega_2, \omega_3, x_1, x_2, x_3)$. Use the class of polynomials $h_i(x) = x^i$. How does the accuracy of your approximated integral compare to the approximations from Exercise **??** and the true known value of the integral?

Exercise 14.6. Use the Python Gaussian quadrature command `scipy.integrate.quad` to numerically approximate the integral from Exercise **??**.

$$\int_{-10}^{10} g(x)dx \quad \text{where} \quad g(x) = 0.1x^4 - 1.5x^3 + 0.53x^2 + 2x + 1$$

How does the approximated integral from the `scipy.integrate.quad` command compare to the exact value of the function?

14.4 Monte Carlo Integration

High-dimensional integration is highly inefficient using the standard one-dimensional methods of Newton-Cotes and Gaussian quadrature. The method of choice in high-dimensional settings is known as Monte Carlo Integration.

In this section, we detail two types of Monte Carlo integration. The standard Monte Carlo simulation approach in Section **??** uses pseudorandomly generated draws from a uniform distribution over the domain of the function in order to approximate the weights and nodes for integration. The quasi-Monte Carlo approach in Section **??** uses elements of low-discrepancy sequences over the domain of the function in order to approximate the integral. Both methods have benefits and drawbacks.

14.4.1 Standard Monte Carlo integration

In the Newton-Cotes quadrature methods of approximating an integral, nodes and weights for the approximation $\sum_{n=1}^{N} \omega_n g(x_n)$ are chosen without much attention to the effect of the placement of these nodes or the levels of the weights on the accuracy of the approximation. Newton-Cotes methods are computationally fast, but lack in accuracy. Gaussian quadrature methods spend more computational time choosing "optimal" weights and nodes, but this

gives an accuracy payoff over Newton-Cotes formulas. Monte Carlo integration methods use the computationally fast method of drawing uniformly from the support of the variable of integration. These methods depend on a large number of draws to get high accuracy.[2] Although Monte Carlo integration methods do not converge as quickly as Gaussian quadrature methods for functions of one variable, they are especially valuable when integrating over functions of multiple variables.

Let $\Omega \subset \mathbb{R}^m$ be the domain of integration. Let $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$ be N uniform random draws from Ω. Then we can write the following approximation of the integral.

$$\int_\Omega g(\mathbf{x}) d\mathbf{x} \approx V \frac{1}{N} \sum_{n=1}^{N} g(\mathbf{x}_n) \quad \text{where} \quad V = \int_\Omega d\mathbf{x} \tag{14.12}$$

Equation (**??**) says we can approximate the integral of a function $g(\mathbf{x})$ on a domain Ω by taking the average of the evaluations of the function g at N random draws of the vector $\mathbf{x}_n$ multiplied by the volume of the domain.

An easy example of a univariate integral is $\int_0^1 x\, dx$ (here $g(\mathbf{x}) = x$). The Monte Carlo approximation formula for this integral is the following.

$$\int_0^1 x\, dx \approx V \frac{1}{N} \sum_{n=1}^{N} x_n = \frac{1}{N} \sum_{n=1}^{N} x_n \tag{14.13}$$

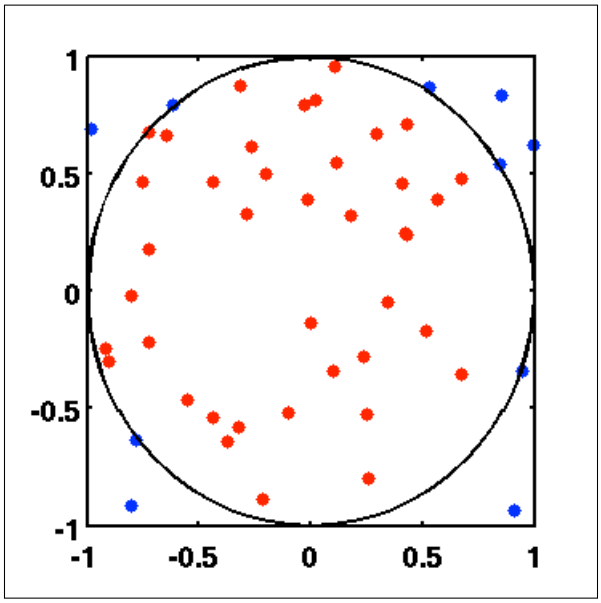
It is easy to see that the answer to the exact integral on the left-hand-side of (**??**) is 1/2. In the approximation on the right-hand-side of (**??**), V is the volume of the domain of $x \in [0, 1]$, which is 1. It is straightforward to see that the average of N draws from a uniform distribution between 0 and 1 will converge quickly to 1/2.

