Tema 6
Integración y diferenciación numéricas

Randall Romero Aguilar, PhD

Universidad de Costa Rica
SP6534 - Economía Computacional

I Semestre 2020

Última actualización: 23 de abril de 2020

UCR
UNIVERSIDAD DE COSTA RICA

ESCUELA de ECONOMÍA
UNIVERSIDAD DE COSTA RICA

Tabla de contenidos

1. Introducción

2. Área bajo una curva

3. Calculando valores esperados

4. Simulación de Montecarlo

5. Integración cuasi Montecarlo

6. Diferenciación numérica

1. Introducción

Un problema de estadística

£Cuál es la probabilidad de que una variable aleatoria normal estándar $\tilde{Z}$ tendrá un valor menor o igual a z?

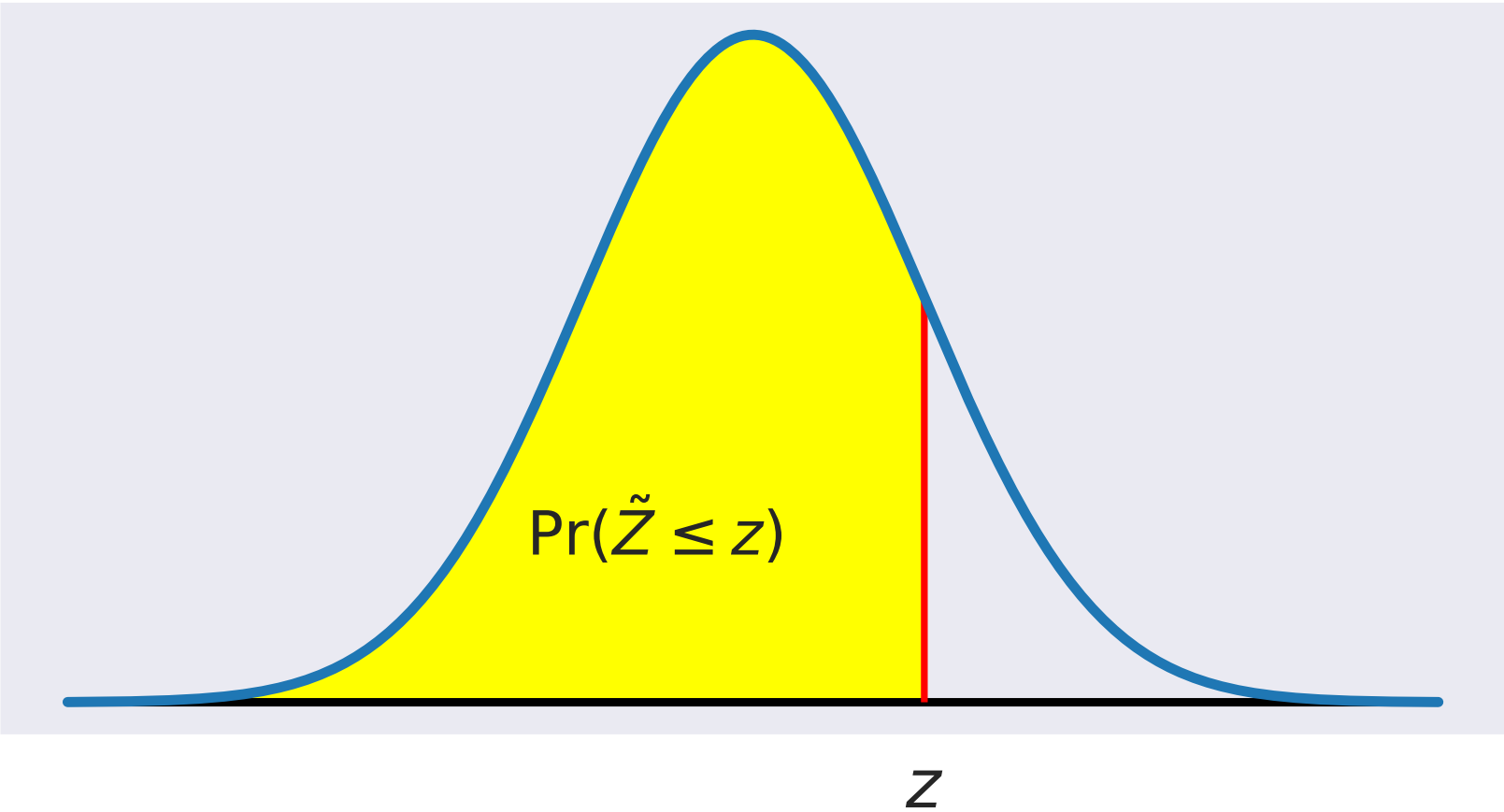

Figura 6.1: Densidad normal estándar

- La probabilidad es el área bajo la función de densidad de probabilidad normal estándar a la izquierda de z:

$$\Pr(\tilde{Z} \leq z) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{z} \exp\left(-\frac{t^2}{2}\right) \, dt$$

- Evalúe esta integral para $z = 1$.
- £Está teniendo dificultades?
- No es de sorprenderse, porque la integral carece de forma cerrada.
- Revise la tabla al final de su libro de estadística.
- Dice que la probabilidad es 0.841.
- £Cómo obtuvo el autor este valor?

Un problema de riesgo

- La utilidad del ingreso y para un agente exhibe aversión absoluta al riesgo constante $\alpha > 0$:

$$u(y) = -\exp(-\alpha y).$$

- El agente enfrenta un ingreso incierto $\tilde{y}$ que está distribuido log-normalmente con log-media μ y log-varianza σ^2.
- £Aceptaría este agente un ingreso cierto y^* en lugar de su ingreso incierto $\tilde{y}$?
- De acuerdo con la teoría de utilidad esperada, sí, siempre que

$$u(y^*) > Eu(\tilde{y}).$$

- Para contestar esta pregunta de manera definitiva, debemos evaluar la utilidad esperada del agente con el ingreso incierto:

$$Eu(\tilde{y}) = -\frac{1}{\sigma\sqrt{2\pi}} \int_0^{\infty} \frac{1}{y} \exp\left(-\frac{(\ln(y)-\mu)^2}{2\sigma^2} - \alpha y\right) dy.$$

- Evalúe esta expresión cuando $\mu = 0$, $\sigma^2 = 0.1$, y $\alpha = 2$.
- £Está teniendo dificultades?
- No es de sorprenderse, porque esta integral carece de forma cerrada.

Cuadratura numérica

- En economía, encontramos dos tipos de problemas de integración:
  - Evaluar el área bajo una curva, como en el ejemplo de estadística.
  - Evaluar la experanza de una función de una variable aleatoria, como en el ejemplo de riesgo.
- En muchas aplicaciones, la integral definida carece de una expresión equivalente de forma cerrada o bien es analíticamente intratable.
- Sin embargo, tales integrales por lo general pueden ser fácil y precisamente evaluadas numéricamente usando métodos de cuadratura .

- Discutimos tres clases de métodos de cuadratura:
  - Reglas de Newton-Cotes
  - Cuadratura de Gauss
  - Simulación Montecarlo
- Todos estos métodos tienen una cosa en común: la integral definida es aproximada usando una suma ponderada de valores de una función en nodos prescritos, una tarea simple en una computadora.
- Los métodos difieren sólo en cómo se escogen las ponderaciones y los nodos.

2. Área bajo una curva

Consideremos el problema de encontrar el área bajo la curva de una función real f en el intervalo $[a, b]$:

$$A = \int_a^b f(x)\, dx$$

3 métodos de cuadratura usualmente usados para calcular áreas:

- regla del trapezoide
- regla de Simpson
- regla de Gauss-Legendre

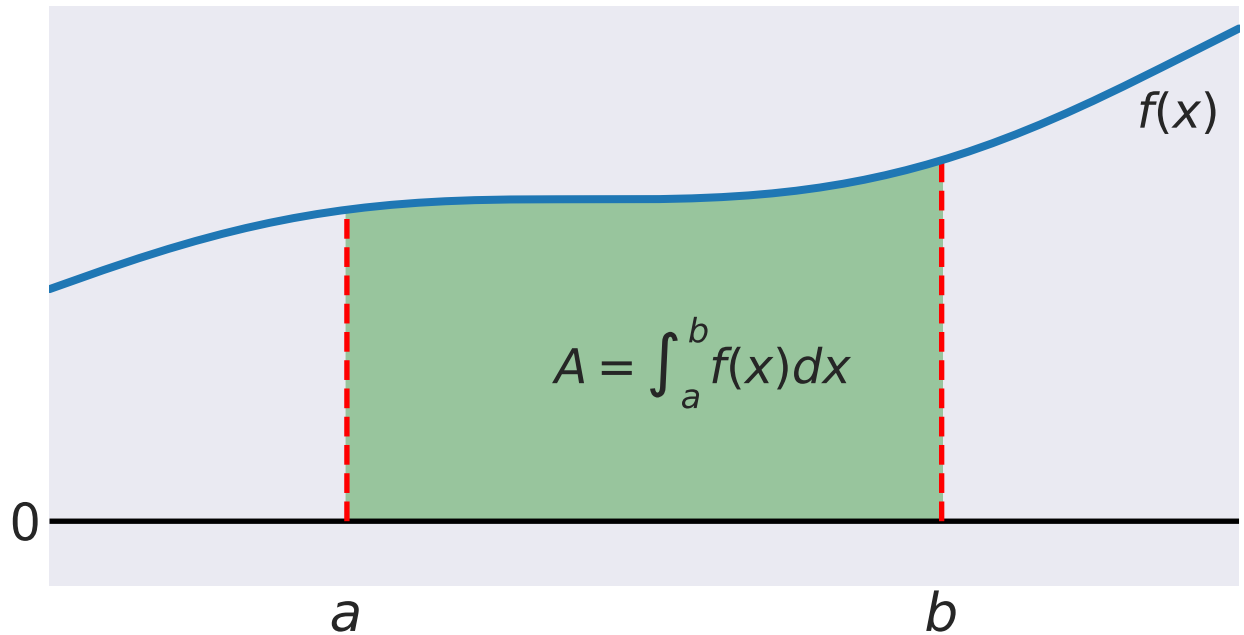

Figura 6.2: área bajo la Curva

Regla del trapezoide

- La regla de trapezoide aproxima el área bajo la función f como el área bajo una aproximación lineal por partes de f, denotada aquí $\tilde{f}$.
- Particionamos el intervalo $[a, b]$ en n subintervalos de igual longitud $h = (b - a)/n$ definidos por los nodos $x_i = a + ih$, $i = 0, 1, \ldots, n$.
- Calculamos el valor de la función $y_i = f(x_i)$ en los nodos.
- Construimos $\tilde{f}$ conectando puntos sucesivos (x_i, y_i) en el gráfico de f con líneas rectas.
- El área bajo $\tilde{f}$ es una series de trapezoides, que el dan su nombre a la regla del trapezoide.

