Integración Numérica

Métodos y Propiedades

Cristopher Arenas
cristopher.arenas@usm.cl

Universidad Técnica Federico Santa María
Computación Científica II - ILI286

v0.36b

Integración Numérica o Cuadratura

Obtención del valor $c \in \mathbb{R}$ de una cierta integral definida

$$c = \int_a^b f(x)\mathrm{d}x$$

utilizando algún método numérico.

Ejemplo:

$$\int_{-1}^{+1} e^x \, \mathrm{d}x = e^{+1} - e^{-1} = 2.35040...$$

- Existe un gran número de métodos.
- La elección del método apropiado depende, entre otras cosas, de los siguientes factores:
  - La función f ha sido evaluada previamente en puntos típicamente equiespaciados.
  - La función f puede evaluarse en puntos elegidos arbitrariamente.

Función Integrable

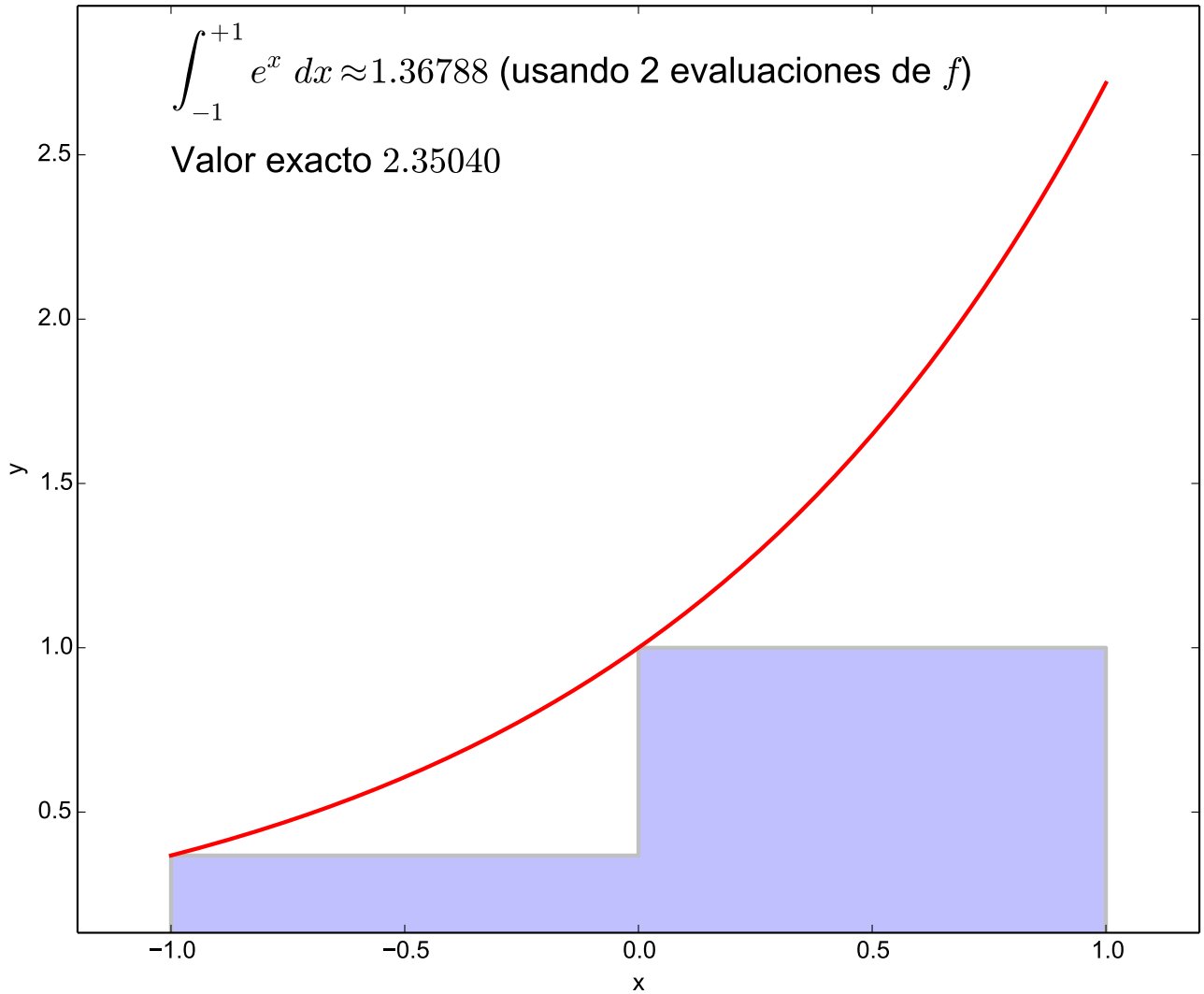
Se dice que una función es integrable cuando las sumas izquierdas y derechas de Riemann convergen al mismo valor:

$$c = \int_a^b f(x)\,\mathrm{d}x = \sup_P \sum_{i=0}^{N-1} f(x_i)(x_{i+1} - x_i)$$

$$= \sup_P \sum_{i=0}^{N-1} f(x_{i+1})(x_{i+1} - x_i)$$

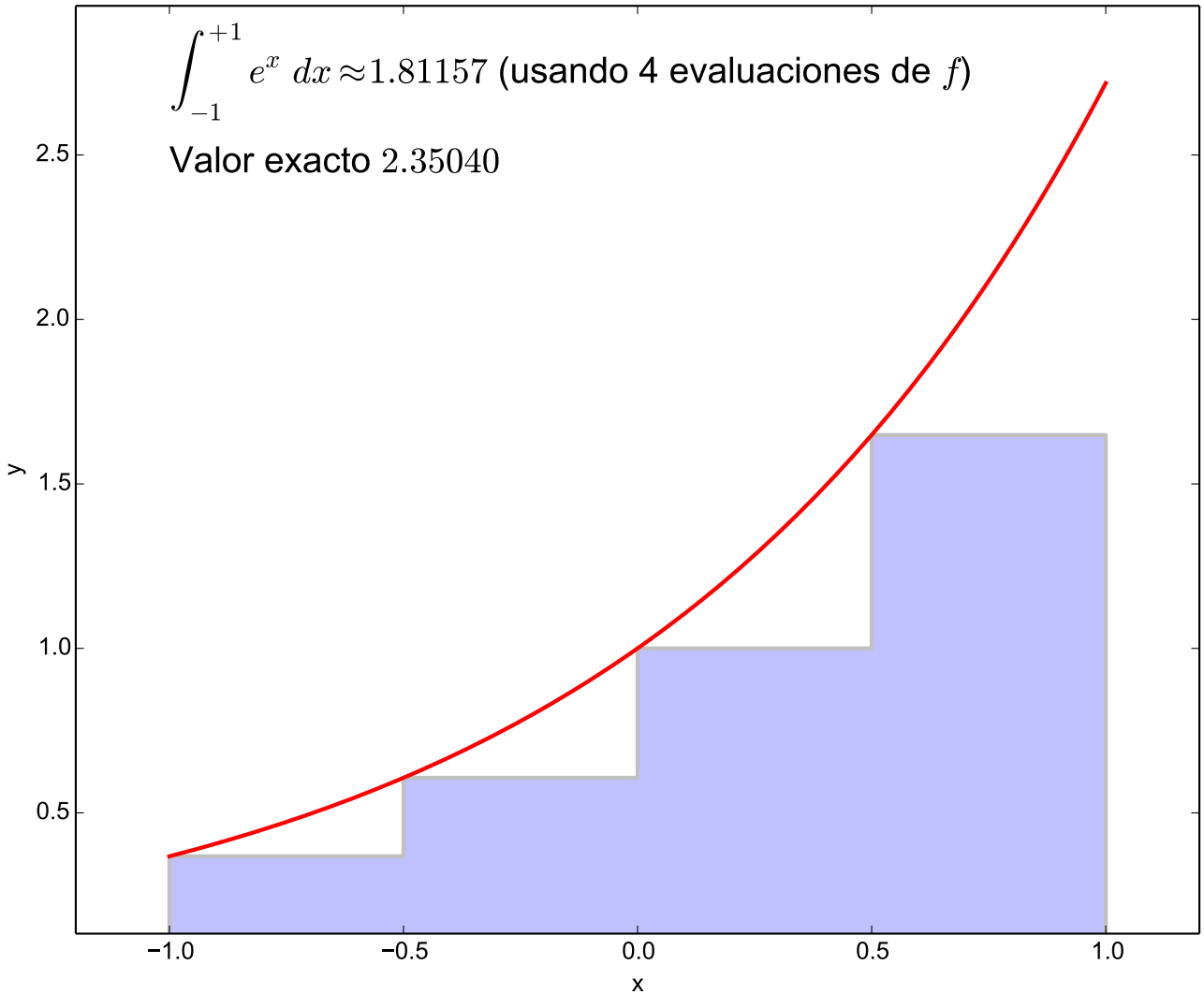
Suma de Riemann
Suma Izquierda de Riemann

$$c = \int_a^b f(x)\,\mathrm{d}x = \sup_P \sum_{i=0}^{N-1} f(x_i)(x_{i+1} - x_i)$$



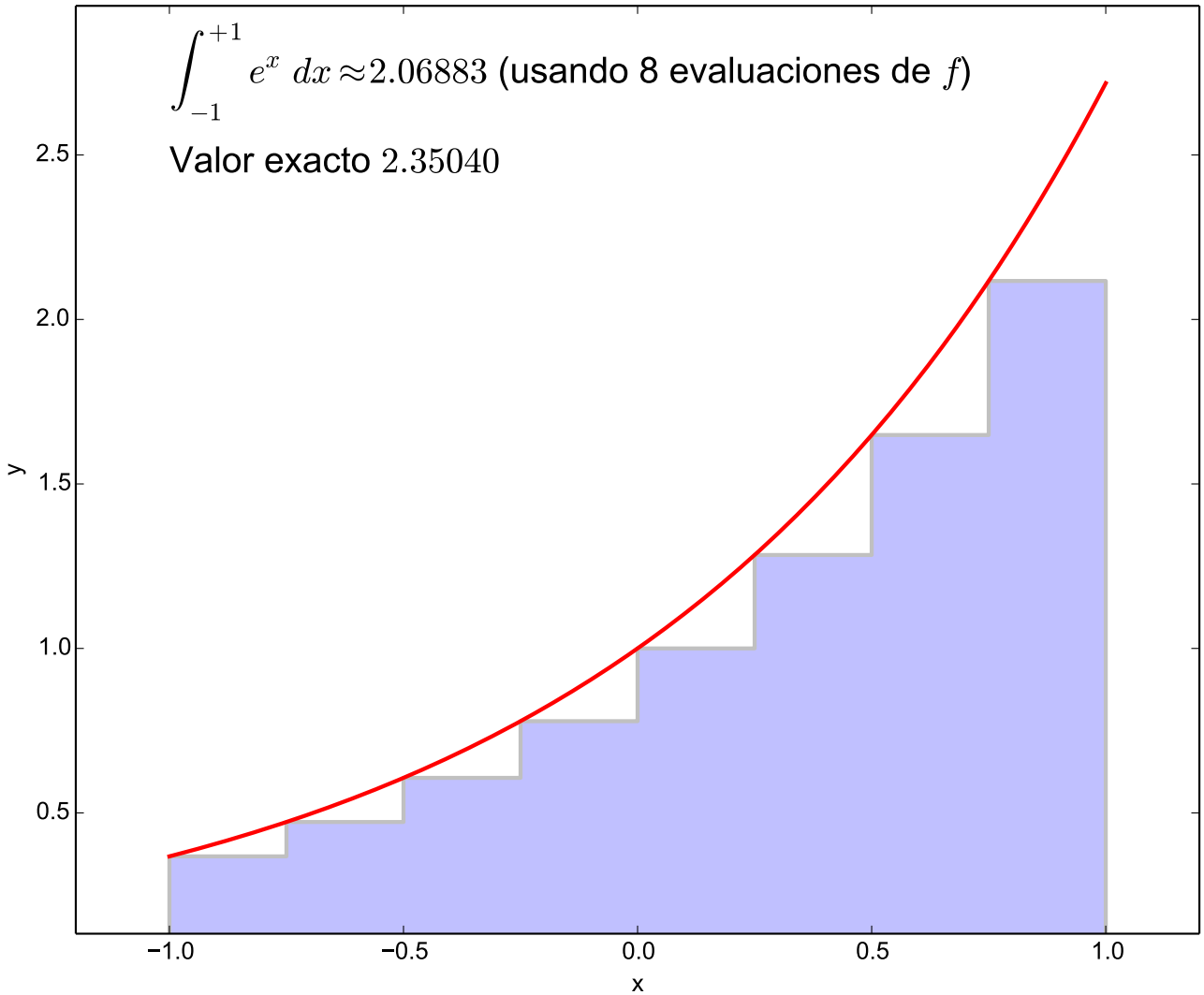
Suma de Riemann
Suma Izquierda de Riemann

$$c = \int_a^b f(x)\,\mathrm{d}x = \sup_P \sum_{i=0}^{N-1} f(x_i)(x_{i+1} - x_i)$$