Exercise **??** lets you try your hand at coding a classic Monte Carlo integration approximation of the integral of a function of two variables to approximate the value of π. The area of a circle with radius $r = 1$ is π. A way to visualize the Monte Carlo approximation of the area of that circle, or π, is to enclose the circle in a square with sides of length 2, in

[2] **?**, pp. 309-311 spends significant time explaining that these methods are more correctly called "pseudo-Monte Carlo methods" because they make use of pseudorandom number generators, the use of which cannot invoke the law of large numbers or the central limit theorem. However, the biases introduced by pseudorandom number generators are rarely significant in practice.

which the x-axis goes from -1 to 1 and the y-axis goes from -1 to 1. The points in Figure **??** are the uniformly distributed random draws from $(x, y) \in [-1, 1] \times [-1, 1]$. Intuitively, the area of the circle is the fraction of the dots (red dots divided by total dots) that are inside the circle or on the boundary of the circle, multiplied by the area or volume of the square in which the circle lies.

Figure 14.1: Monte Carlo integral estimation of area of unit radius circle



Following the intuition of the previous paragraph and of Figure **??**, the exact area of the circle can be written as an integral of the indicator function of coordinate variables x and y in the following way.

$$\text{Area} = \int_{\Omega} g(x, y) dx\, dy = \pi$$
$$\text{where} \quad g(x, y) = \begin{cases} 1 & \text{if} \quad x^2 + y^2 \leq 1 \\ 0 & \text{else} \end{cases} \qquad \text{and} \quad \Omega = [-1, 1] \times [-1, 1] \tag{14.14}$$

The exact integral (**??**) for the area of a unit radius circle can be Monte Carlo approximated

using the form (**??**) resulting in the following function.

$$\int_{\Omega} g(x,y)dx\,dy \approx 4\frac{1}{N}\sum_{n=1}^{N} g\left(x_n, y_n\right) \tag{14.15}$$

Exercise 14.7. Use Monte Carlo integration to approximate the value of π. Define a function in `Python` that takes as arguments an anonymous function $g(\mathbf{x})$ of a vector of variables $\mathbf{x}$, the domain Ω of $\mathbf{x}$, and the number of random draws N and returns the Monte Carlo approximation of the integral $\int_{\Omega} g(\mathbf{x})d\mathbf{x}$. Let Ω be a generalized rectangle. In order to approximate π, let the functional form of the anonymous function be $g(x,y)$ from (**??**) with domain $\Omega = [-1,1] \times [-1,1]$. What is the smallest number of random draws N from Ω that matches the true value of π to the 4th decimal 3.1415? [Answers may differ slightly depending on your draws of (x_n, y_n) from the bivariate uniform distribution.]

14.4.2 Quasi-Monte Carlo integration

It is important to realize what Monte Carlo methods really are in practice. Due to the impracticality of generating truly "random"" sequences, Monte Carlo methods utilize pseudorandom sequences. Any sequence generated using a pseudorandom number generator will have a small amount of artificial correlation, and this problem is compounded in higher dimensions.

Quasi-Monte Carlo methods dispense with the attempt to create deterministic samples that mimic random samples, and instead embrace their deterministic character. **?**, ch. 9 defines quasi-Monte Carlo methods as sampling methods that do not rely on probabilistic ideas and pseudorandom sequences for constructing the sample and analyzing the estimate. Quasi-Monte Carlo methods use the same approximating function $\sum_{n=1}^{N} \omega_n g(x_n)$, but draw on number theory and Fourier analytic methods to create low-discrepancy sequences that are used as sample points. However, other than the selcetion of sample points, quasi-Monte Carlo integration proceeds in exactly the same way as standard Monte Carlo integration as detailed in the previous section.

Many different deterministic sequences can be used in quasi-Monte Carlo sampling. All of these "quasirandom" sequences strive for uniformity in a general sense. Hence it is useful

to have a precise way of measuring the degree to which a point set exhibits uniformity. If we have a uniformly distributed sequence $\mathbf{x}_n$ in the s-dimensional unit cube $I^s = [0,1)^s$, we would intuitively expect that every subset of I^s with the same volume would contain the same number of points. This idea is described precisely by the discrepancy of $\mathbf{x}_n$. We first define local discrepancy, and then define global discrepancy.