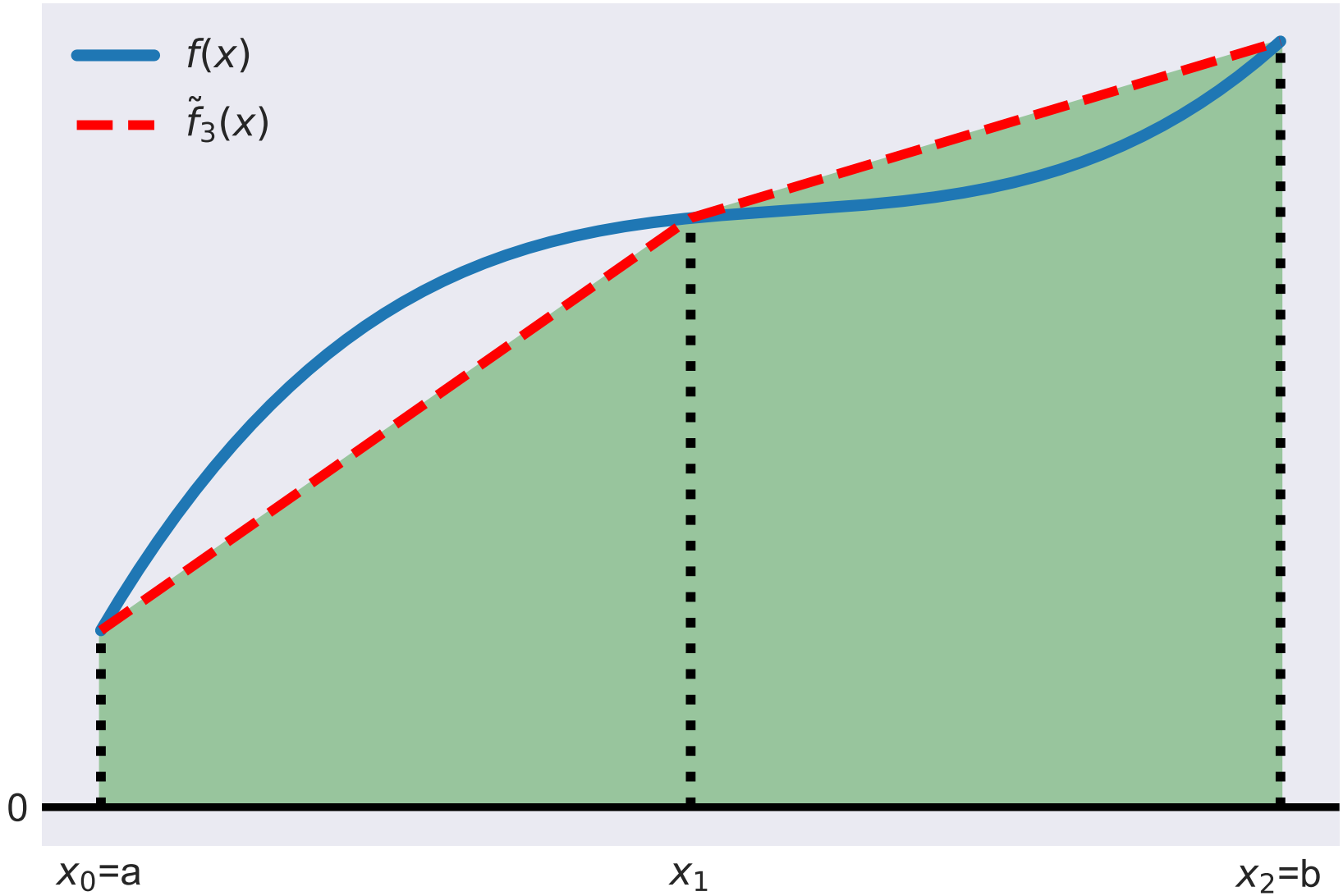

Figura 6.3: Regla del trapezoide, $n = 2$

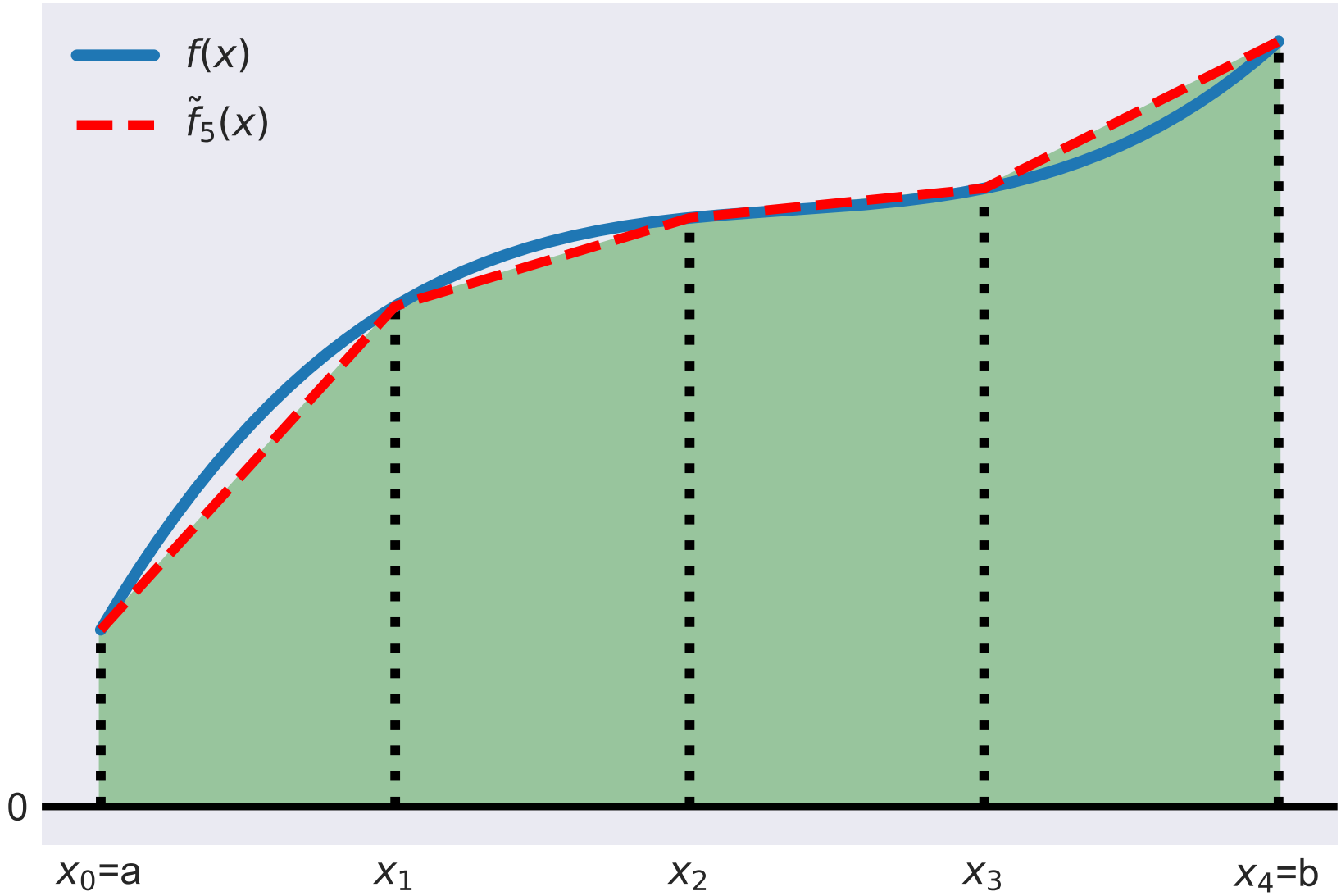

Figura 6.4: Regla del trapezoide, $n = 4$

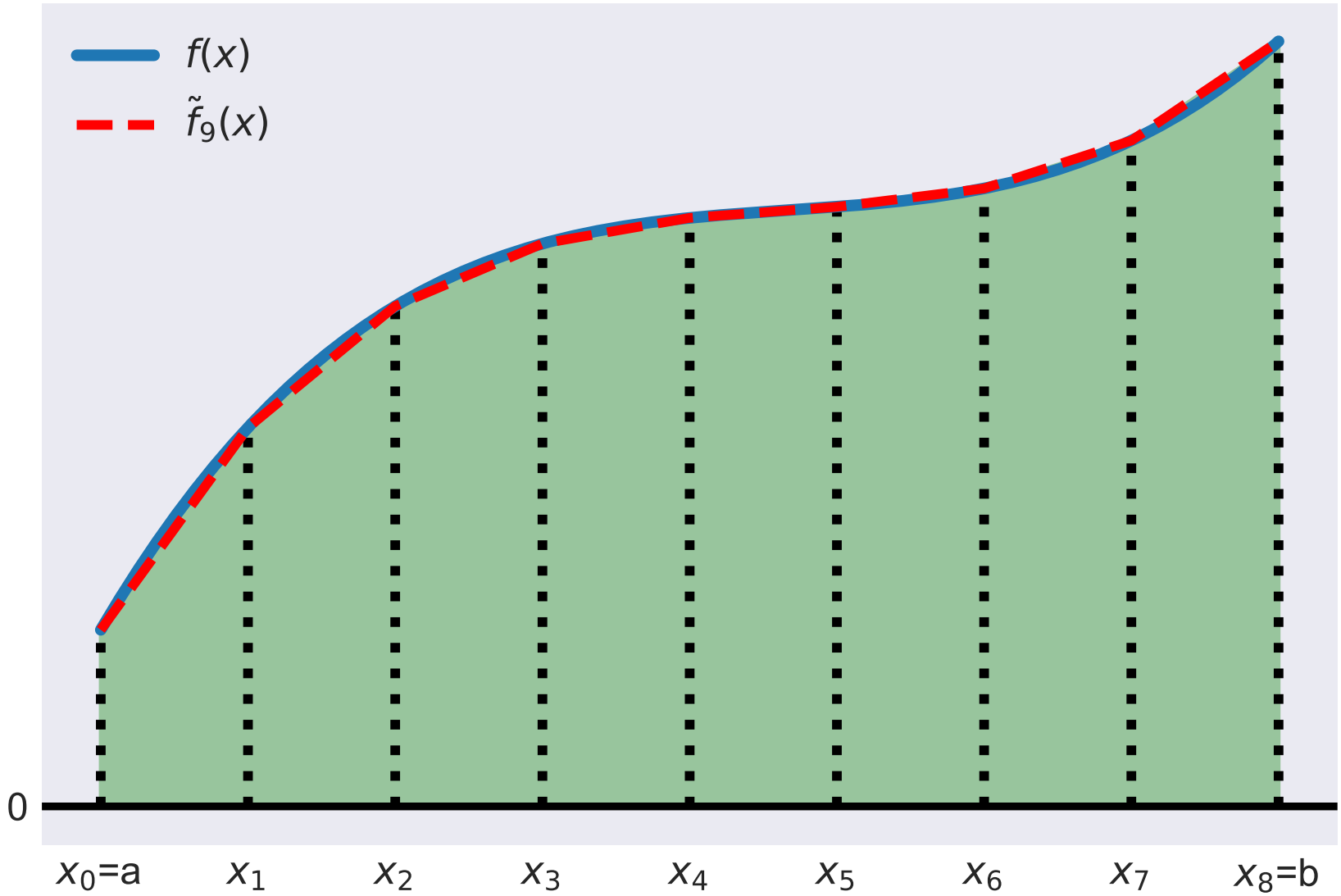

Figura 6.5: Regla del trapezoide, $n = 8$

- El área del i-ésimo trapezoide es:

$$\int_{x_{i-1}}^{x_i} \tilde{f}(x)\, dx = \frac{h}{2}[f(x_{i-1}) + f(x_i)].$$

- Sumando las áreas de todos los n trapezoides da por resultado la regla del trapezoide:

$$\int_a^b f(x)\, dx \approx \sum_{i=0}^{n} w_i\, f(x_i)$$

donde

$$w_i = \begin{cases} h/2 & i = 0, i = n \\ h & \text{en caso contrario} \end{cases}$$

- La regla del trapezoide es simple y robusta.
- Si f es suave, la regla del trapezoide alcanza un error de aproximación error proporcional a h^2.
- Duplicando el número de nodos reducirá el error de aproximación por un factor de cuatro.