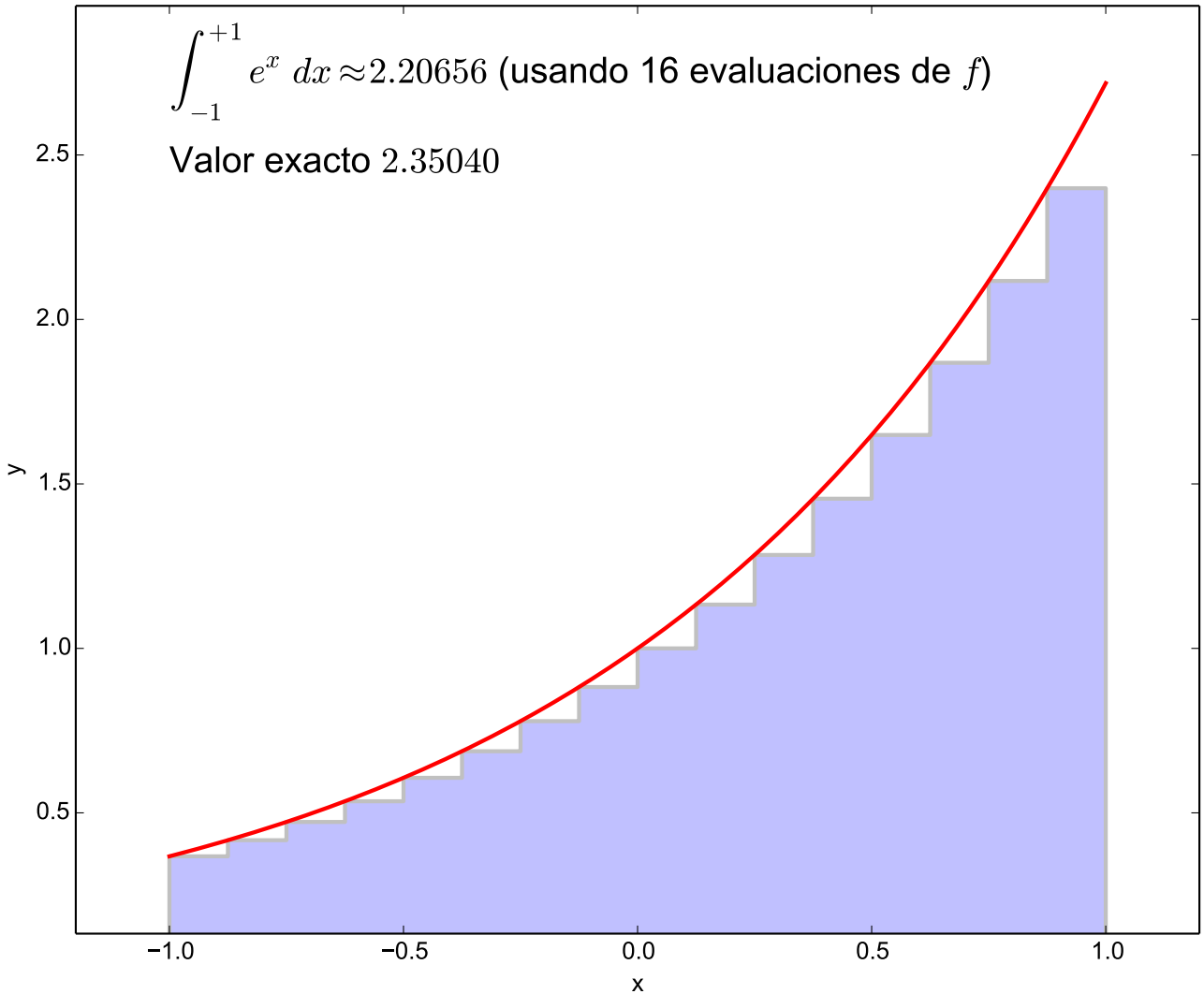
Suma de Riemann
Suma Izquierda de Riemann

$$c = \int_a^b f(x)\,\mathrm{d}x = \sup_P \sum_{i=0}^{N-1} f(x_i)(x_{i+1} - x_i)$$



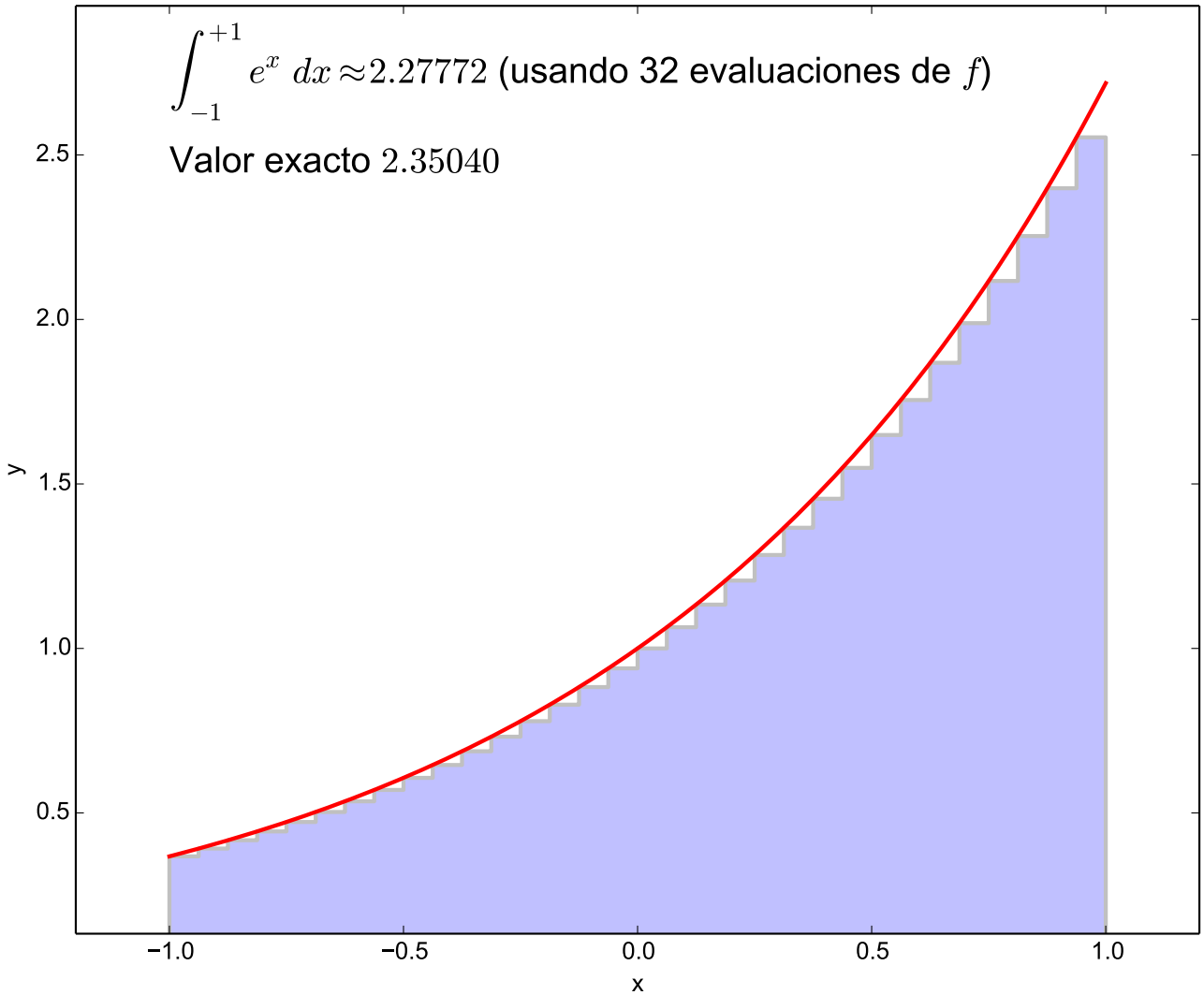
Suma de Riemann
Suma Izquierda de Riemann

$$c = \int_a^b f(x)\,\mathrm{d}x = \sup_P \sum_{i=0}^{N-1} f(x_i)(x_{i+1} - x_i)$$



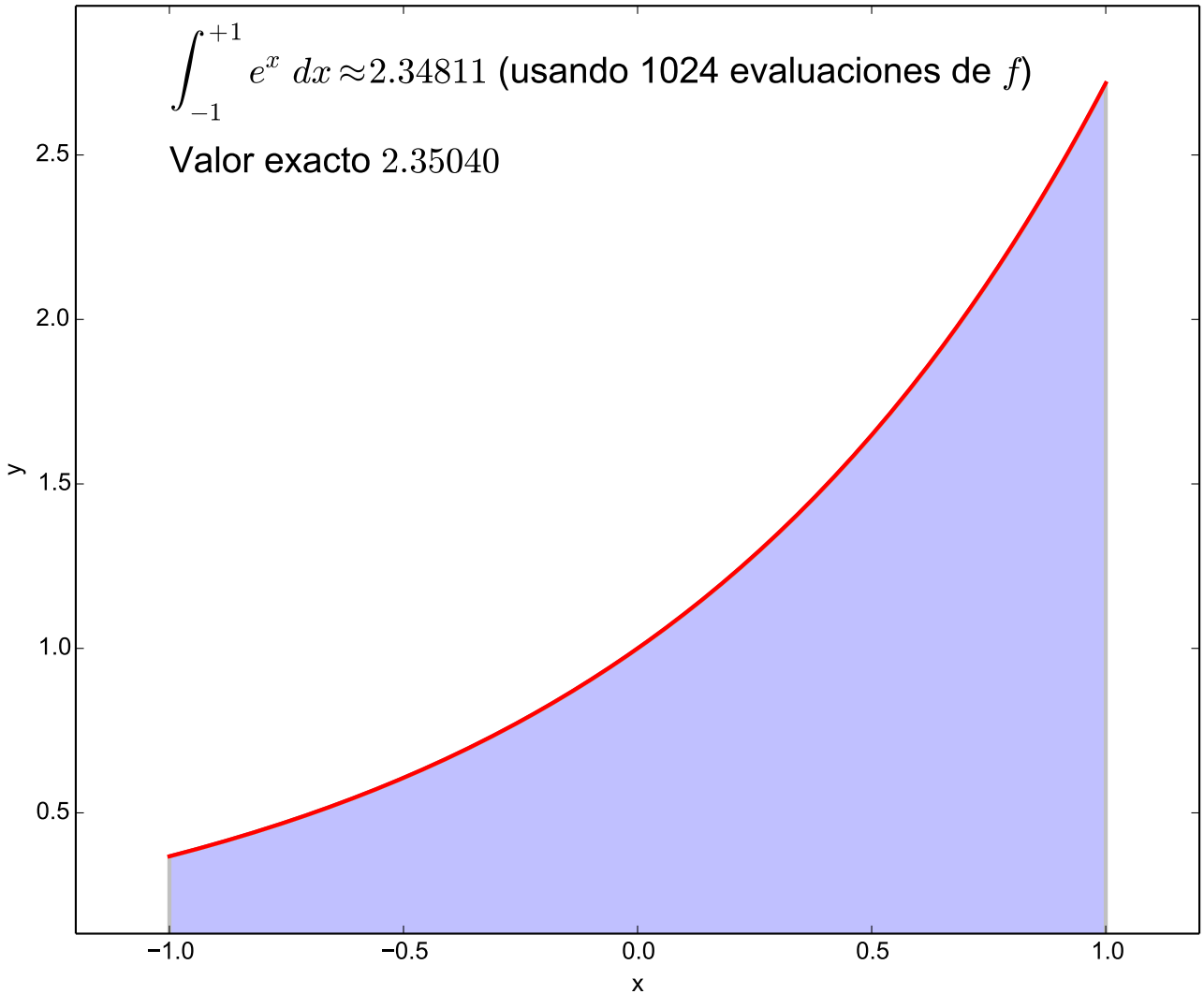
Suma de Riemann
Suma Izquierda de Riemann

$$c = \int_a^b f(x)\,\mathrm{d}x = \sup_P \sum_{i=0}^{N-1} f(x_i)(x_{i+1} - x_i)$$