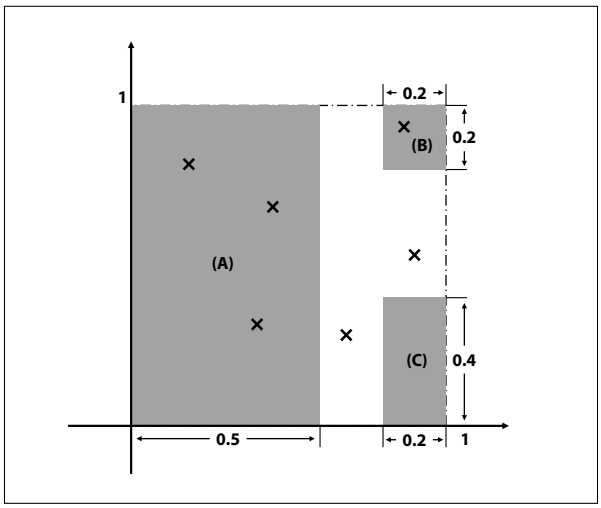
For N points $\{\mathbf{x}_n\}_{n=1}^N$ in I^s, $s \geq 0$, and $J \subseteq I^s$, the local discrepancy $D(J;N)$ is defined by

$$D(J;N) = S(J;N) - V(J)N, \tag{14.16}$$

where $V(J)$ is the volume of the subinterval J and $S(J;N)$ is the number of points from $\{\mathbf{x}_n\}_{n=1}^N$ that are in J. If the N points are uniformly distributed, then the local discrepancy should be very small for all J's.

Figure **??** shows a 2-dimensional unit cube $I^s = [0,1)^2$ with six points. Three subintervals A, B, and C are shaded. The local discrepancy of A, which contains 3 points, is calculated from Equation **??** as $D(A;6) = S(A;6) - V(A)6 = 3 - 6/2 = 0$. Similarly, the local discrepancies of B and C are 0.1267 and 0.8 respectively.

Figure 14.2: An illustration of discrepancy in two dimensions



A global concept of discrepancy is given by the star-discrepancy. We define the star-

discrepancy $\Delta(N)$ of N points by

$$\Delta(N) = \sup_J |D(J;N)|, \tag{14.17}$$

where the supremum is taken over all subsets J of the form $J = \prod_{i=1}^{s}[0, u_i)$. The star-discrepancy can be thought of as the worst-case local discrepancy, looking only at subintervals that have the origin as a corner.

A common class of sequences used in quasi-Monte Carlo sampling is equidistributed sequences. Equidistributed sequences sequences where the star-discrepancy $\Delta(N)$ tends to zero as N tends to infinity. In other words, in the limit the proportion of terms falling in any subinterval is proportional to the length of that interval.

There are a number of equidistributed sequences, and here we will provide some examples. By way of notation, let $p_1, p_2, \ldots$ denote the sequence of prime numbers $2, 3, 5, \ldots$, and let $\langle x \rangle$ represent the fractional part of x, that is $\langle x \rangle = x - \lfloor x \rfloor$. Table **??** contains formulas for a number of s-dimensional equidistributed sequences on $[0, 1)^s$. Figure **??** shows the first $10,000$ points for two-dimensional Weyl, Haber, Niederreiter, and Baker sequences.