Regla de Simpson

- La regla de Simpson aproxima el área bajo una función f como el área bajo una aproximación cuadrática por partes de f, denotada aquí $\tilde{f}$.
- Particionamos el intervalo $[a, b]$ en n subintervalos de igual longitud $h = (b-a)/n$ definidos por los nodos $x_i = a + ih$, $i = 0, 1, \ldots, n$, n es par.
- Calculamos el valor de la función $y_i = f(x_i)$ en los nodos.
- Formamos una aproximación cuadrática por partes $\tilde{f}$ de f interpolando tripletas sucesivas de puntos del gráfico (x_{i-2}, y_{i-2}), (x_{i-1}, y_{i-1}), y (x_i, y_i), $i = 2, 4, \ldots, n$, con funciones cuadráticas.

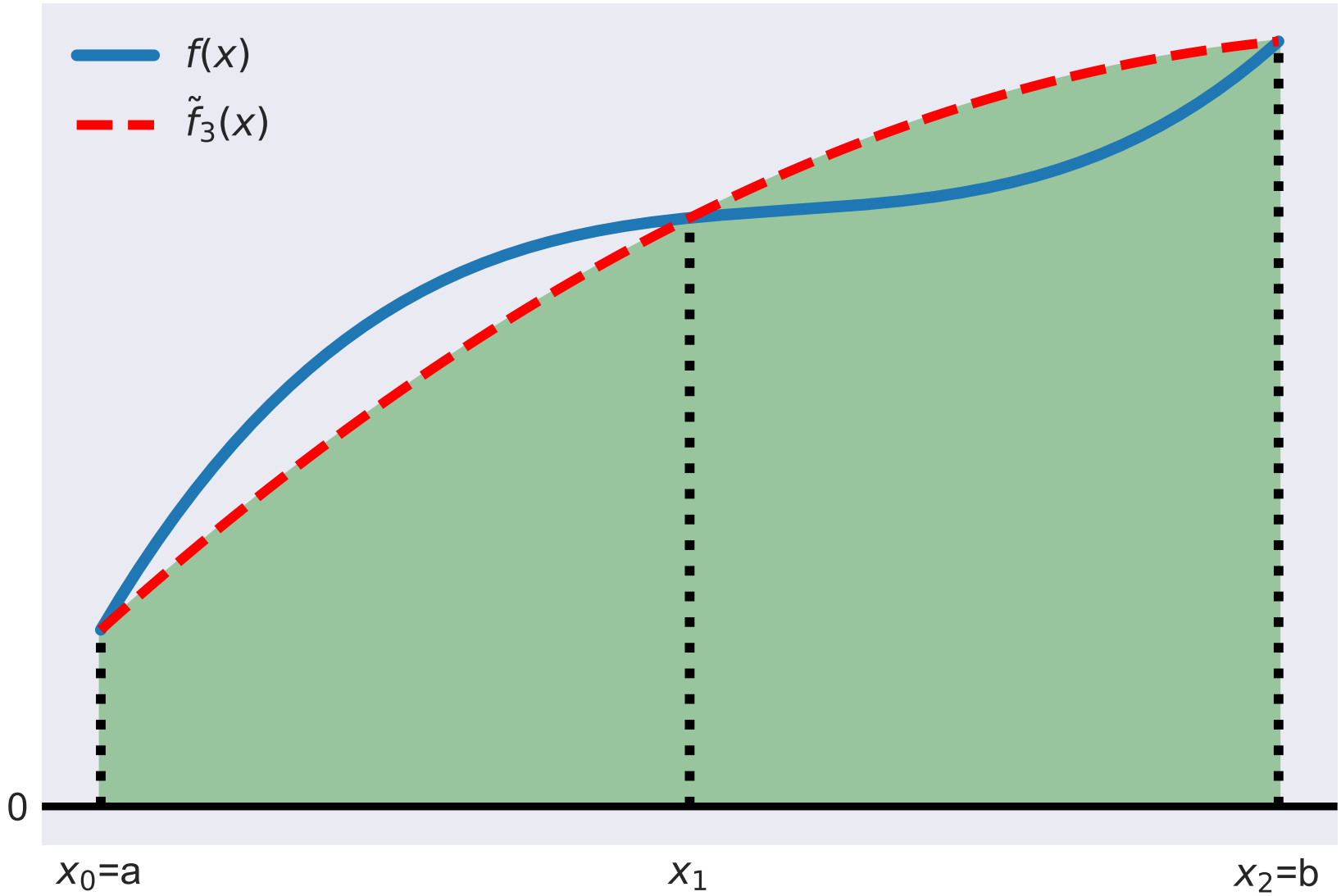

Figura 6.6: Regla de Simpson, $n = 2$

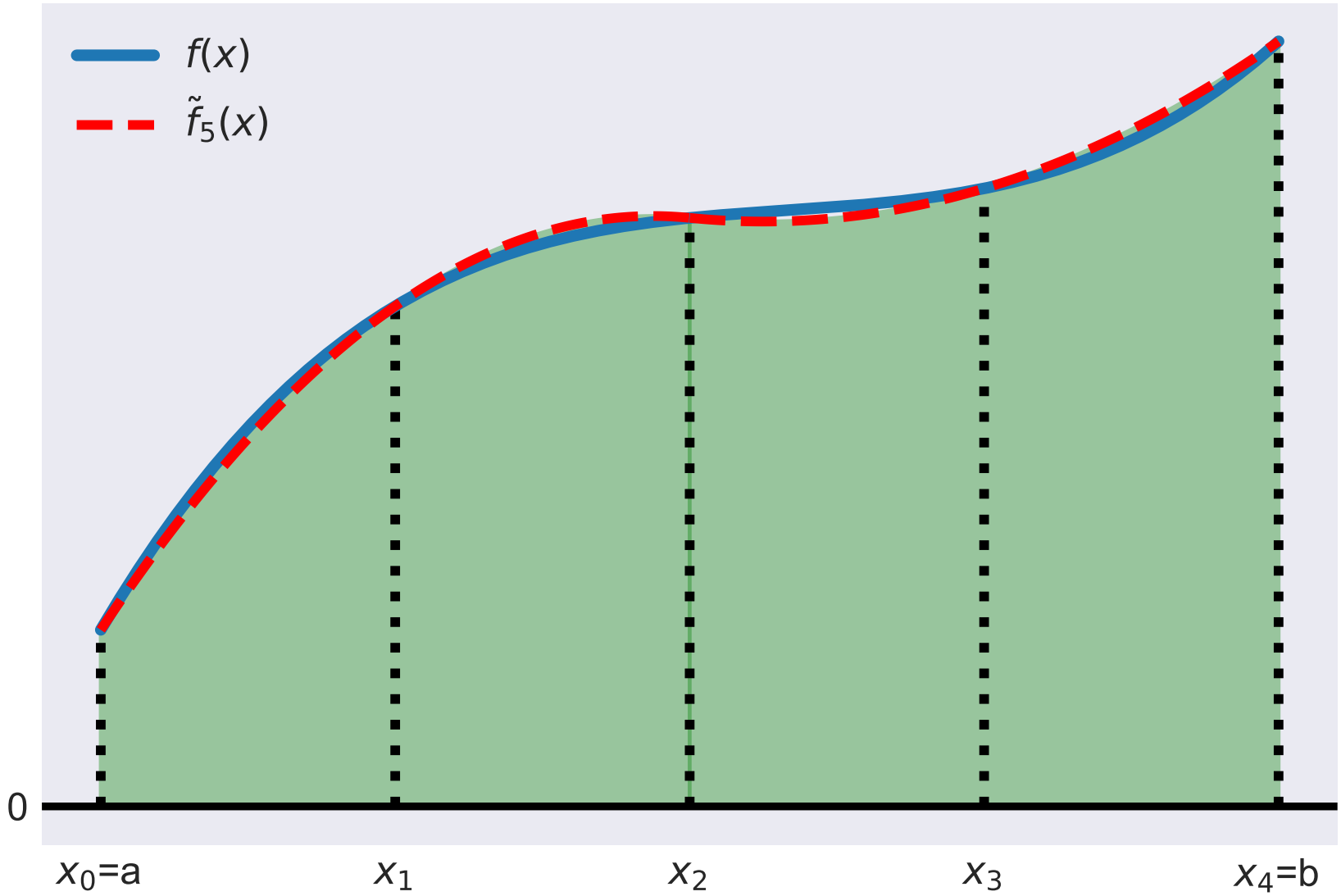

Figura 6.7: Regla de Simpson, $n = 4$

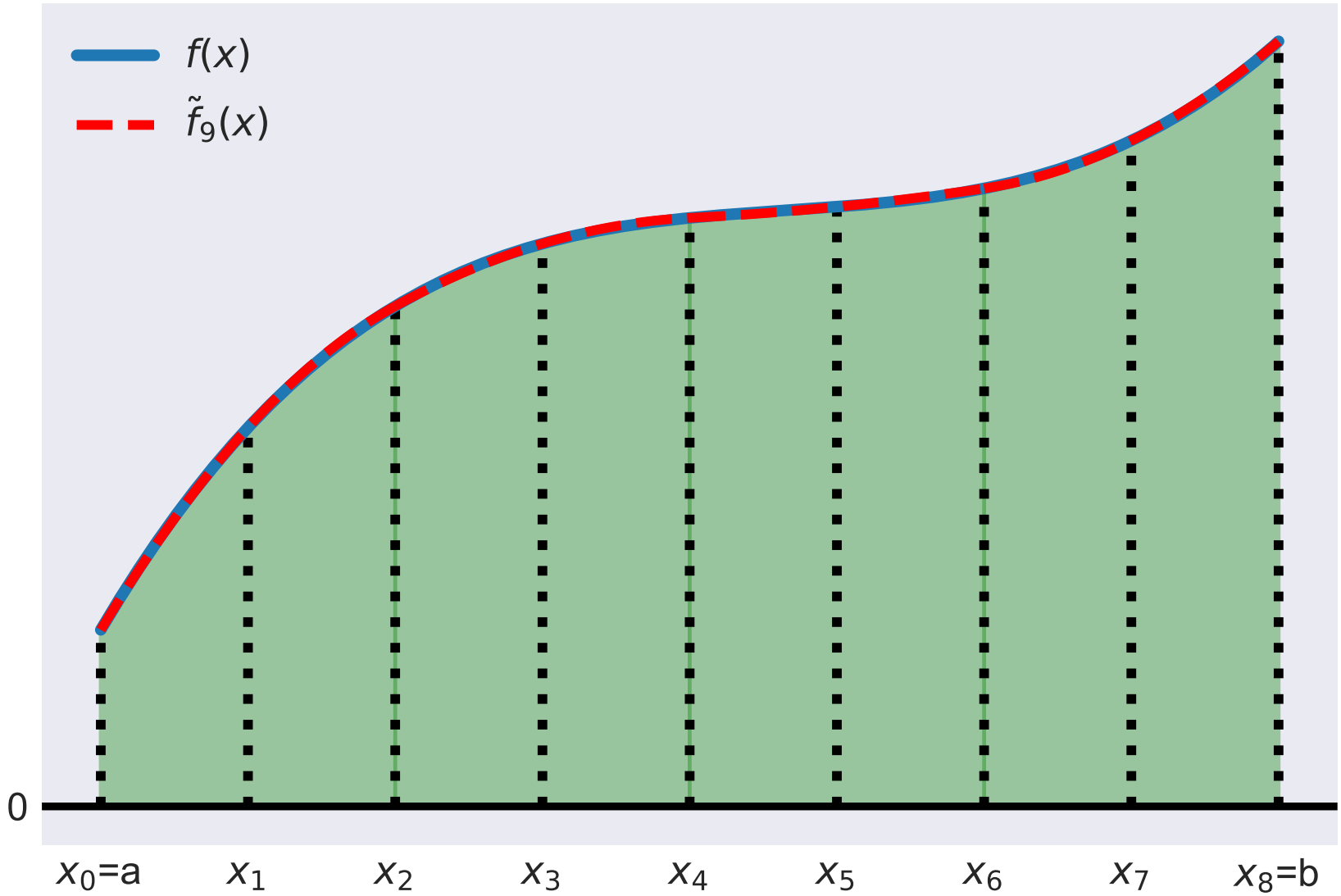

Figura 6.8: Regla de Simpson, $n = 8$

- The área bajo $\tilde{f}$ entre los subintervalos $i-1$ y i, para $i = 2, 4, \ldots, n$, es:

$$\int_{x_{i-2}}^{x_i} \tilde{f}(x)\, dx = \frac{h}{3}\left(f(x_{i-2}) + 4f(x_{i-1}) + f(x_i)\right).$$

- Sumando todos los pares sucesivos de subintervalos resulta en la regla de Simpson:

$$\int_a^b f(x)\, dx \approx \sum_{i=0}^{n} w_i f(x_i)$$

donde

$$w_i = \begin{cases} h/3 & i = 0, i = n \\ 4h/3 & 0 < i < n, i \text{ par} \\ 2h/3 & 0 < i < n, i \text{ impar} \end{cases}$$

- Si f es suave, la regla de Simpson alcanza un error de aproximación proporcional a h^4, el cuadrado del error de la regla del trapezoide.
- Duplicando el número de nodos reducirá el error de aproximación por un factor de dieciséis.
- La regla de Simpson es preferida a la regla del trapezoide porque es casi tan simple, pero mucho más precisa.

Reglas Newton-Cotes

- Las reglas del trapezoide y de Simpson son ejemplos de reglas Newton-Cotes, las cuales reemplazan el integrando con un polinomio por partes de grado bajo.
- Podemos definir reglas Newton-Cotes basadas en polinomios por partes de grados 3 o mayores, pero hacerlo no es práctico.
- En la práctica, no es posible determinar a priori cuántos nodos son necesarios para alcanzar un nivel deseado de precisión.
- Existen también reglas Newton-Cotes adaptativas que aumentan el número de nodos, sistemáticamente concentrando nuevos nodos donde el integrando es más irregular, hasta que las aproximaciones de la integral converjan.

Cuadratura Gauss-Legendre

- La cuadratura Gauss-Legendre emplea una lógica distinta al cálculo del área bajo la curve de f en el intervalo $[a, b]$.
- Específicamente, los n nodos de cuadratura x_i y las n ponderaciones de cuadratura w_i son escogidas para integrar con exactitud polinomios de grado $2n-1$ o menos.