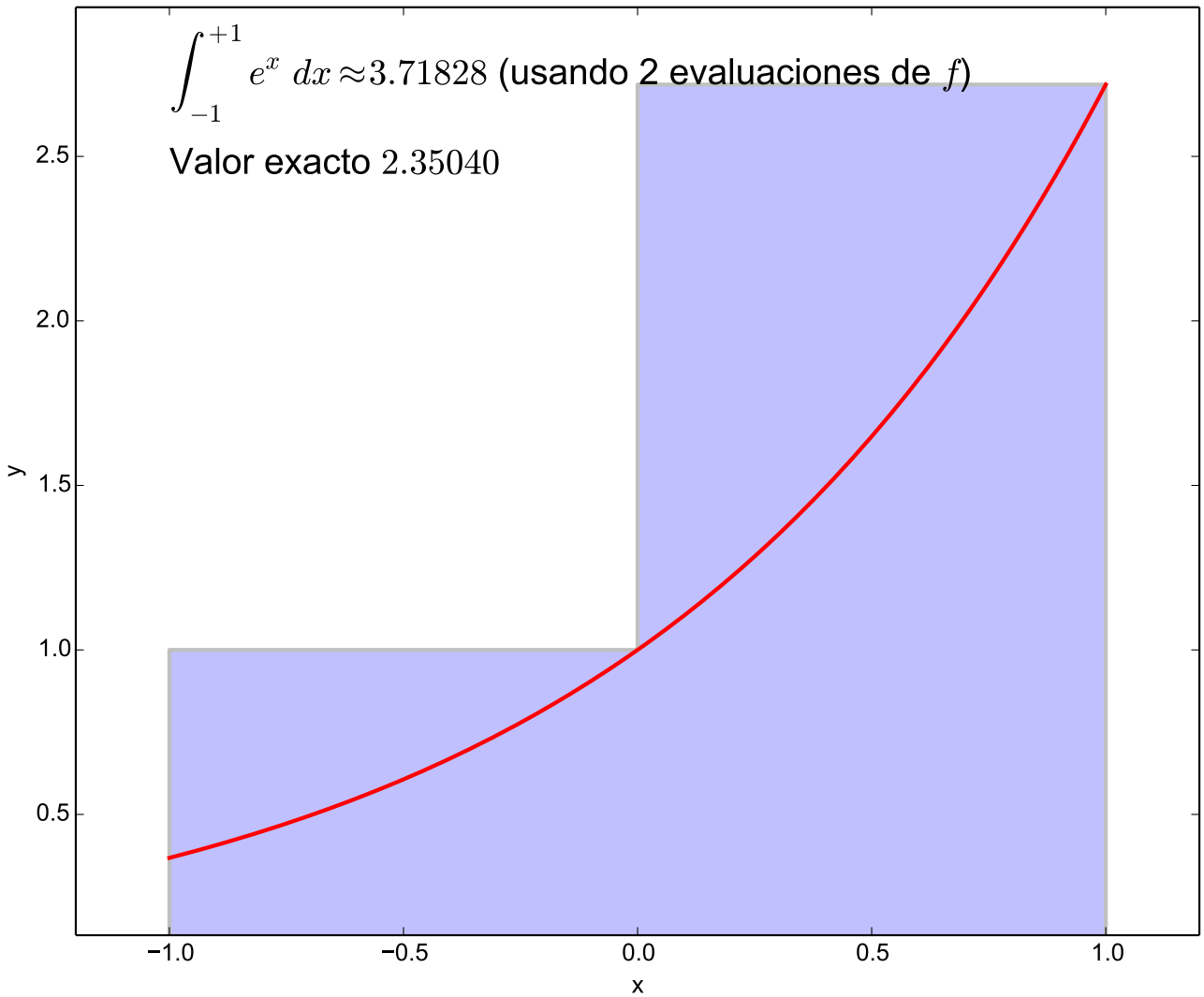
Suma de Riemann
Suma Izquierda de Riemann

$$c = \int_a^b f(x)\,\mathrm{d}x = \sup_P \sum_{i=0}^{N-1} f(x_i)(x_{i+1} - x_i)$$



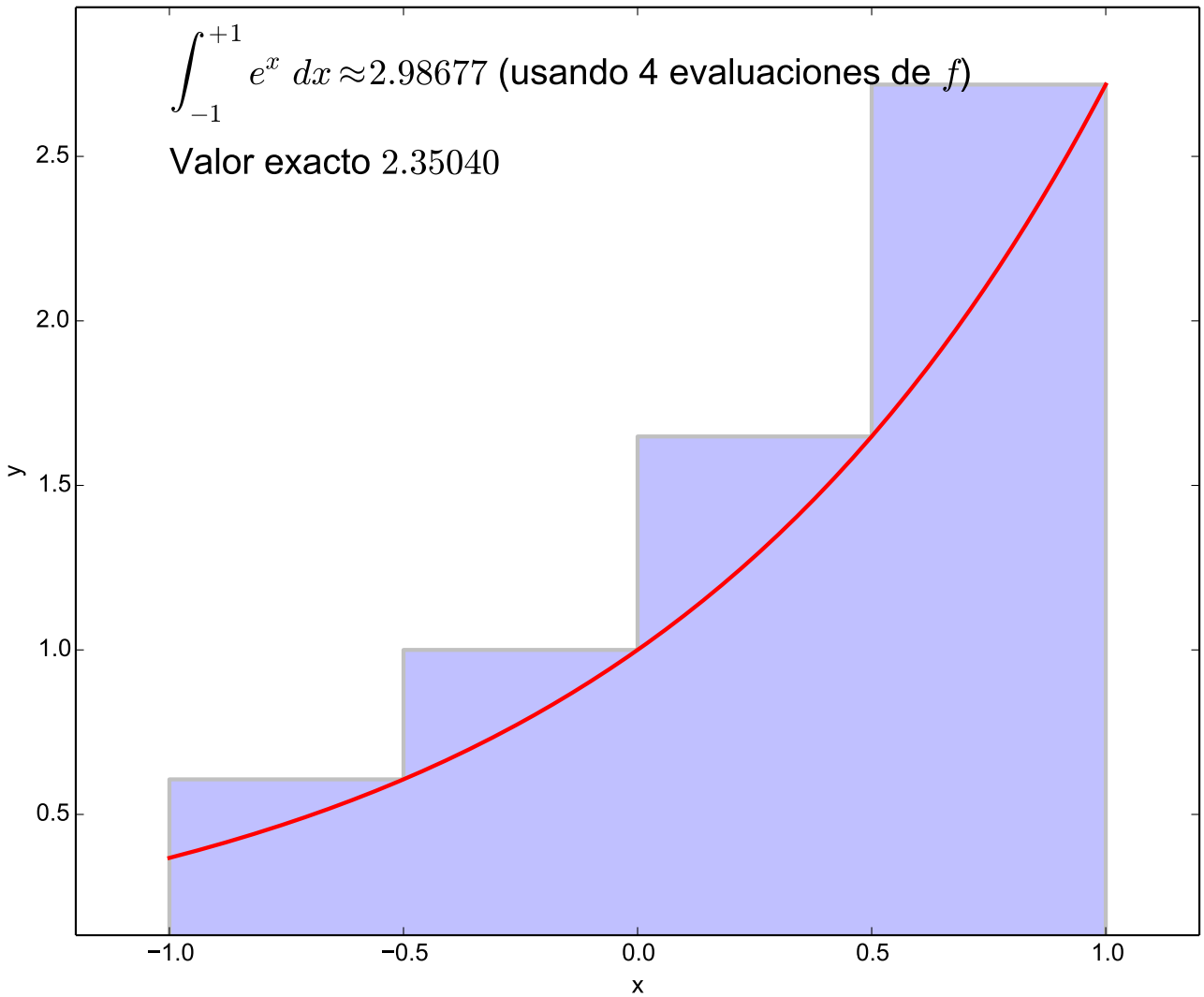
Suma de Riemann
Suma Derecha de Riemann

$$c = \int_a^b f(x)\,\mathrm{d}x = \sup_P \sum_{i=0}^{N-1} f(x_{i+1})(x_{i+1} - x_i)$$



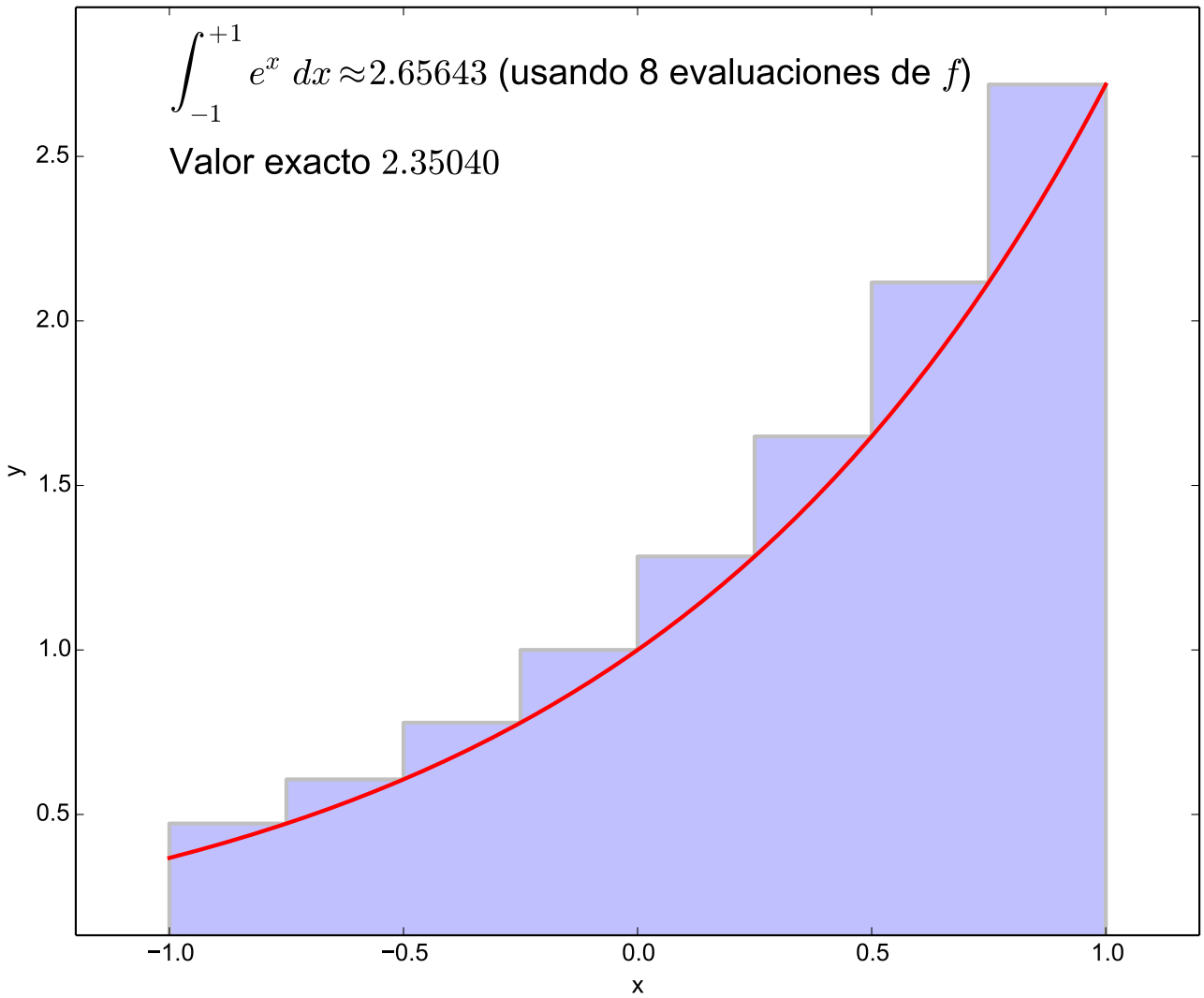
Suma de Riemann
Suma Derecha de Riemann

$$c = \int_a^b f(x)\,\mathrm{d}x = \sup_P \sum_{i=0}^{N-1} f(x_{i+1})(x_{i+1} - x_i)$$