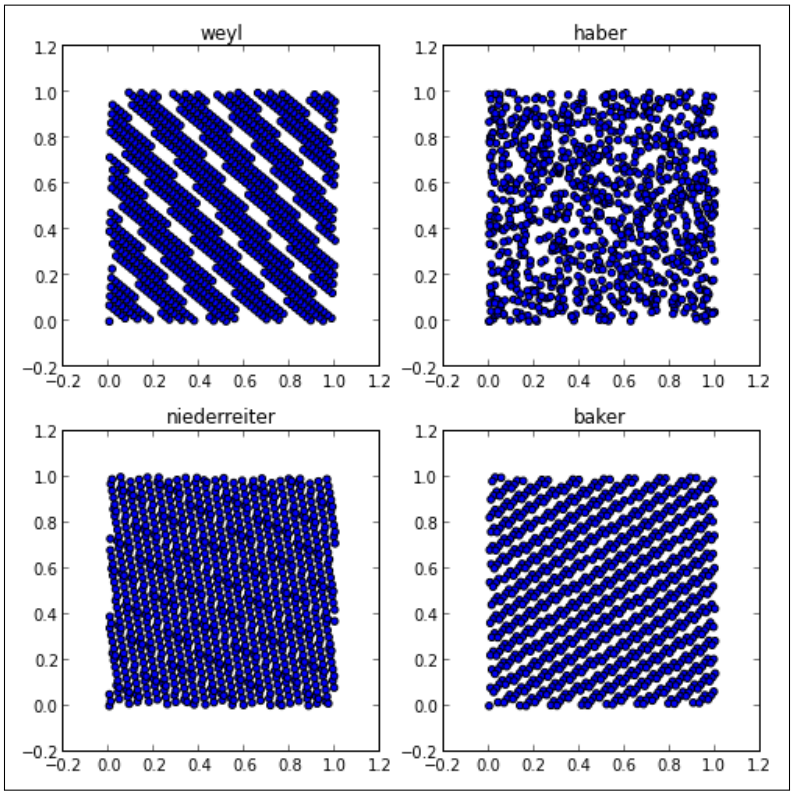
Table 14.1: Equidistributed Sequences in $\mathbb{R}^s$

Name of Sequence	Formula for $(x_1, x_2, \ldots, x_s)_n$
Weyl	$(\langle np_1^{1/2} \rangle, \ldots, \langle np_s^{1/2} \rangle)$
Haber	$\left(\left\langle \frac{n(n+1)}{2} p_1^{1/2} \right\rangle, \ldots, \left\langle \frac{n(n+1)}{2} p_s^{1/2} \right\rangle\right)$
Niederreiter	$\left(\left\langle n2^{1/(n+1)} \right\rangle, \ldots, \left\langle n2^{s/(n+1)} \right\rangle\right)$
Baker	$(\langle ne^{r_1} \rangle, \ldots, \langle ne^{r_s} \rangle)$, r_j rational and distinct

A key distinction between quasirandom sequences and pseudorandom sequences is that quasirandom sequences do not "look like" random numbers. As can be seen in Figures **??**, they generally display quite obvious patterns. From the outset, quasirandom sequences are chosen so as to have low discrepancy, and are not encumbered by any other requirements of random numbers.

However, equidistribution is a rather weak criterion to express the idea that a sequence is uniform. Even though the discrepancy approaches zero as n approaches infinity, the Weyl sequence in Figure **??** shows how there will often be large gaps for small n. An alternative

Figure 14.3: Four two-dimensional equidistributed sequences with $n = 10,000$



approach is to use low-discrepancy sequeces

The goal of a low-discrepancy sequence is to minimize the star-discrepancy of every subsequence. In contrast to equidistributed sequences, the algorithms to generate these sequences take into account the total number of points desired so that maximum uniformity is achieved for every subsequence, and not just in the limit.

Halton sequences describe a class of low-discrepancy multidimensional sequences that fill the interval $[0, 1)$. To construct a Halton sequence, begin with a consecutive sequence of positive integers of length N, for example $n = 1, 2, \ldots, N$. Now choose any prime number p and convert each integer n into its representation in the base p number system. For multiple dimensions, repeat the process with a different prime p. Then reflect the base p representation of each integer about the decimal point to obtain a number in the interval $[0, 1)$.

Many other techniques exist for creating low-discrepancy sequences. Fauré sequences are permutations of Halton sequences. Sobol sequences are a reordering of Halton sequences.

Other methods include (t, m, s)-Nets and the method of good lattice points. **?** is a good resource on low-discrepancy sequences in quasi-Monte Carlo methods.

Quasi-Monte Carlo methods do far better asymptotically than any Monte Carlo method for many problems. With N points in s dimensions, quasi-Monte Carlo techniques have a worst-case convergence rate of $O\left(\frac{(\log N)^s}{N}\right)$ as opposed to $O\left(\frac{1}{\sqrt{N}}\right)$ for standard Monte Carlo techniques. However, standard Monte Carlo integration is easier to implement properly and is generally sufficient for most purposes.