- Este requerimiento impone las $2n$ condiciones

$$\int_a^b x^k \, dx = \sum_{i=1}^{n} w_i x_i^k, \qquad k = 0, \ldots, 2n-1,$$

que pueden ser resueltas para los n nodos y n ponderaciones usando métodos de ecuaciones no lineales.

- A diferencia de las reglas Newton-Cotes rules, los nodos Gauss-Legendre no están uniformemente espaciados y no incluyen los límites de integración.
- Por ejemplo, los nodos Gauss-Legendre de orden 4 para el intervalo $[0, 1]$ son 0.069, 0.330, 0.670, y 0.931, y las ponderaciones correspondientes son 0.174, 0.326, 0.326, y 0.174.

La cuadratura Gauss-Legendre supera a las reglas Newton-Cotes rules si el integrando f es suave, pero de lo contrario puede desempeñarse igual o peor.
Por ejemplo, consideremos integrar e^{-x} y $\sqrt{|x|}$ en $[-1, 1]$.

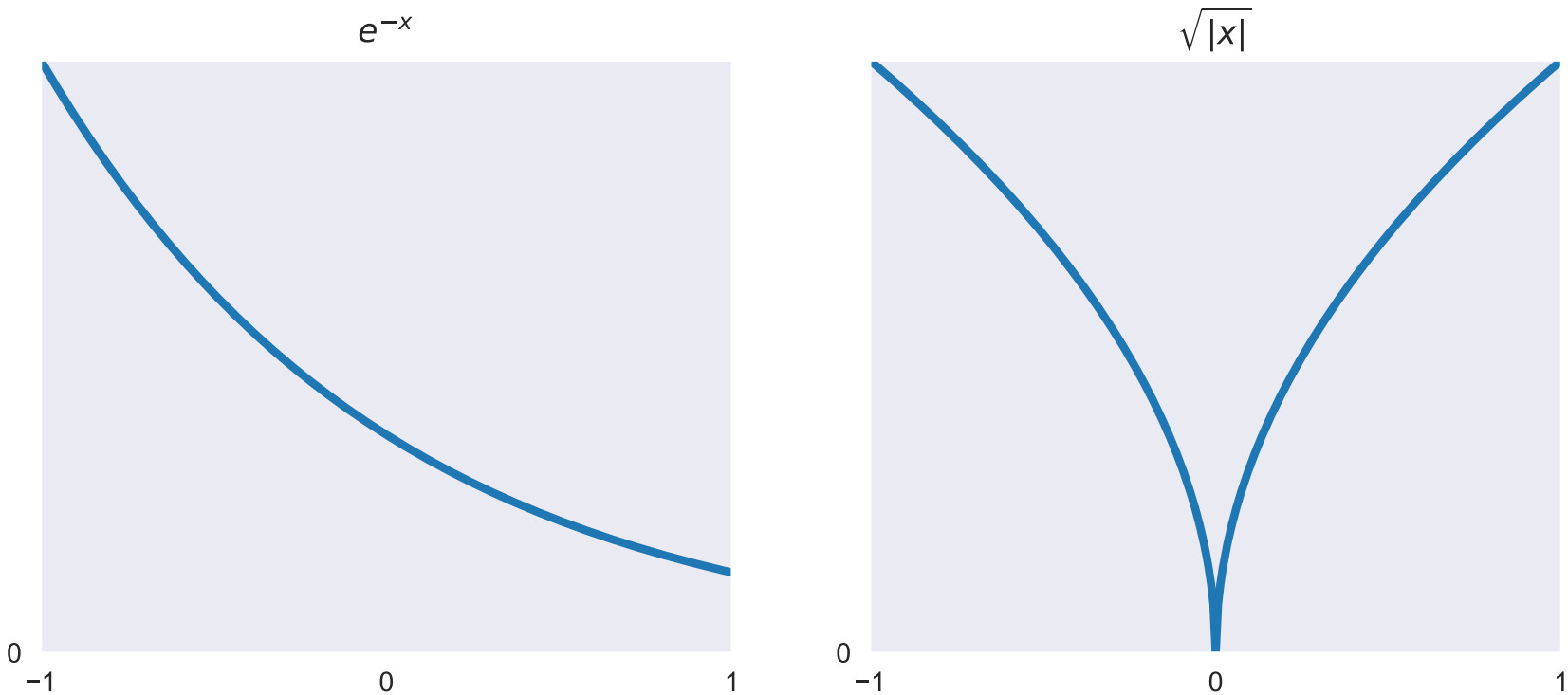

Figura 6.9: Una función suave y otro no-suave

Integral	n nodos	Trapezoide	Regla Simpson	Gauss-Legendre
$\int_{-1}^{1} e^{-x} dx$	5	-1.7	-3.5	-9.5
	11	-2.5	-5.1	-14.3
	21	-3.1	-6.3	-14.7
	31	-3.4	-7.0	-inf
$\int_{-1}^{1} \sqrt{\lvert x \rvert} dx$	5	-1.0	-1.4	-0.9
	11	-1.6	-1.3	-1.4
	21	-2.0	-2.4	-1.8
	31	-2.3	-2.1	-2.0

Cuadro 1: Log10 de los errores relativos de aproximación de la integral definida

Herramientas de CompEcon

- Las funciones de CompEcon `qnwtrap`, `qnwsimp` y `qnwlege` generan los nodos y ponderaciones de las reglas del trapezoide, de Simpson, y de Gauss-Legendre, de esta manera:

```
x, w = qnwtrap(n,a,b)
x, w = qnwsimp(n,a,b)
x, w = qnwlege(n,a,b)
```

- Insumos: `n` el número de nodos, `a` el límite izquierdo de integración, y `b` el límite derecho de integración.
- Resultado: `x` y `w`, los `n`-vectores nodos y ponderaciones de cuadratura, respectivamente.

Ejemplo 1:
Calculando una probabilidad

Para calcular la probabilidad de que una variable aleatoria normal estándar sea menor a 1, ejecutamos el código

```
from numpy import exp, sqrt, pi
from compecon import qnwsimp

f = lambda x: exp(-x**2/2) / sqrt(2*pi)
x, w = qnwsimp(11, 0, 1)
prob = 0.5 + w.dot(f(x))
```

El valor calculado, 0.8413, es correcto hasta cuatro dígitos significativos.

Así es como se calculan las tablas en los libros de estadística.

Ejemplo 2:
Excedente del consumidor

- Supongamos que la demanda por una mercancía está dada por

$$q(p) = 0.15p^{-1.25}$$

y el precio baja de $p_1 = 0.7$ a $p_2 = 0.3$.

- Entonces la ganancia en el excedente del consumidor

$$\int_{p_2}^{p_1} q(p)\, dp,$$

puede calcularse ejecutando

```
from compecon import qnwlege

q = lambda x: 0.15*p**(-1.25)
p, w = qnwlege(11, 0.3, 0.7)
change = w.dot(q(p))
```

- El valor calculado, 0.1548, es correcto hasta cuatro dígitos significativos.