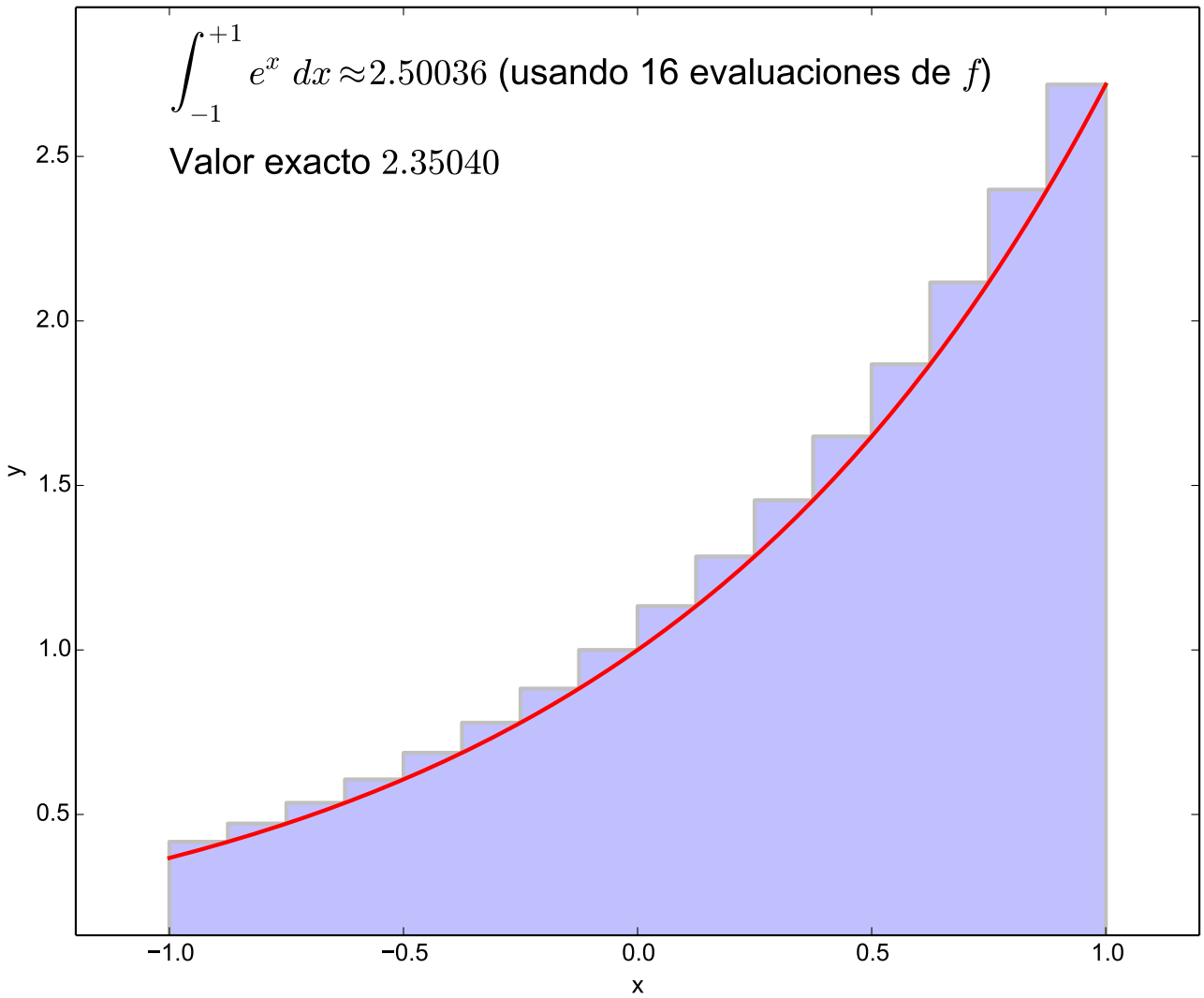
Suma de Riemann
Suma Derecha de Riemann

$$c = \int_a^b f(x)\,\mathrm{d}x = \sup_P \sum_{i=0}^{N-1} f(x_{i+1})(x_{i+1} - x_i)$$



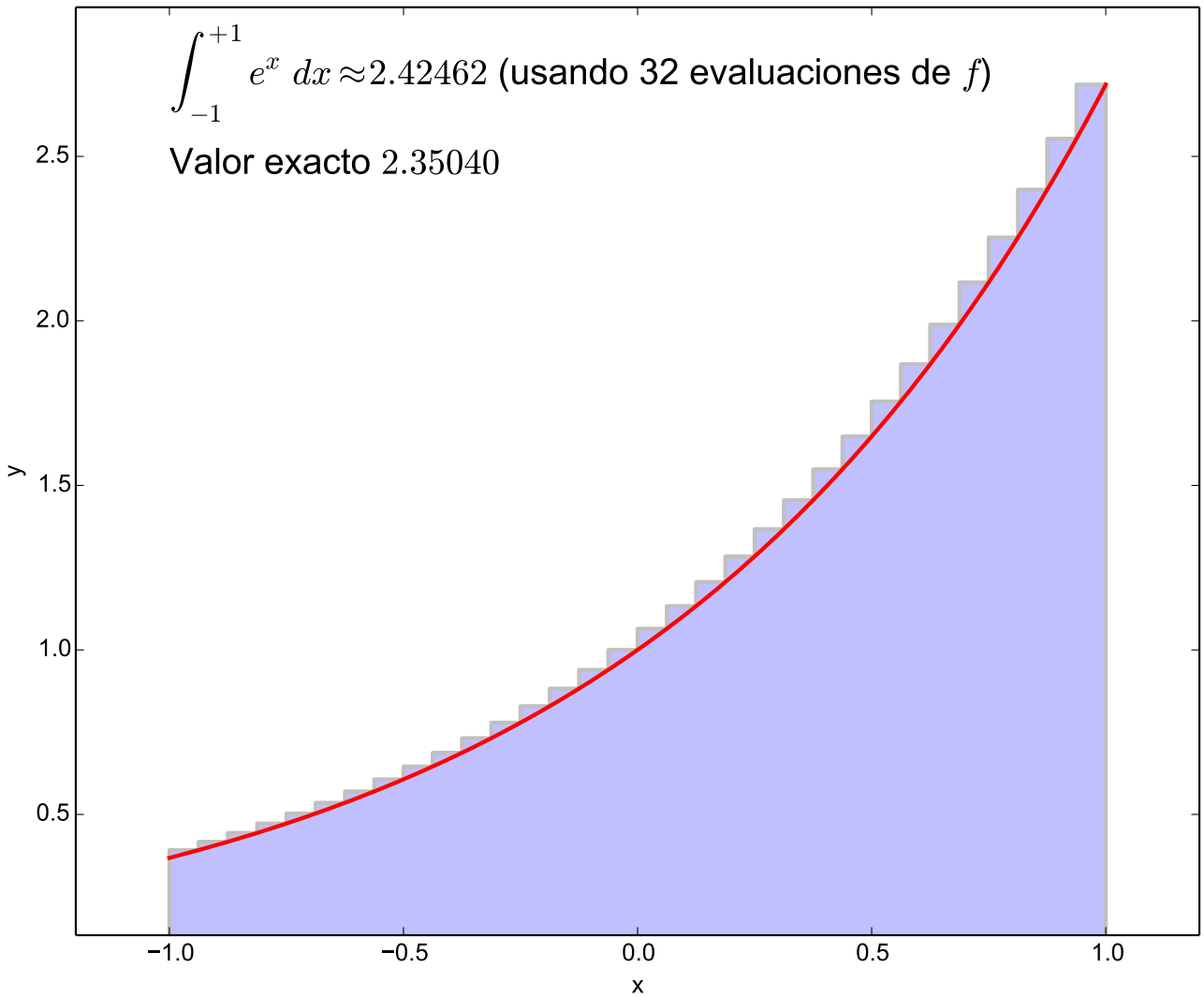
Suma de Riemann
Suma Derecha de Riemann

$$c = \int_a^b f(x)\,\mathrm{d}x = \sup_P \sum_{i=0}^{N-1} f(x_{i+1})(x_{i+1} - x_i)$$



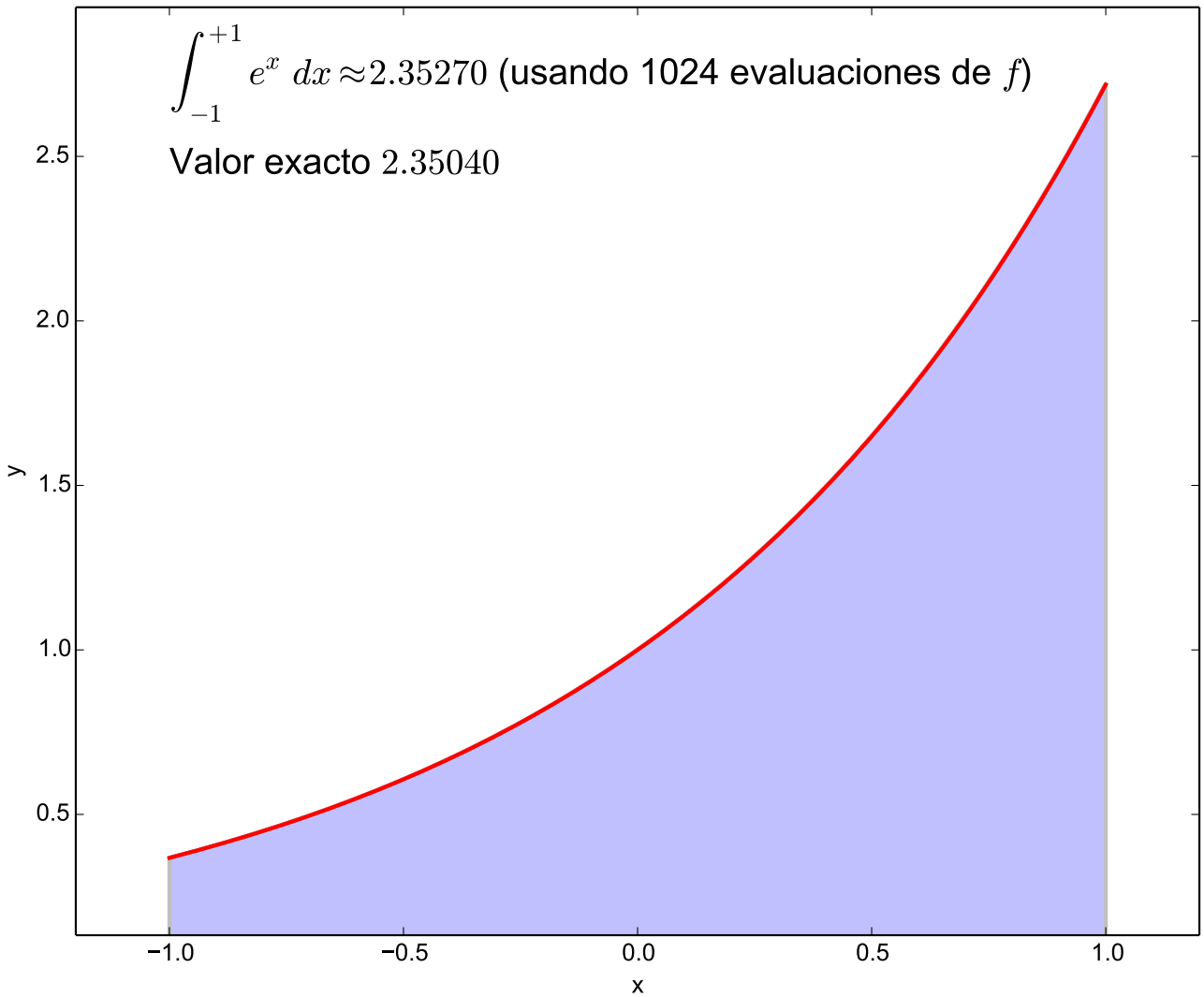
Suma de Riemann
Suma Derecha de Riemann

$$c = \int_a^b f(x)\,\mathrm{d}x = \sup_P \sum_{i=0}^{N-1} f(x_{i+1})(x_{i+1} - x_i)$$



Suma de Riemann
Suma Derecha de Riemann

$$c = \int_a^b f(x)\,\mathrm{d}x = \sup_P \sum_{i=0}^{N-1} f(x_{i+1})(x_{i+1} - x_i)$$



Suma de Riemann

Una primera aproximación al verdadero valor de la integral puede obtenerse al tomar un valor pequeño y finito para δx, como en las ilustraciones anteriores:

$$c = \int_a^b f(x)\mathrm{d}x \approx \sum f(x_i)\Delta x$$
$$= \sum f(x_{i+1})\Delta x$$

donde $\Delta x = x_{i+1} - x_i$.