Exercise 14.8. Define a function in `Python` that returns the n-th element of a d-dimensional equidistributed sequence. It should have support for the four sequences in Table **??**.

Exercise 14.9. Repeat exercise **??** to approximate the value of π, this time using quasi-Monte Carlo integration. Follow exactly the same procedure as before, except instead of using random draws from the domain use the function in exercise **??** to select the sample points. You will need to appropriately scale the sequences. Compare the rates of convergence.

14.5 Sparse Grids

Sparse grid interpolation is a method of approximating functions with many dimensions. A direct way of representing multidimensional functions is to use a full grid, such as in Newton-Cotes quadrature. However, using this method the number of grid points that have to be computed depends exponentially on the number of dimensions. Because of this curse of dimensionality, processing functions of beyond five or six dimensions becomes intractable using the techniques of Sections **??** and **??**. While Monte Carlo methods perform reasonably well in multidimensional integration, sparse grids remain the most efficient.

The sparse grid method selects the nodes of integration by a special truncation of the tensor product expansion of a one-dimensional multilevel selection of nodes.

To construct a sparse grid, we first select a series of one-dimensional quadrature formulas indexed by l for a univariate function f, and write it as

$$Q_l^{(1)} f = \sum_{i=1}^{N_l} w_{li} f(x_{li}). \tag{14.18}$$

For example, the nodes and weights of this initial quadrature formula series could be based on the trapezoid rule or Simpson's rule with N_l nodes.

Now define the difference formulas by

$$\Delta_k^{(1)} f = (Q_k^{(1)} - Q_{k-1}^{(1)}) f \tag{14.19}$$

where $Q_0^{(1)} f = 0$. Note that the differences $\Delta_k^{(1)} f$ are just univariate quadrature formulas.

For a given level $l \in \mathbb{N}$, the sparse gride integration approximation for a d-dimensional function f is given by

$$Q_l^{(d)} = \sum_{\|\mathbf{k}\| \leq l+d-1} (\Delta_{k_1}^{(1)} \otimes \ldots \otimes \Delta_{k_d}^{(1)}) f. \tag{14.20}$$

Using the quadrature rule in **??**, every possible tensor product of the difference formulas is considered, but only those whose sum of indices is smaller than the constant $l + d - 1$ are used.

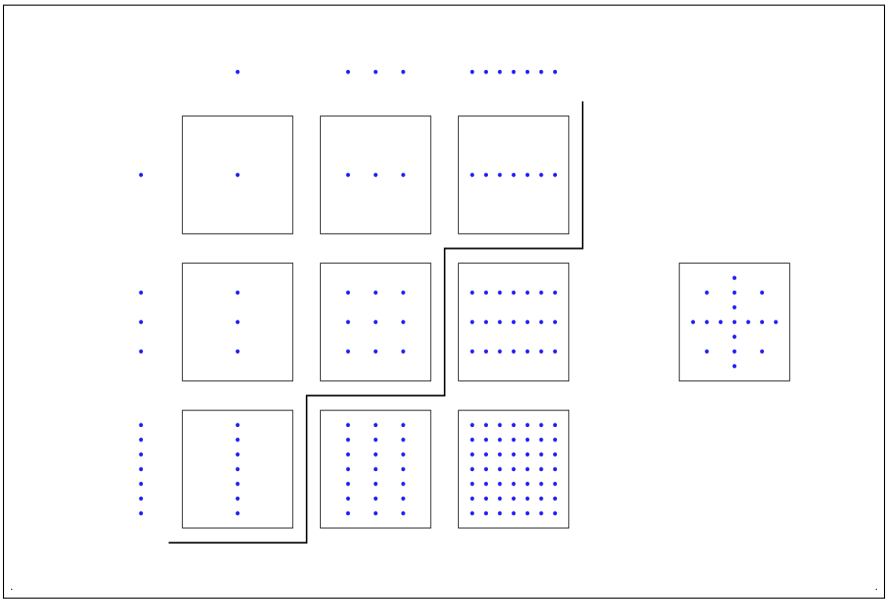
The selcetion of nodes in a two-dimensional sparse grid of level $l = 3$ using the trapezoid rule as the intial formula is visualized in **??**. Along the top and left are the one-dimensional grid points for $l = 1, 2, 3$ in the x- and y-directions. These points are used to create the corresponding product grids $\Delta_{k_1} \otimes \Delta_{k_2}$ for $1 \leq k_1, k_2 \leq 3$. Because the grid points used in the trapezoid rule are nested, many of the grid points are cancelled out, and the only ones that remain are formed from a union of the grids along the diagonal as indicated by the black line. The resulting sparse grid $Q_3^{(2)}$ is shown on the right.