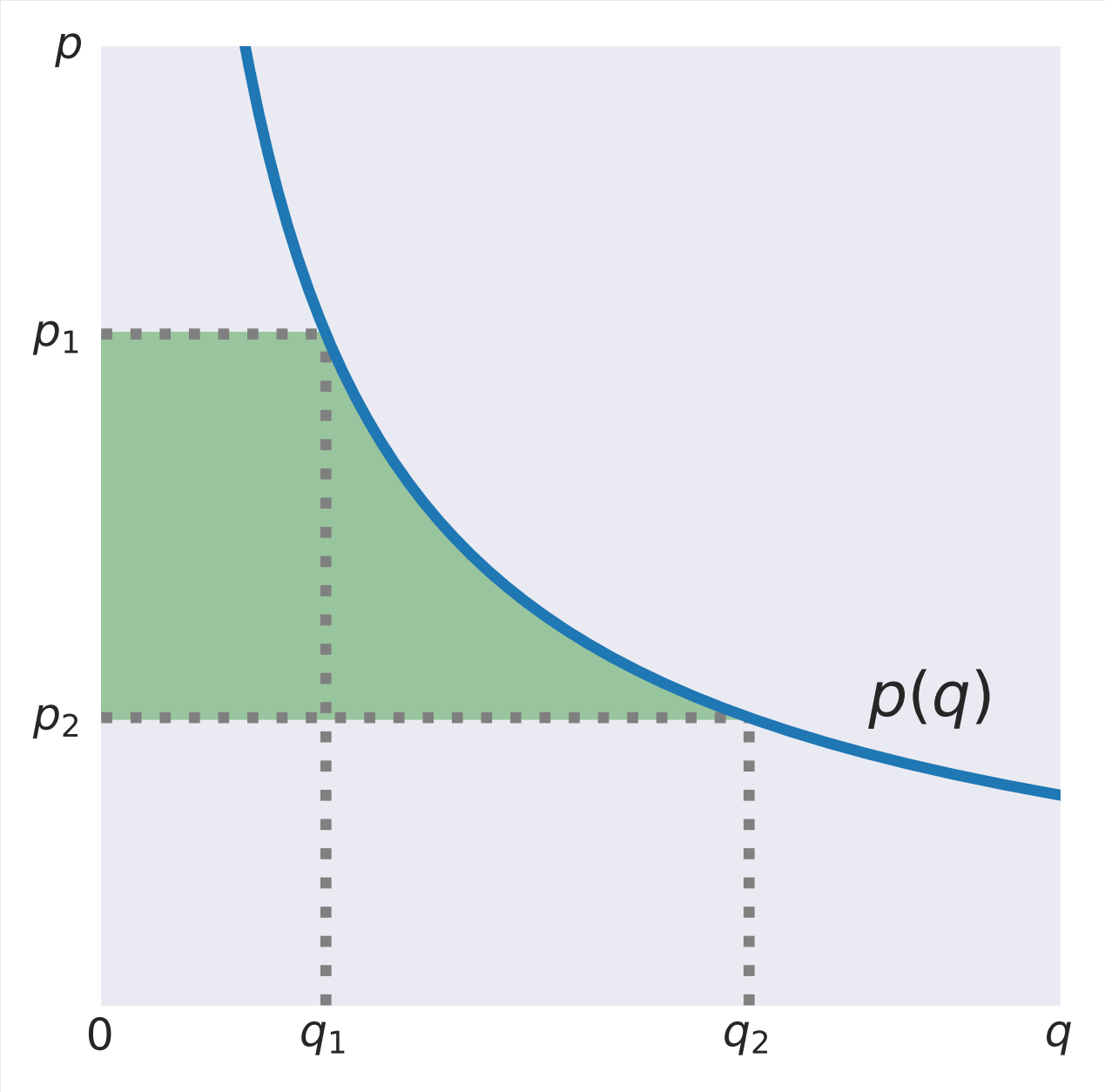

Figura 6.10: Cambio en el excedente del consumidor

3. Calculando valores esperados

- En economía, frecuentemente calculamos el valor esperado de una función f de una variable aleatoria $\tilde{X}$ cond función de densidad de $w$conocida:

$$\mathbb{E}\, f(\tilde{X}) = \int f(x) w(x)\, \mathrm{d}x.$$

- Por ejemplo, el el problema de riesgo:
  - $\tilde{X}$ es el ingreso aleatorio del agente,
  - f es la función de utilidad del ingreso del agente, y
  - w es la función de densidad de probabilidad densidad del ingreso.

- En economía, dos técnicas numéricas son ampliamente usadas para calcular valores esperados.
- La cuadratura gaussiana es especialmente potente cuando la dimensión de la variable aleatoria es baja y el integrando f es suave.
- La simulación Montecarlo, discutida en la siguiente sección, es fácil de implementar, y es especialmente útil cuando la variable aleatoria tiene dimensión alta.

Cuadratura de Gauss

- La cuadratura de Gauss reemplaza la variable aleatoria continua $\tilde{X}$ con una variable aleatoria discreta con la cual es más fácil trabajar.
- Específicamente, los n puntos de masa x_i y n probabilidades w_i de la variable aleatoria discreta son escogidos de tal manera que se repliquen la media, varianza, asimetría, curtosis, y, más generalmente, los mismos $2n-1$ primeros momentos de $\tilde{X}$.
- Esto impone $2n$ condiciones de "coincidencia de momentos"

$$\sum_{i=1}^{n} w_i x_i^k = E\tilde{X}^k, \qquad k = 0, \ldots, 2n-1,$$

que puede resolverse para los n puntos de masa y n probabilidades usando métodos de ecuaciones no lineales.

- Dada la aproximación discreta de $\tilde{X}$, fácilmente podemos calcular una aproximación de la esperanza de una función arbitraria f de $\tilde{X}$ así:

$$Ef(\tilde{X}) \approx \sum_{i=1}^{n} w_i f(x_i).$$

- Por construcción, la aproximación de la esperanza será exacta si f es a polinomio de grado $2n-1$ o menos.
- Esto sugiere que la aproximación de la esperanza debe ser precisa si f puede ser razonablemente aproximada con un polinomio de gradeo $2n-1$ o menos; es decir, si f es suave.

Ejemplo 3:
Cuadratura de Gauss

- Para $n = 3$, los puntos de masa x_1, x_2, x_3 y probabilidades w_1, w_2, w_3 de la cuadratura de Gauss para la variable normal estándar $\tilde{Z}$ se obtienen haciendo coincidir los momentos 0 al 5:

$$\sum_{i=1}^{n} w_i x_i^k = E\tilde{Z}^k \qquad (k = 0, \ldots, 2n-1)$$

$$w_1 x_1^0 + w_2 x_2^0 + w_3 x_3^0 = \mathbb{E}\,\tilde{Z}^0 = 1 \qquad (k = 0)$$

$$w_1 x_1^1 + w_2 x_2^1 + w_3 x_3^1 = \mathbb{E}\,\tilde{Z}^1 = 0 \qquad (k = 1)$$

$$w_1 x_1^2 + w_2 x_2^2 + w_3 x_3^2 = \mathbb{E}\,\tilde{Z}^2 = 1 \qquad (k = 2)$$

$$w_1 x_1^3 + w_2 x_2^3 + w_3 x_3^3 = \mathbb{E}\,\tilde{Z}^3 = 0 \qquad (k = 3)$$

$$w_1 x_1^4 + w_2 x_2^4 + w_3 x_3^4 = \mathbb{E}\,\tilde{Z}^4 = 3 \qquad (k = 4)$$

$$w_1 x_1^5 + w_2 x_2^5 + w_3 x_3^5 = \mathbb{E}\,\tilde{Z}^5 = 0 \qquad (k = 5)$$

- Resolviendo las ecuaciones no lineales encontramos que los puntos de masa y probabilidades son

$$\begin{aligned} x_1 &= -\sqrt{3} & w_1 &= 1/6 \\ x_2 &= 0 & w_2 &= 2/3 \\ x_3 &= \sqrt{3} & w_3 &= 1/6 \end{aligned}$$

- El valor exacto de $\mathbb{E}\exp(\tilde{Z})$, con cuatro decimales significativos, es 1.6487.
- La aproximación de la cuadratura de Gauss,

$$E\exp(\tilde{Z}) \approx \frac{1}{6}\exp(-\sqrt{3}) + \frac{2}{3}\exp(0) + \frac{1}{6}\exp(\sqrt{3}) = 1.6382,$$

es precisa a menos de 1%, un hecho notable si se considera que usamos una aproximación de solo tres puntos.

Herramientas de CompEcon

- El paquete CompEcon contiene funciones para generar aproximaciones discretas de distribuciones de probabilidad comunes.
- Todas estas funciones generan puntos de masa `x` y probabilidades `w`, y requieren el número de puntos de masa `n` como insumo, pero difieren respecto a los demás insumos:

- Distribución normal

```
x, w = qnwnorm(n,mu,var)
```

donde `mu` es la media y `var` es la varianza.

- Distribución lognormal

```
x, w = qnwlogn(n,mu,var)
```

donde `mu` es la log-media y `var` es la log-varianza.

- Distribución beta

```
x, w = qnwbeta(n,a,b)
```

donde `a` y `b` son los parámetros de forma.

- Distribución gamma

```
x,w = qnwgamma(n,a,b)
```

donde `a` es el parámetro de forma y `b` es el parámetro de escala.

Ejemplo 4:
Un problema de riesgo

- Veamos de nuevo el “problema de riesgo”, en el cual un agente tiene función de utilidad del ingreso $u(y) = -\exp(-\alpha y)$, $\alpha > 0$, y enfrenta un ingreso incierto $\tilde{y}$ que es lognormalmente distribuido con parámetros μ y σ^2.
- £Aceptaría el agente un ingreso cierto $y^* = 1$ en vez del ingreso incierto $\tilde{y}$ si $\mu = 0$, $\sigma^2 = 0.1$, y $\alpha = 2$?

- Para calcular la utilidad esperada del agente con ingreso aleatorio, ejecutamos el código

```
from compecon import qnwlogn
from numpy import exp

n = 100
mu, var, alpha = 0, 0.1, 2
y, w = qnwlogn(n,mu,var)
Eu = -w.dot(exp(-alpha*y))
```

- Esto resulta en la aproximación $\mathbb{E}\,u(\tilde{y}) = -0.148$, que es menor a $u(y^*) = -0.135$.
- Sí, el agente aceptaría el ingreso cierto.

- La función `qnwnorm` también genera aproximaciones discretas para variables normales multivariadas.
- Para generar puntos de masa y probabilidades para d variables conjuntamente distribuidas normalmente, ejecutamos el código

```
x, w = qnwnorm(n,mu,var);
```

  donde `n` es un d vector que indica el número de puntos de masa para cada variable, `mu` es el d vector de medias, y `var` es la matriz $d \times d$ de covarianzas.
- Como resultado, `x` es una matriz $d \times N$ de puntos de masa y `w` es un N vector de probabilidades, donde $N = n(1) \cdot n(2) \cdot \ldots \cdot n(d)$.

Ejemplo 5:
Un problema del agricultor

- El ingreso por hectárea de un agricultor es el producto del precio por unidad $\tilde{p}$ y el rendimiento por hectárea $\tilde{y}$, cuyos logaritmos están distribuidos como una normal conjunta con vector media y matriz de covarianza

$$\mu = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \qquad \Sigma = \begin{bmatrix} -0.2 & -0.1 \\ -0.1 & -0.4 \end{bmatrix}$$

- Para calcular el ingreso esperado del agricultor, usamos una cuadrícula de 150 puntos de masa formada por el producto cartesiano de 10 nodos de precio y 15 nodos de rendimiento:

```
from compecon import qnwnorm
from numpy import exp
mu, sigma = [1, 2], [[0.2,-0.1],[-0.1, 0.4]]
(p,y), w = qnwnorm([10,15], mu, sigma)
expectedrevenue = w.dot(exp(p+y))
```

4. Simulación de Montecarlo

Simulación de Montecarlo

- La simulación de Montecarlo es una forma alternativa de calcular esperanzas, que es especialmente útil cuando la variable aleatoria tiene una dimensión alta.
- La simulación de Montecarlo está motividad por la ley de los grandes números.
- Esta ley afirma que si $x_1, x_2, \ldots$ son realizaciones independientes de una variable aleatoria $\tilde{X}$ y f es una función continua, entonces

$$\mathbb{E}\, f(\tilde{X}) = \lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} f(x_i)$$

con probabilidad uno.