Fórmulas de Newton-Cotes

- Las fórmulas de Newton-Cotes se definen para ser exactas para polinomios de grado n.
- **Idea:** se tiene puntos $(x_0, f(x_0)), (x_1, f(x_1)), \ldots, (x_n, f(x_n))$ y se supone que son interpolados por polinomios, puesto que la integral de polinomios puede ser estimada fácilmente.
- Se considerarán las siguientes reglas:
  - **Regla del punto medio**, integrando polinomios de grado 0 (constantes).
  - **Regla del trapecio**, integrando polinomios de grado 1 (rectas).
  - **Regla de Simpsons**, integrando polinomios de grado 2 (parábolas).

Regla del Punto Medio
Midpoint Rule

Se sabe que para una constante se tiene:

$$\int_{x_0}^{x_1} c\,\mathrm{d}x = (x_1 - x_0)\,c$$

Si se supone que $f(x)$ es constante e igual al valor correspontiente al punto medio del intervalo, se tiene:

$$\int_{x_0}^{x_1} f(x)\mathrm{d}x \approx (x_1 - x_0) f(x_*), \quad x_* = \frac{1}{2}(x_0 + x_1)$$

Regla del Punto Medio
Error Directo

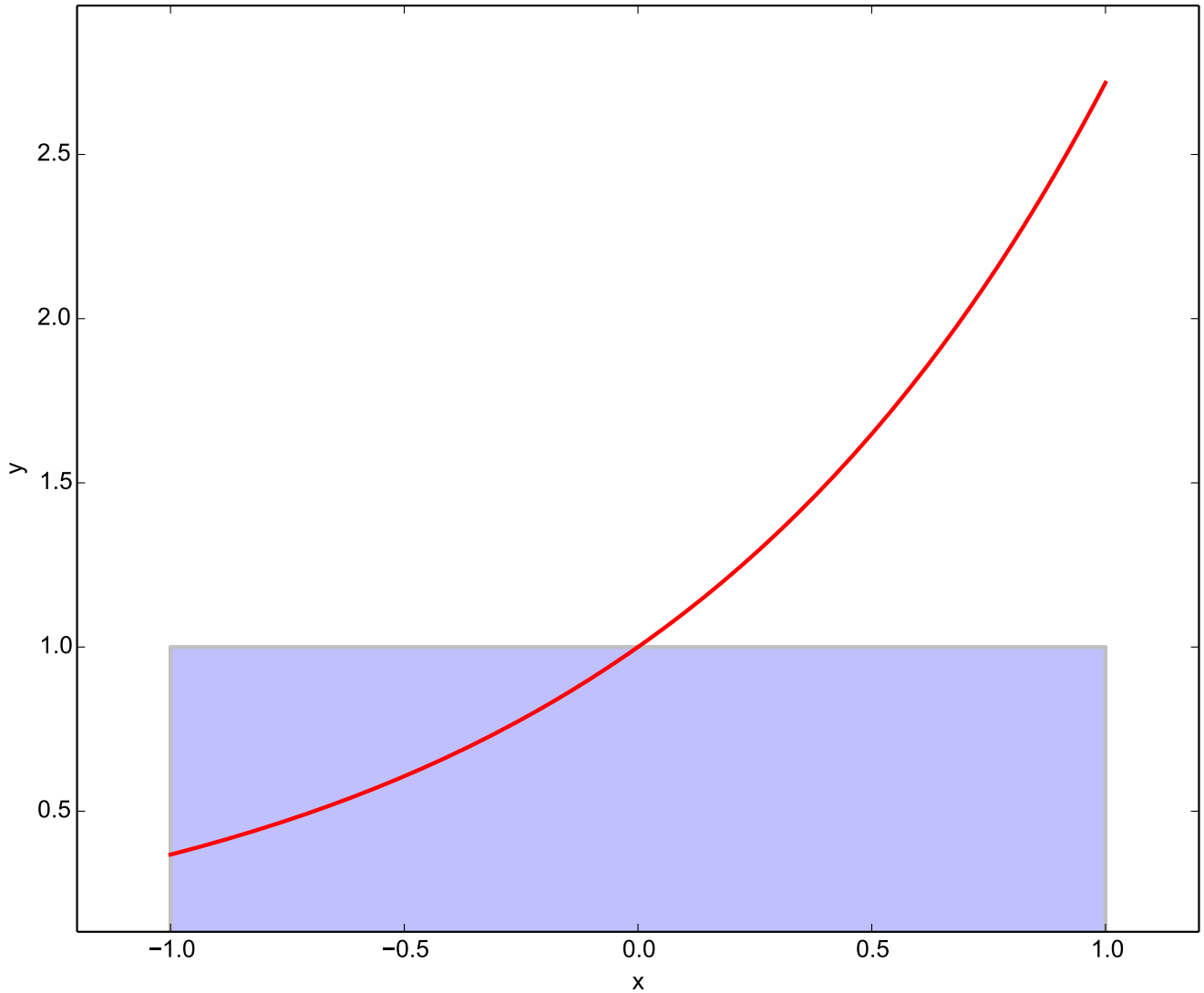
La regla del punto medio tiene un error asociado:

$$\int_{x_0}^{x_1} f(x)\mathrm{d}x = (x_1 - x_0)f(x_*) + \frac{h^3}{24}f''(c)$$

donde

- $h = (x_1 - x_0)$
- $x_* = \frac{1}{2}(x_1 + x_0)$
- $c \in [x_0, x_1]$

Regla del Punto Medio
Error Directo



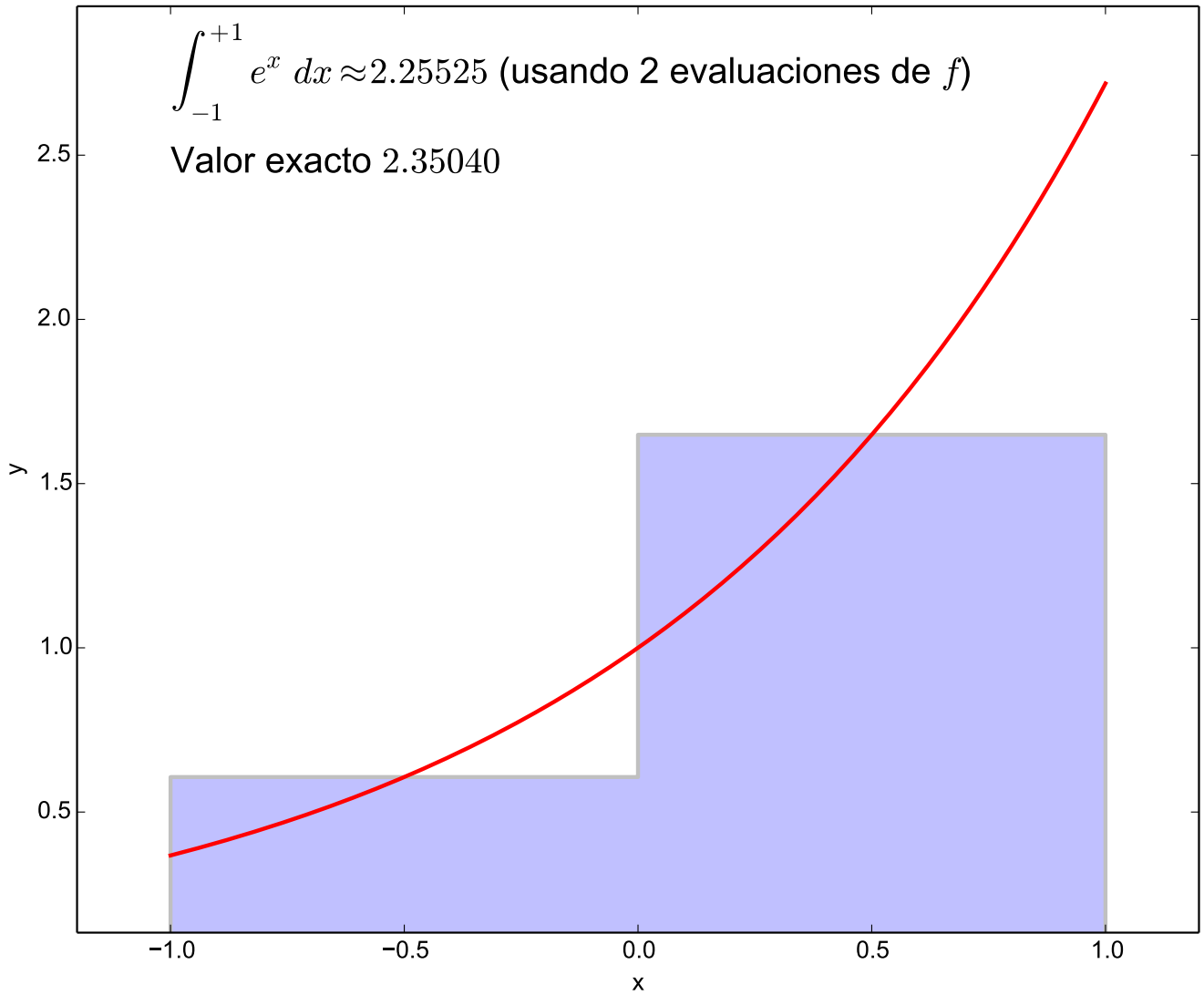
Regla del Punto Medio
Error Compuesto

Al aplicar a un intervalo $[a, b]$ subdividido en m segmentos y $m + 1$ puntos, $x_0, x_1, \ldots, x_m$ se tiene:

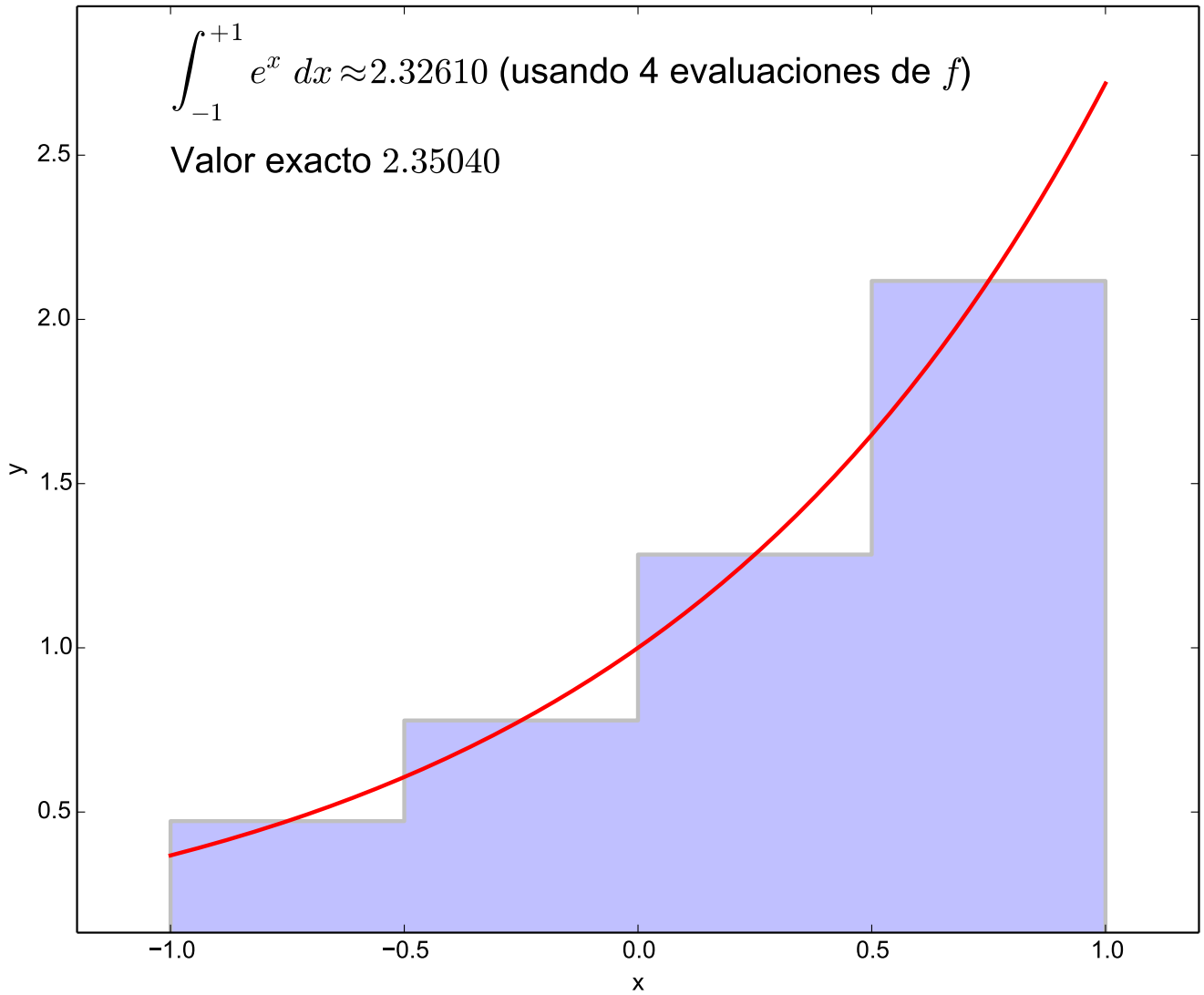
$$\begin{aligned}\int_a^b f(x)\mathrm{d}x &= \sum_{i=1}^{m} \int_{x_{i-1}}^{x_i} f(x)\mathrm{d}x \\ &= h\sum_{i=1}^{m} f(w_i) + (b-a)\frac{h^2}{24} f''(c)\end{aligned}$$

donde $h = \dfrac{(b-a)}{m}$, $w_i = \frac{1}{2}(x_{i-1} + x_i)$ y $c \in [a, b]$.