14.6 Discrete Markov Approximation of Continuous AR(1) Process

Suppose you have a random shock z_t in your model that has some persistence according to the following AR(1) process.

$$z_{t+1} = \rho z_t + (1 - \rho)\mu + \varepsilon_{t+1} \quad \text{where} \quad \varepsilon_t \sim N(0, \sigma) \quad \text{and} \quad \rho \in [0, 1) \tag{14.21}$$

Figure 14.4: Visualization of sparse grid quadrature using the trapezoid rule.



The expected value of z_{t+1} is conditional on the current realization of the shock $E[z_{t+1}|z_t] = \rho z_t + (1-\rho)\mu$ but the variance of z_{t+1} is unconditional $Var[z_{t+1}] = \sigma^2$. The AR(1) process in (**??**) generates a variable that fluctuates around its mean μ, and the expected value of the variable tomorrow $E[z_{t+1}|z_t]$ is some convex combination of the variable today z_t and the mean μ.

Typical examples of these types of shocks in economics are shocks to ability, health status, and productivity shocks—all of which exhibit persistence or dependence on recent values. If the shock must be strictly positive, as is the case with productivity shocks, the variable z_t is simply exponentiated.

$$Y_t = A_t K_t^{\alpha} L_t^{1-\alpha} \quad \text{where} \quad A_t = e^{z_t} \tag{14.22}$$

Notice that the variable A_t is lognormally distributed $A_t \sim LN\big(\rho z_{t-1} + (1-\rho)\mu, \sigma\big)$ because $\log(A_t) = z_t$ and $z_t \sim N\big(\rho z_{t-1} + (1-\rho)\mu, \sigma\big)$. You made a discretized approximation of the i.i.d. (no persistence) version of this distribution in Exercise **??** and estimated average income in the U.S. using it in Exercise **??**.

? describe a quadrature-based method for producing efficient nodes and probabilities of

a discrete first-order Markov process to approximate a continuous AR(1) random variable.[3] A classic example of where this discretization is extremely valuable is in the stochastic intertemporal Euler equation (**??**) from Section **??**.

$$\begin{aligned} & u'(c_t) = \beta E_{z_{t+1}|z_t}\Big[(1 + r_{t+1} - \delta)u'(c_{t+1})\Big] \\ \Rightarrow \quad & u'(c_t) = \beta \int_a^b \Big(1 + r_{t+1}(z_{t+1}) - \delta\Big)u'\Big(c_{t+1}(z_{t+1})\Big) f(z_{t+1}|z_t)dz_{t+1} \end{aligned} \tag{??}$$

The expectation on the right-hand-side of (**??**) is over z_{t+1} given z_t, where z_{t+1} is the AR(1) process described in (**??**). One of the most common nonlinear solution techniques for the functional equations of the model characterized by (**??**) is value function iteration on the following recursive Bellman equation.

$$V(k, z) = \max_{k'} \; u(k, z, k') + \beta E_{z'|z}\big[V(k', z')\big] \tag{14.23}$$

The expectation on the right-hand-side of the Bellman equation (**??**) is simply an integral of the form $E_{z'|z}\big[V(k', z')\big] = \int_{z'} V(k', z')f(z'|z)dz'$. However, it is difficult to use standard Gaussian quadrature or Monte Carlo integration methods because the value function $V(k', z')$ is often only known at a few points.

One solution to this problem is to interpolate or fit some continuous function $\tilde{V}(k', z')$ to the known points of $V(k', z')$ and then use Gaussian quadrature or Monte Carlo integration to approximate the integral $\int_{z'} \tilde{V}(k', z')f(z'|z)dz'$. **?** and **?**, p. 237 find that the errors in the extrapolated values of the interpolated function $\tilde{V}$ beyond the bounds of the known points of V cause the solution to be less accurate than using the Tauchen-Hussey method of approximating $f(z'|z)$ with a discrete first order Markov process.

[3] **?** details a simpler non-quadrature based method for producing efficient nodes and probabilities of a discrete first-order Markov process to approximate a continuous AR(1) random variable.