- El esquema de simulación de Montecarlo es simple.
- Para calcular una aproximación de $Ef(\tilde{X})$, tomamos una muestra aleatoria $x_1, x_2, \ldots, x_n$ de la distribución de $\tilde{X}$ y fijamos

$$\mathbb{E}\, f(\tilde{X}) \approx \frac{1}{n} \sum_{i=1}^{n} f(x_i).$$

- £Pero como tomamos una muestra aleatoria?

Generadores de números aleatorios

- Un generador de números aleatorios es un algoritmo que genera lo que aparenta ser una secuencia de realizaciones independientes de una variable aleatoria con una distribución especificada.
- Un problema fundamental con los llamados generadores de números aleatorios es que utilizan reglas puramente determinísticas, no aleatorias. iteration rules.
- Si iniciamos un generador repetidamente desde un mismo punto, siempre generará la misma secuencia de números "aleatorios".
- Lo más que podemos decir de los generadores de números aleatorios es que uno bueno generará realizaciones que pasan algunas pruebas estadísticas de aleatoriedad.
- Por esta razón, a los generadores de números aleatorios es mejor llamarlos generadores de números "pseudo-aleatorios".

Números aleatorios con numpy

- `numpy.random` contiene dos generadores intrínsecos de números aleatorios.
- `rand(m,n)` genera una matriz $m \times n$ de números que son uniforme e independientemente distribuidos en el intervalo $[0, 1]$.
- `randn(m,n)` genera una matriz $m \times n$ de números que son normal estándar e independientemente distribuidos.

Ejemplo 6:

Generando números aleatorios de una distribución normal estándar

- Para general 100 000 realizaciones independientes de una variable aleatoria con distribución normal estándar, ejecutamos

```
from numpy.random import randn
x = randn(100_000)
```

- Basándonos en lo que sabemos de teoría de la distribución, £qué esperamos obtener con este código?:

```
x.media()
x.std()
(x < 1.6449).media()
```

- Esperamos obtener 0, 1, y 0.95.
- Los valores calculados están cerca, pero no son exactos.

Ejemplo 7:
Generando números aleatorios de una distribución uniforme

- Para generar 100 000 realizaciones independientes de una variable aleatoria (0,1), ejecutamos

```
from numpy.aleatorio import rand
x = rand(100_000)
```

- Basándonos en lo que sabemos de teoría de la distribución, £qué esperamos obtener con este código?:

```
x.media()
x.std()
(x < 0.2).media()
```

- Esperamos obtener 0.5, $\sqrt{1/12} = 0.2887$, y 0.2.
- Los valores calculados están cerca, pero no son exactos.

- La librería `scipy.stats` contiene generadores de números aleatorios para más de 90 distribuciones, incluyendo las distribuciones beta, binomial, exponencial, valor extremo, gamma, logística, lognormal, normal, y Poisson.
- Para generar una matriz $n \times m$ de realizaciones independientes para una variable aleatoria dada, el prototipo de código es

```
x = dist.rvs(*parameters,size=[n,m])
```

donde `dist` es el nombre de la distribución y `parameters` sus parámetros.

Ejemplo 8:
Aproximando un valor esperado

- Para aproximar $\mathbb{E}\,\tilde{X}^{-1}$ donde $\tilde{X}$ tiene distribución beta con parámetros de forma 1.5 y 3.0, ejecutamos

```
x = beta.rvs(1.5, 3.0, size=100_000)
(1 / x).mean()
```

- Para aproximar $\mathbb{E}\min(\tilde{X}, \tilde{Y})$ donde $\tilde{X}$ y $\tilde{Y}$ son independientes, $\tilde{X}$ tiene distribución gammacon parámetro de forma 1.5 y parámetro de escala 3.0, y $\tilde{Y}$ tiene distribución exponencial con media 1, ejecutamos

```
x = gamma.rvs(1.5,3.0,size=100_000)
y = exponential.rvs(1,size=100_000)
np.minimum(x,y).mean()
```

Ejemplo 9:
Aproximando una probabilidad

- Para aproximar $\Pr(\tilde{Y} < \tilde{X}^2)$ donde $\tilde{X}$ y $\tilde{Y}$ son independientes, $\tilde{X}$ tiene distribución de valor extremo con parámetro de ubicación 0.5 y parámetro de escala 1.0, y $\tilde{Y}$ tiene distribución geométrica con parámetro de probabilidad 0.3, ejecutamos

```
x = genextreme.rvs(0,0.5,1.0,size=100_000)
y = geom.rvs(0.3,size=100_000)
(y<x**2).mean()
```

- Una manera más precisa sería ejecutar

```
geom.cdf(x**2,0.3).mean()
```

Ejemplo 10:
Un precio de mercancía aleatorio

- El movimiento semanal del precio de una mercancía está descrito por

$$\log(p_{t+1}) = \log(p_t) + \tilde{\epsilon}_t$$

donde los $\tilde{\epsilon}_t$ tiene distribución normal independiente e idéntica con media $\mu = 0.005$ y desviación estándar $\sigma = 0.02$.

- Para simular tres series de tiempo semanales de 40 precios, empezando con un precio de 2, ejecutamos

```
m, n = 3, 40
mu, sigma = 0.005, 0.02
e = norm.rvs(mu,sigma,size=[n,m])
logp = np.zeros([n+1,m])
logp[0] = np.log(2)
for t in range(40):
    logp[t+1] = logp[t] + e[t]

plt.plot(np.exp(logp))
```

Figura 6.11: Simulación de series de tiempo

- La librería `scipy.stats` también contiene un generador de números aleatorios para vectores de variables aleatorias multinormales.
- Para generar n realizaciones independientes de un vector de dimensión d de variables normalmente distribuidas, el prototipo de código es

```
r = multivariate_normal.rvs(mu, var, size=n)
```

donde `mu` es un d vector de medias y `var` es una matriz $d \times d$ de varianzas definida positiva.

Ejemplo 11:
Un agricultor enfrenta precios y rendimientos aleatorios

- El ingreso por hectárea de un agricultor el el producto del precio por unidad $\tilde{p}$ y el rendimiento por hectárea $\tilde{y}$, cuyos logaritmos tienen una distribución normal conjunta con media covarianza

$$\mu = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \qquad \Sigma = \begin{bmatrix} 0.2 & -0.1 \\ -0.1 & 0.4 \end{bmatrix}$$

- Para calcular el ingreso esperado del agricultor usando 100 000 realizaciones independientes de la distribución normal conjunta del precio y rendimiento, ejecutamos

```
mu, sigma = [1, 2], [[0.2, -0.1],[-0.1, 0.4]]
p, y = multivariate_normal.rvs(mu, sigma, size=100_000).T
expectedrevenue = np.exp(p + y).mean()
```

Ventajas de la simulación de Montecarlo

La simulación de Montecarlo simulación tiene ciertas ventajas.

- Es fácil de implementar.
- La mayoría de los paquetes de software de aplicaciones (e.g., Excel) proveen generadores de números aleatorios, pero no funciones que calculen puntos de masa y probabilidades de la cuadratura de Gauss.
- No sufre de la "maldición de la dimensionalidad":
  - la cuadratura de Gauss multidimensional sí la sufre: si usamos n nodos en cada una de las d dimensiones, terminaríamos con n^d puntos de masa.
  - la simulación de Montecarlo no la sufre. Además, es especialmente útil cuando se simulan series de tiempo de variables aleatorias autocorrelacionadas, cuya dimensión es igual al tamaño de la series.

Desventajas de la simulación de Montecarlo

Sin embargo, la simulación de Montecarlo tiene algunas desventajas

- Las aproximaciones generadas cambiarán de una simulación a la siguiente, y están sujetas a errores de muestreo que no pueden acotarse con certidumbre.
- Puede aumentarse la precisión de la aproximación, en un sentido estadístico dudoso, incrementando el tamaño de la muestra aleatoria, pero esto puede ser costoso.
- La simulación de Montecarlo debe evitarse cuando otros métodos sean prácticos, y utilizarse sólo cuando se calculan esperanzas de variables aleatorias de muchas dimensiones.

Los métodos cuasi Montecarlo, que veremos a continuación, evitan algunos de los problemas asociados con la simulación de Montecarlo.

5. Integración cuasi Montecarlo

- Los métodos cuasi Montecarlo emplean secuencias determinísticas de nodos x_i con la propiedad de que

$$\lim_{n\to\infty} \frac{b-a}{n} \sum_{i=1}^{\infty} f(x_i) = \int_a^b f(x)\, dx,$$

para funciones suaves f sin importar que pasen pruebas de aleatoriedad.

- Secuencias determinísticas de nodos escogidas para llenar espacio de manera regular en general proveen aproximaciones de integración más precisas que las de secuencias pseudo-aleatorias.

- Hay numerosos algoritmos para genera secuencias equidistribuidas, entre ellas las secuencias de Neiderreiter y de Weyl.
- Los algoritmos se explican con detalle en los libros de Judd (1998) y Miranda y Fackler (2002), pero no los estudiaremos con detalle acá.
- Examinemos ejemplos de secuencias equidistribuidas en el cuadrado unitario.