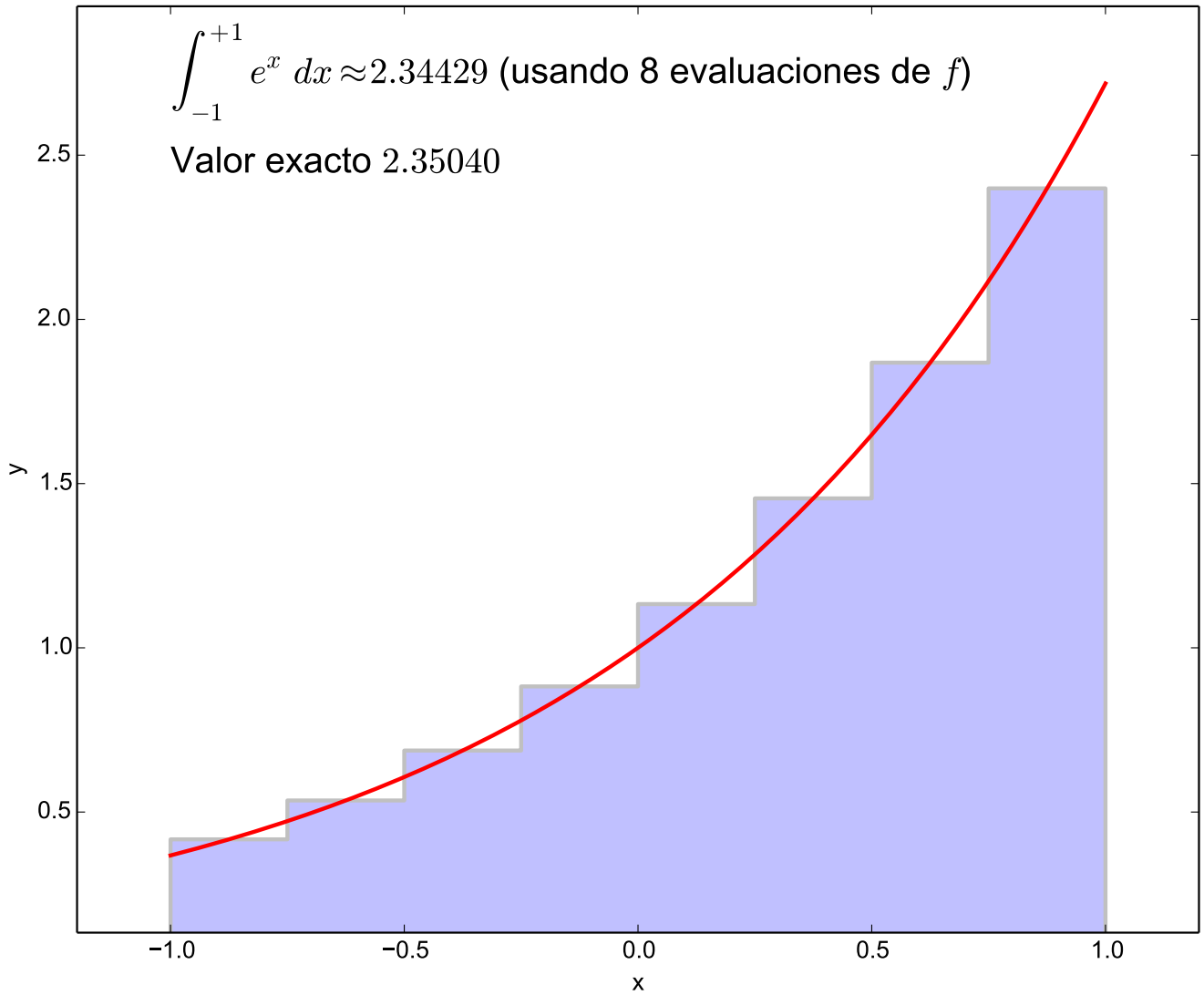
Regla del Punto Medio
Error Compuesto



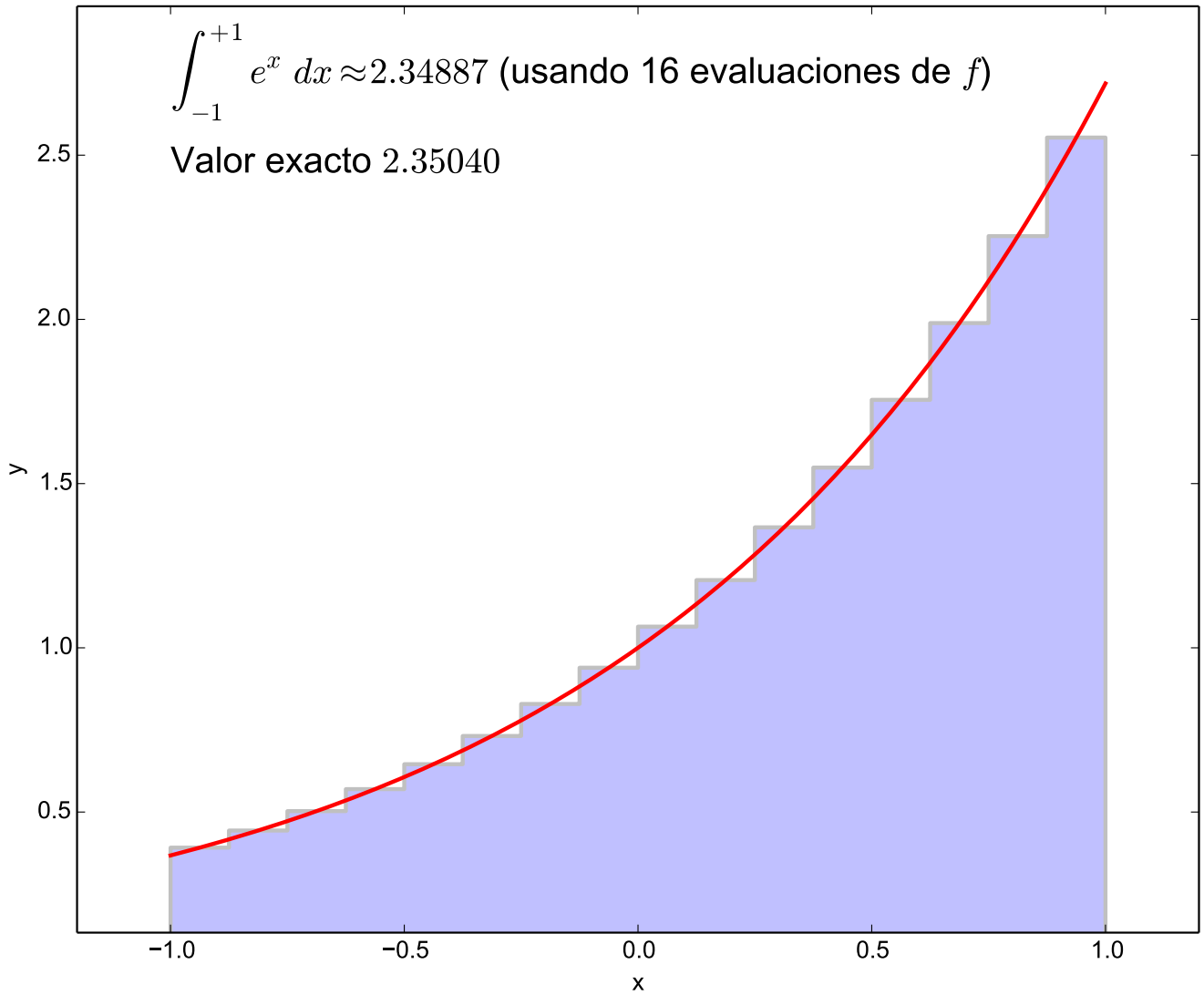
Regla del Punto Medio
Error Compuesto



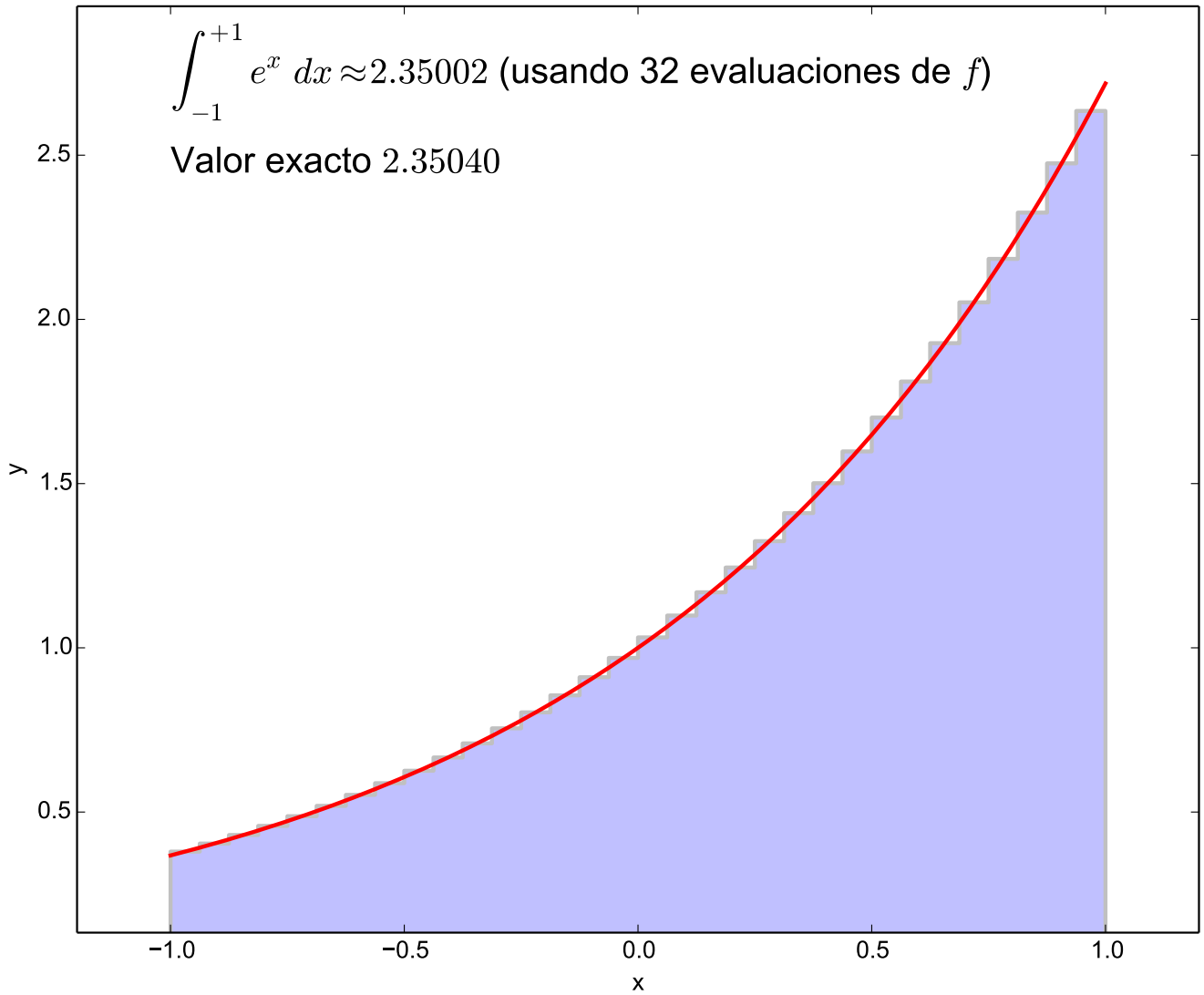
Regla del Punto Medio
Error Compuesto



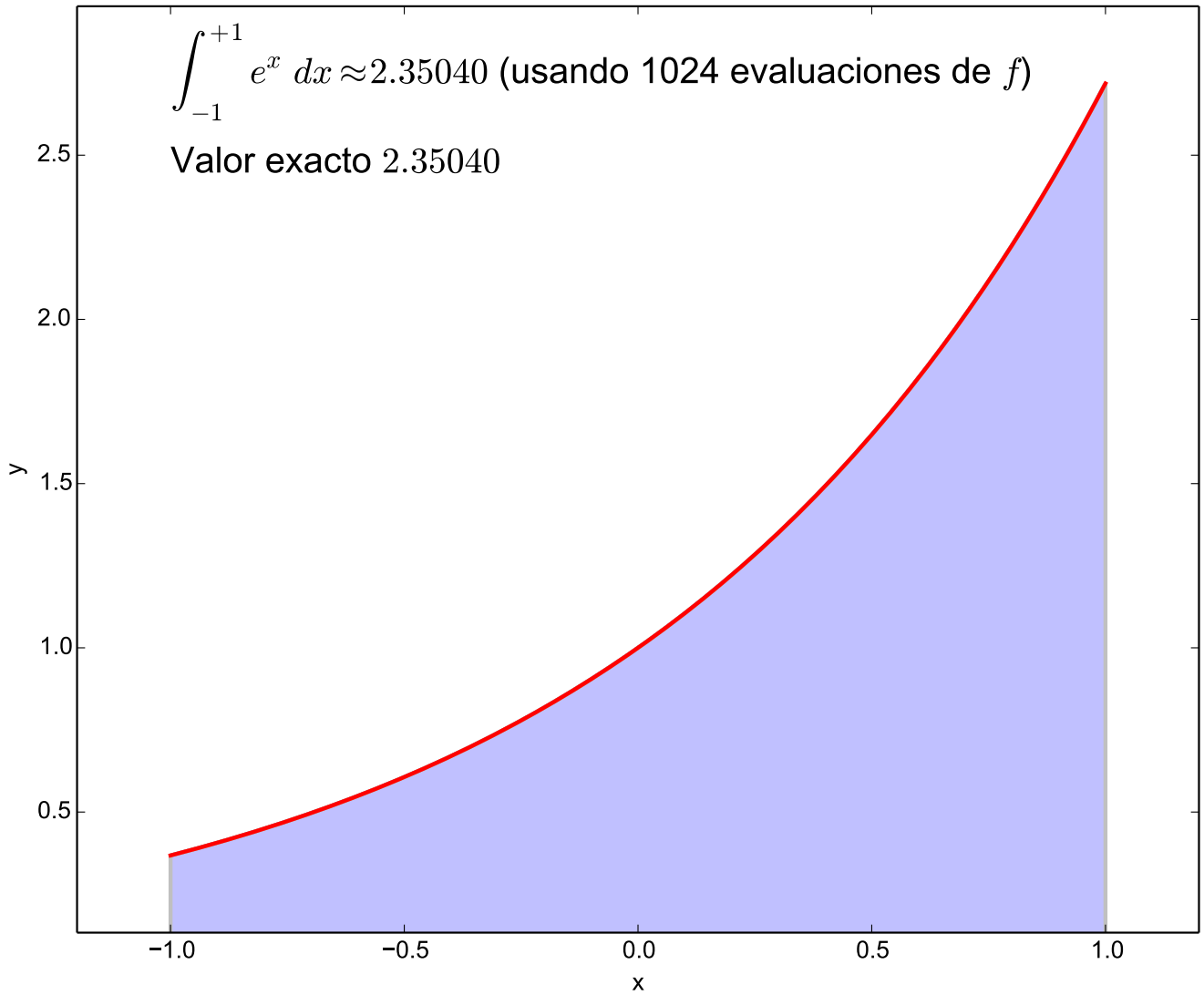
Regla del Punto Medio
Error Compuesto



Regla del Punto Medio
Error Compuesto



Regla del Punto Medio
Error Compuesto



Regla del Punto Medio
Algoritmo

```
import numpy as np

def midpoint(f, N, a, b):
  x = np.linspace(a, b, N+1)
  dx = x[1]-x[0]
  midpoints = x[:-1] + .5*dx
  int_val = dx * sum( f(midpoints) )
  return int_val
```

Pregunta:
¿Es posible escribir el algoritmo como un producto punto entre 2 vectores?
Si fuera posible, ¿cuáles serían?

Regla del Trapecio
Trapezoid Rule