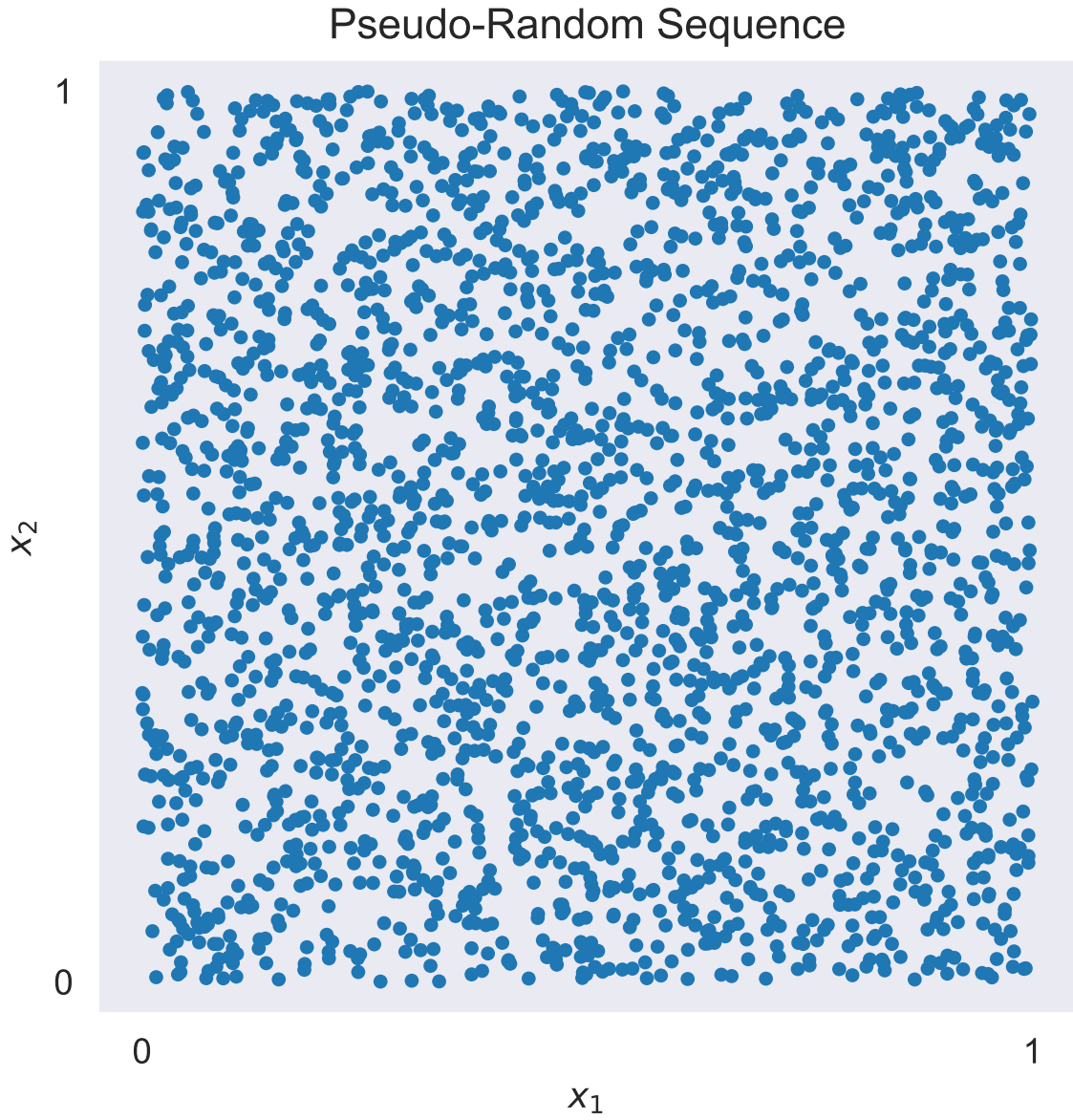

Figura 6.12: Secuencia pseudo-aleatoria en el cuadrado unitario

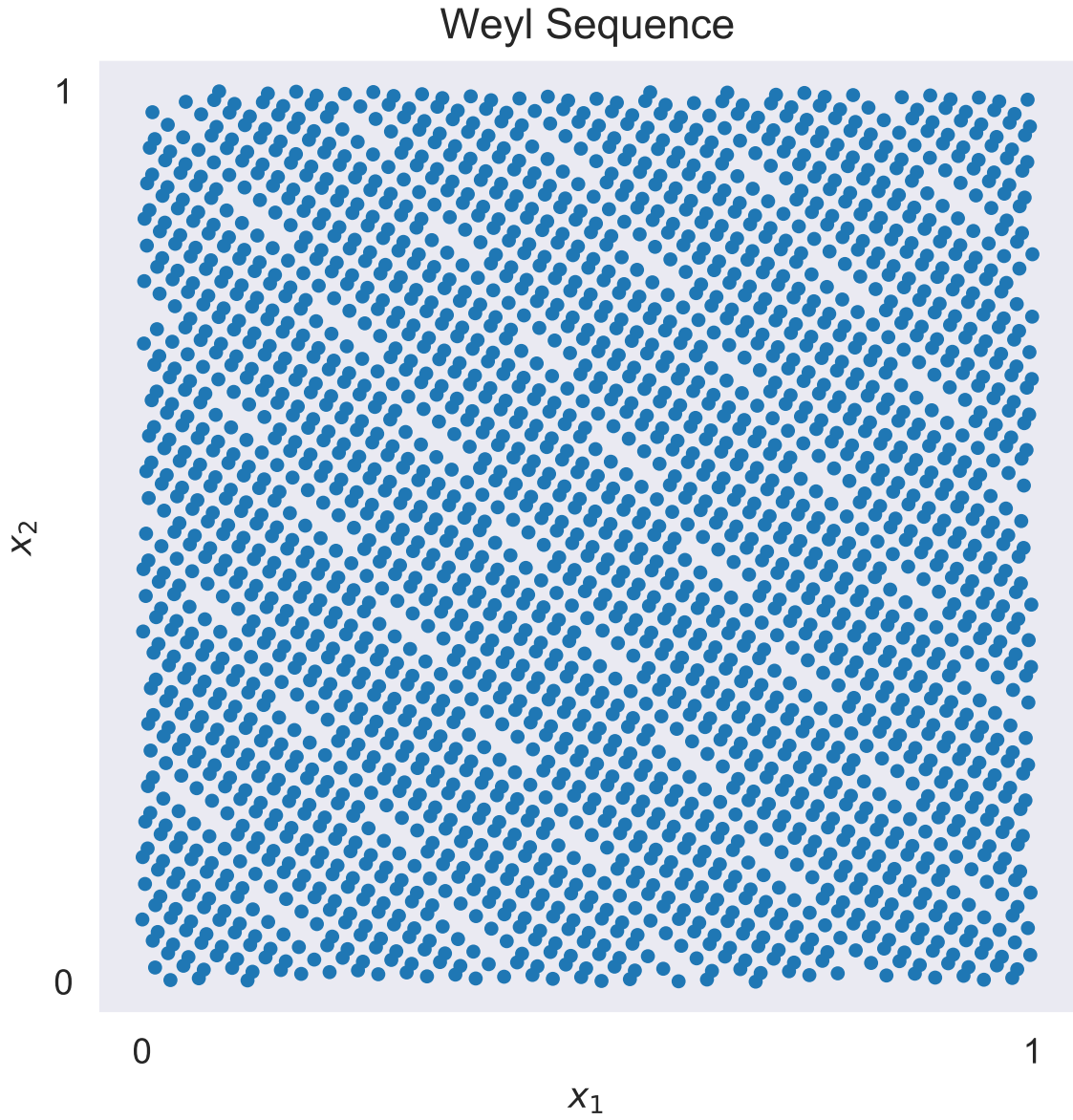

Figura 6.13: Secuencia Weyl en el cuadrado unitario

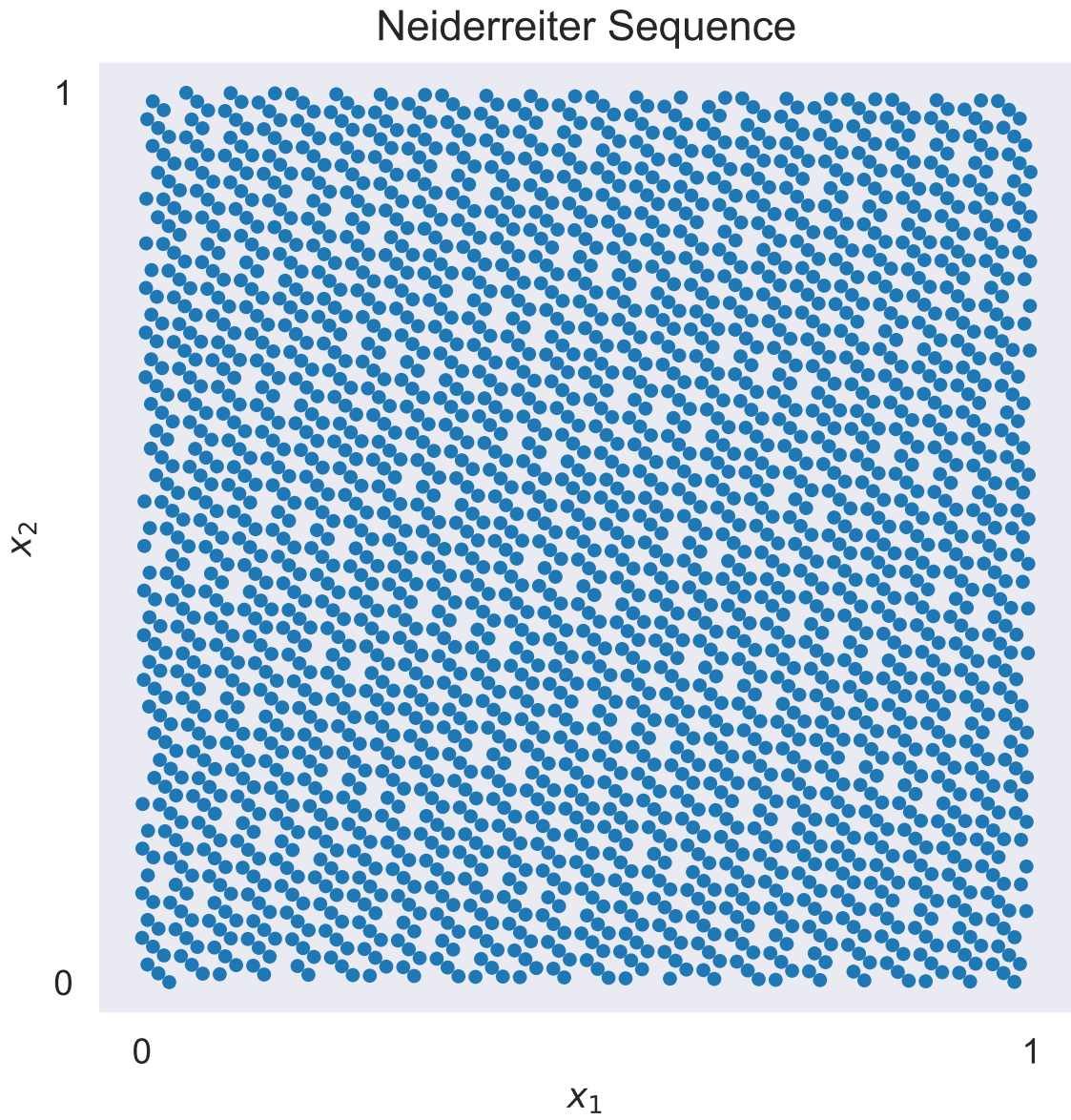

Figura 6.14: Secuencia Neiderreiter en el cuadrado unitario

- La función `qnwequi` de CompEcon genera nodos equidistribuidos y ponderaciones así

```
x, w = qnwequi(n,a,b,type)
```

- Insumos `n` = el número de nodos de integración, `a` = límite izquierdo de integración, y `b` = límite derecho de integración.
- El insumo adicional `type` se refiere al tipo de secuencia equidistribuida:

  'N' = Neiderrieter (predeterminado),
  'W' = Weyl, y
  'R' = pseudo-aleatorio uniforme.

- Los límites de integración son d vectores si la integración es sobre un hipercubo de dimensión d.

Ejemplo 12:
Integración cuasi Montecarlo

Con 7 dígitos significativos,

$$
\begin{aligned}
A &= \int_{-1}^{1}\int_{-1}^{1} e^{-x_1}\cos^2(x_2)\,\mathrm{d}x_1\,\mathrm{d}x_2 \\
&= \int_{-1}^{1} e^{-x_1}\,\mathrm{d}x_1 \times \int_{-1}^{1}\cos^2(x_2)\,\mathrm{d}x_2 \\
&= \left(e-\tfrac{1}{e}\right)\times\left(1+\tfrac{1}{2}\sin(2)\right) && \approx 3.4190098
\end{aligned}
$$

Para aproximar la integral usando un esquema Neiderrieter de 10 000 nodos, ejecutamos

```
from numpy import exp, cos
from compecon import qnwequi
n, a, b = 10_000, [-1, -1], [1, 1]
(x1, x2), w = qnwequi(n,a,b,'N')
A = w.dot(exp(-x1) * cos(x2)**2)
```

```
A = 3.421441412
```

$$A = \int_{-1}^{1} \int_{-1}^{1} e^{-x_1} \cos^2(x_2)\, dx_1\, dx_2 \approx 3.4190098$$

Nodes	Random	Neiderreiter	Weyl
10^3	-2.8	-2.9	-3.5
10^4	-2.5	-3.1	-3.9
10^5	-2.7	-4.0	-4.4
10^6	-3.2	-5.4	-5.7

Cuadro 2: Log10 de los errores de aproximación para A, usando secuencias equidistribuidas alternativas.

6. Diferenciación numérica

Derivadas de primer orden

- La manera más natural de aproximar una derivada es reemplazarla con una diferencia finita:

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}.$$

- En teoría, el error de aproximación desaparece conforme h va a 0, por lo que si escogemos un h suficientemente pequeño, el error deberá ser pequeño.

- El error de aproximación puede ser acotado usando el teorema de Taylor, que afirma que

$$f(x+h) = f(x) + f'(x)h + O(h^2),$$

donde $O(h^2)$ es proporcional al cuadrado de h.

- Reacomodando,

$$f'(x) = \frac{f(x+h) - f(x)}{h} + O(h)$$

- Como $O(h^2)/h = O(h)$, el error de aproximación es proporcional a $|h|$.

- Sin embargo, existe una aproximación de diferencia finita más precisa para la derivada de f en x.
- Consideremos las expansiones de Taylor de segundo orden

$$f(x+h) = f(x) + f'(x)h + f''(x)\frac{h^2}{2} + O(h^3)$$

$$f(x-h) = f(x) - f'(x)h + f''(x)\frac{h^2}{2} + O(h^3).$$

- Restando la segunda expresión de la primera, reacomodando, y dividiendo por $2h$ obtenemos

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} + O(h^2).$$

- La expresión

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}$$

se conoce como la aproximación de diferencia finita centrada de la derivada de f en x.

- Su error $O(h^2)$ es un orden de magnitud más preciso que el de la aproximación de diferencia finita de un lado.

- Como en teoría los errores de aproximación de diferencia finita desaparecen conforme h se acerca a 0, podríamos estar tentados a hacer h tan pequeño como sea posible.
- Desafortunadamente, si hacemos h demasiado pequeño, los errores de redondeo podrían hacer que los resultados pierdan sentido.
- Consideremos el error de aproximación en las diferencia finitas de uno y dos lados de $\exp(x)$ en $x = 1$ como función del tamaño de h.

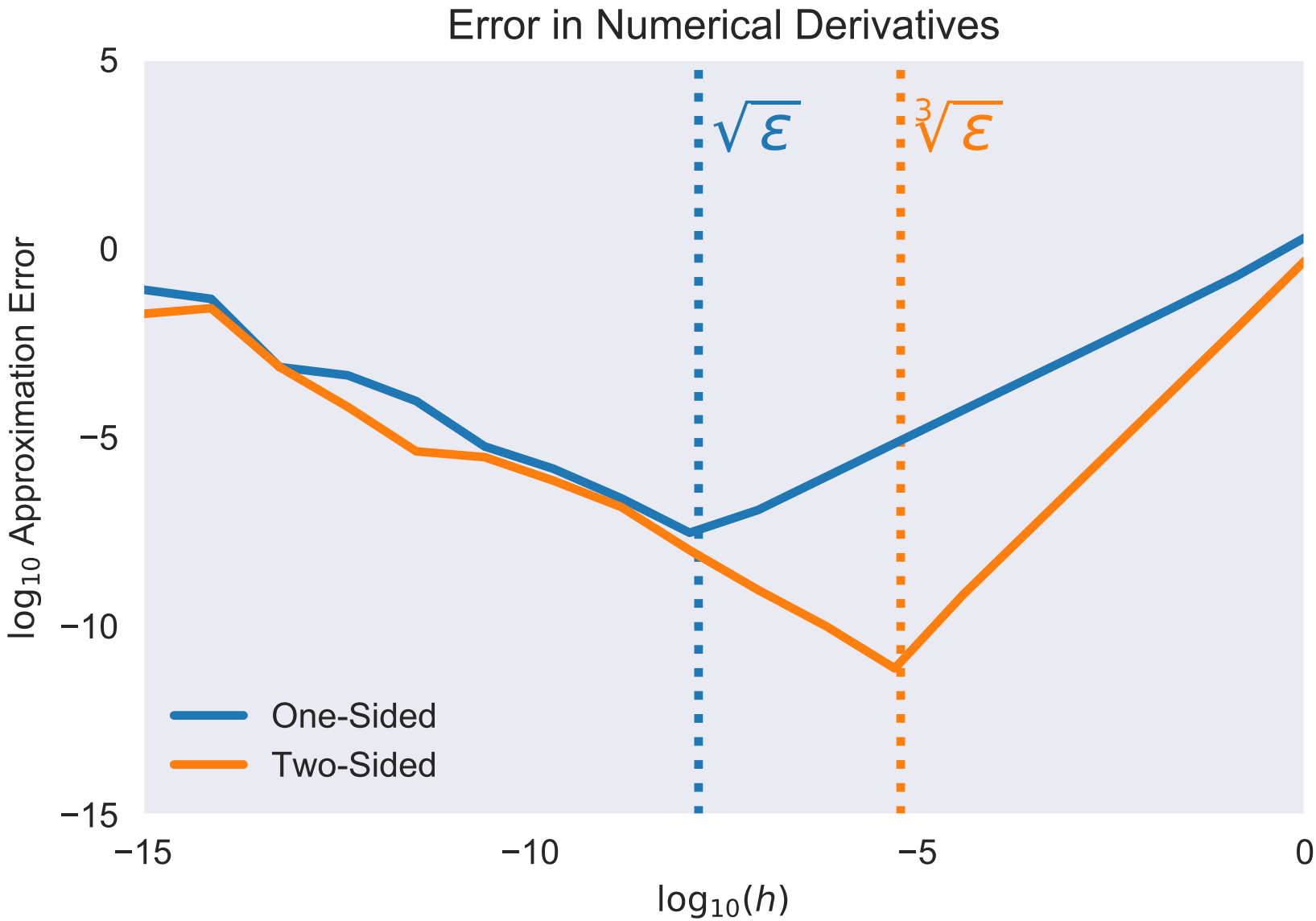

Figura 6.15: Errores de aproximación de diferencia finita de uno y dos lados para la derivada de $\exp(x)$ at $x = 1$

- La aproximación de diferencia finita centrada mejora conforme h se encoge hasta que alcanza la raíz cúbica de la precisión de máquina $\sqrt[3]{\epsilon}$.
- Reduccciones adicionales de h empeoran el error de aproximación error por los errores de redondeo.
- Esto sugiere que fijemos $h \approx \sqrt[3]{\epsilon}$ relativo a x para aproximaciones de diferencia finita centradas.
- Análisis empíricos similares sugieren que fijemos $h \approx \sqrt{\epsilon}$ relativo a x para aproximaciones de diferencia finita de un lado.

Derivadas de orden superior

- Podemos encontrar aproximaciones de diferencia finita para derivadas de orden superior siguiendo un enfoque similar.
- Por ejemplo, una aproximación de diferencia finita centrada de orden $O(h^2)$ para la segunda derivada es

$$f''(x) \approx \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}.$$

- Para demostrar que el error es $O(h^2)$, sumamos estas dos expansiones de Taylor de tercer orden

$$f(x+h) = f(x) + f'(x)h + f''(x)\frac{h^2}{2} + f'''(x)\frac{h^3}{6} + O(h^4)$$

$$f(x-h) = f(x) - f'(x)h + f''(x)\frac{h^2}{2} - f'''(x)\frac{h^3}{6} + O(h^4),$$

para obtener

$$f(x+h) + f(x-h) = 2f(x) + f''(x)h^2 + O(h^4),$$

reacomodamos y dividimos por h^2.

Herramientas de CompEcon

- Las funciones `jacobian` y `hessian` del paquete CompEcon calculan jacobianos y hessianos numéricamente.
- Esta funciones fueron introducidas previamente en el curso.
- Por conveniencia, repectimos ejemplos de cómo usarlas.

jacobian

- La función `jacobian` de CompEcon calcula la matriz $m \times n$ jacobiana de diferencia finita para una función arbitraria $f : \Re^n \mapsto \Re^m$.
- El prototipo del código es

```
J = jacobian(f, #función de la forma fval=f(x)
             x) #punto de evaluación
```

- Resultado: `J` = jacobiano de `f` en `x`

Ejemplo 13:
Calculando un jacobiano

- El jacobiano exacto de

$$f(x_1, x_2) = \begin{bmatrix} \exp(x_1) - x_2 \\ x_1 + x_2^2 \\ (1 - x_1)\log(x_2) \end{bmatrix}$$

en $(0, 1)$ es

$$f'(x_1, x_2) = \begin{bmatrix} 1 & -1 \\ 1 & 2 \\ 0 & 1 \end{bmatrix}$$

Para calcular el jacobiano numéricamente, ejecutamos

```
def f(x):
    x1, x2 = x
    y = [np.exp(x1)-x2,
         x1 + x2**2,
         (1-x1)*np.log(x2)]
    return np.array(y)

np.set_printoptions(precision=15)
print(jacobian(f,np.array([0,1])))
```

Esto da por resultado

```
[[ 1.000000000014386 -1.                ]
 [ 0.999999999996052  1.999999999990833]
 [ 0.                 1.000000000012223]]
```

hessian

- La función `hessian` de CompEcon calcula la matriz $n \times n$ hessiana de diferencia finita de una función arbitraria $f : \Re^n \mapsto \Re$.
- El prototipo de código es

```
H = hessian(f, #función de la forma fval=f(x)
            x) #punto de evaluación
```

- Resultado: `H` = hessiano de `f` en `x`

Ejemplo 14:
Calculando un hessiano

- El hessiano exacto de

$$f(x_1, x_2) = x_1^2 \exp(-x_2)$$

en $(1, 0)$ es

$$f''(x_1, x_2) = \begin{bmatrix} 2 & -2 \\ -2 & 1 \end{bmatrix}.$$

Para calcular el hessiano numéricamente, ejecutamos

```
def f(x):
    x1, x2 = x
    return x1**2 * np.exp(-x2)

np.set_printoptions(precision=15)
print(hessian(f,np.array([1, 0])))
```

Esto da por resultado

```
[[ 2.                  -2.000000006519258]
 [-2.000000006519258   0.999999985098839]]
```

References I

- Judd, Kenneth L. (1998). *Numerical Methods in Economics*. MIT Press. isbn: 978-0-262-10071-7.
- Miranda, Mario J. y Paul L. Fackler (2002). *Applied Computational Economics and Finance*. MIT Press. isbn: 0-262-13420-9.
- Romero-Aguilar, Randall (2016). *CompEcon-Python*. url: http://randall-romero.com/code/compecon/.