Se sabe que para una recta se tiene

$$\int_{x_0}^{x_1} (mx + b)\mathrm{d}x = (x_1 - x_0)\frac{(m\,x_0 + b) + (m\,x_1 + b)}{2}$$

Si se supone que $f(x)$ es una recta que pasa por los valores extremos del intervalo, se tiene:

$$\int_{x_0}^{x_1} f(x)\mathrm{d}x \approx (x_1 - x_0)\frac{1}{2}(f(x_0) + f(x_1))$$

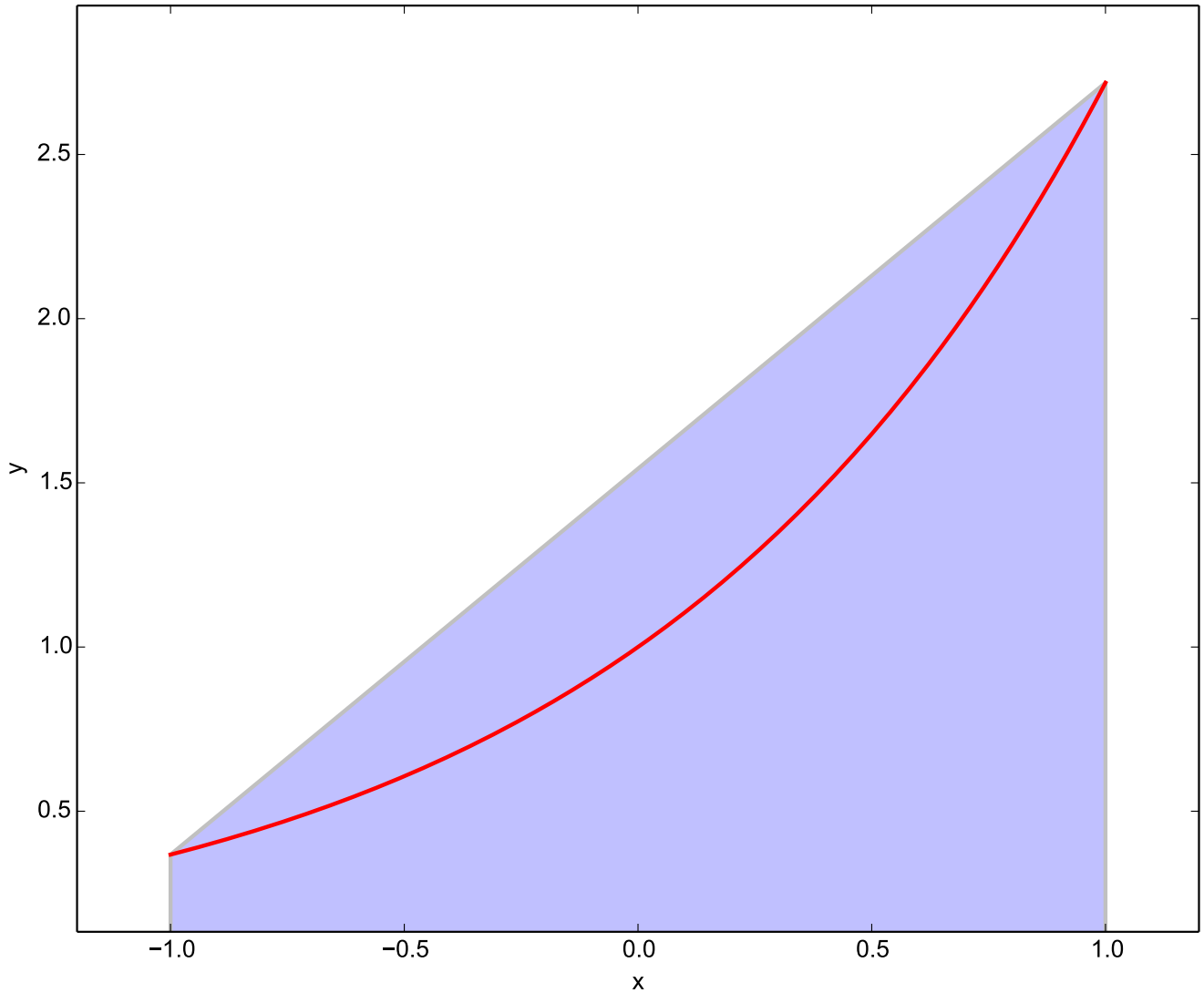
Regla del Trapecio
Error Directo

La regla del trapecio tiene un error asociado:

$$\int_{x_0}^{x_1} f(x)\mathrm{d}x = \frac{h}{2}\left(f(x_0) + f(x_1)\right) - \frac{h^3}{12}f''(c)$$

- $h = (x_1 - x_0)$
- $c \in [x_0, x_1]$

Regla del Trapecio
Error Directo



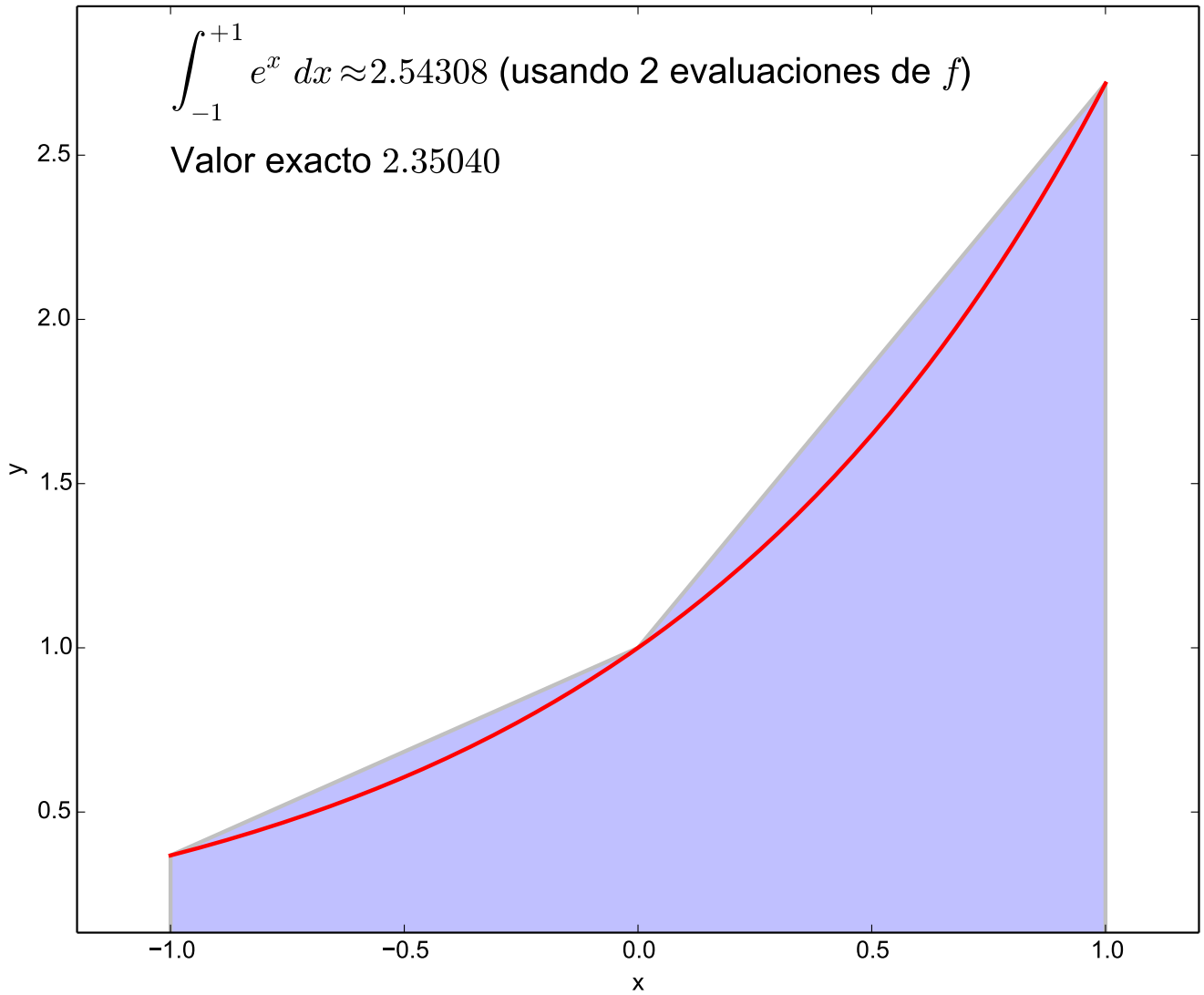
Regla del Trapecio
Error Compuesto

Al aplicar a un intervalo $[a, b]$ subdividido en m segmentos y $m + 1$ puntos, $x_0, \ldots, x_m$ se tiene

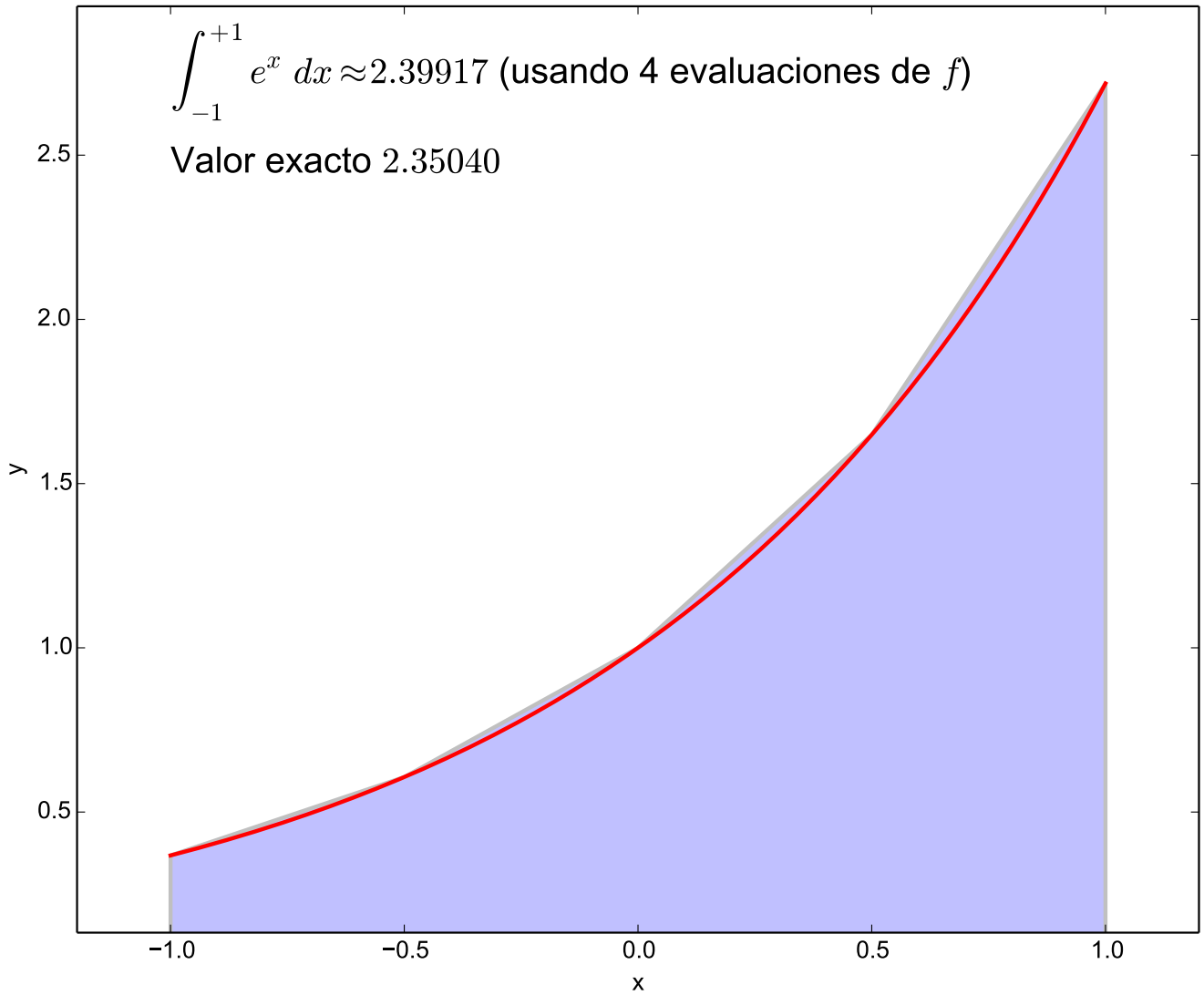
$$\begin{aligned}\int_a^b f(x)\mathrm{d}x &= \sum_{i=1}^{m} \int_{x_{i-1}}^{x_i} f(x)\mathrm{d}x \\ &= \frac{h}{2}\left[f(a) + f(b) + 2\sum_{i=1}^{m-1} f(x_i)\right] - (b-a)\frac{h^2}{12} f''(c)\end{aligned}$$

donde $h = \dfrac{(b-a)}{m}$ y $c \in [a, b]$

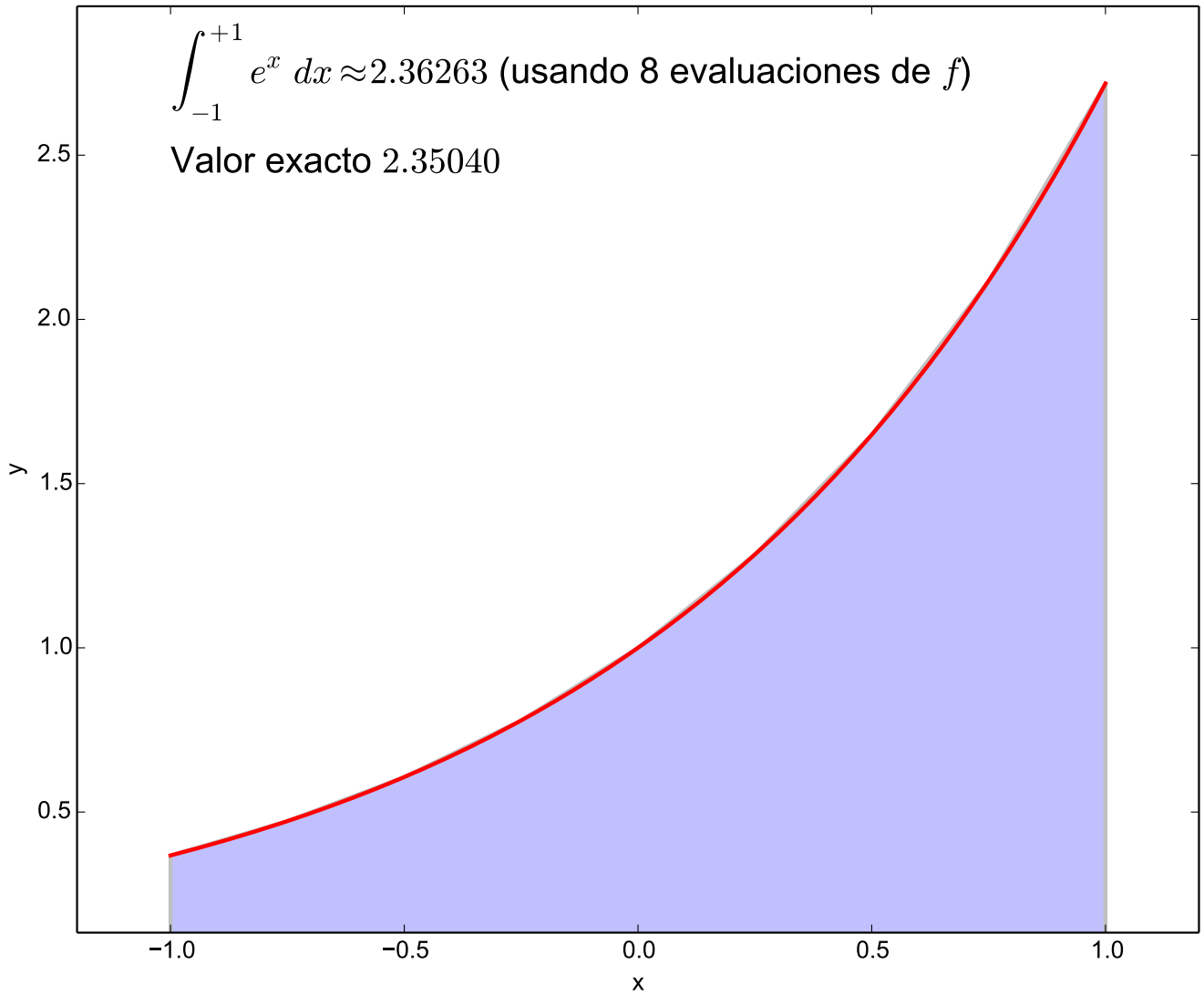
Regla del Trapecio
Error Compuesto



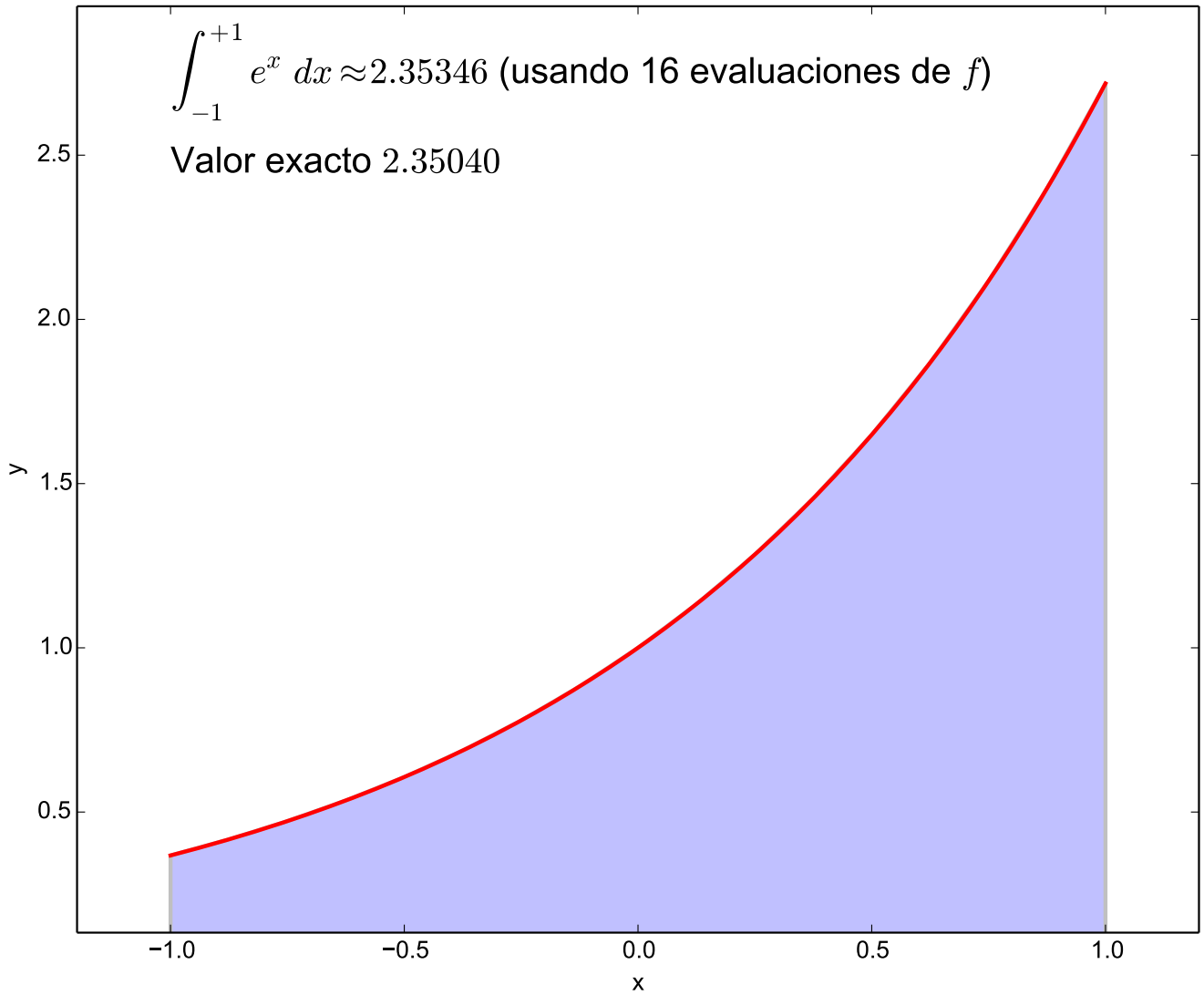
Regla del Trapecio
Error Compuesto



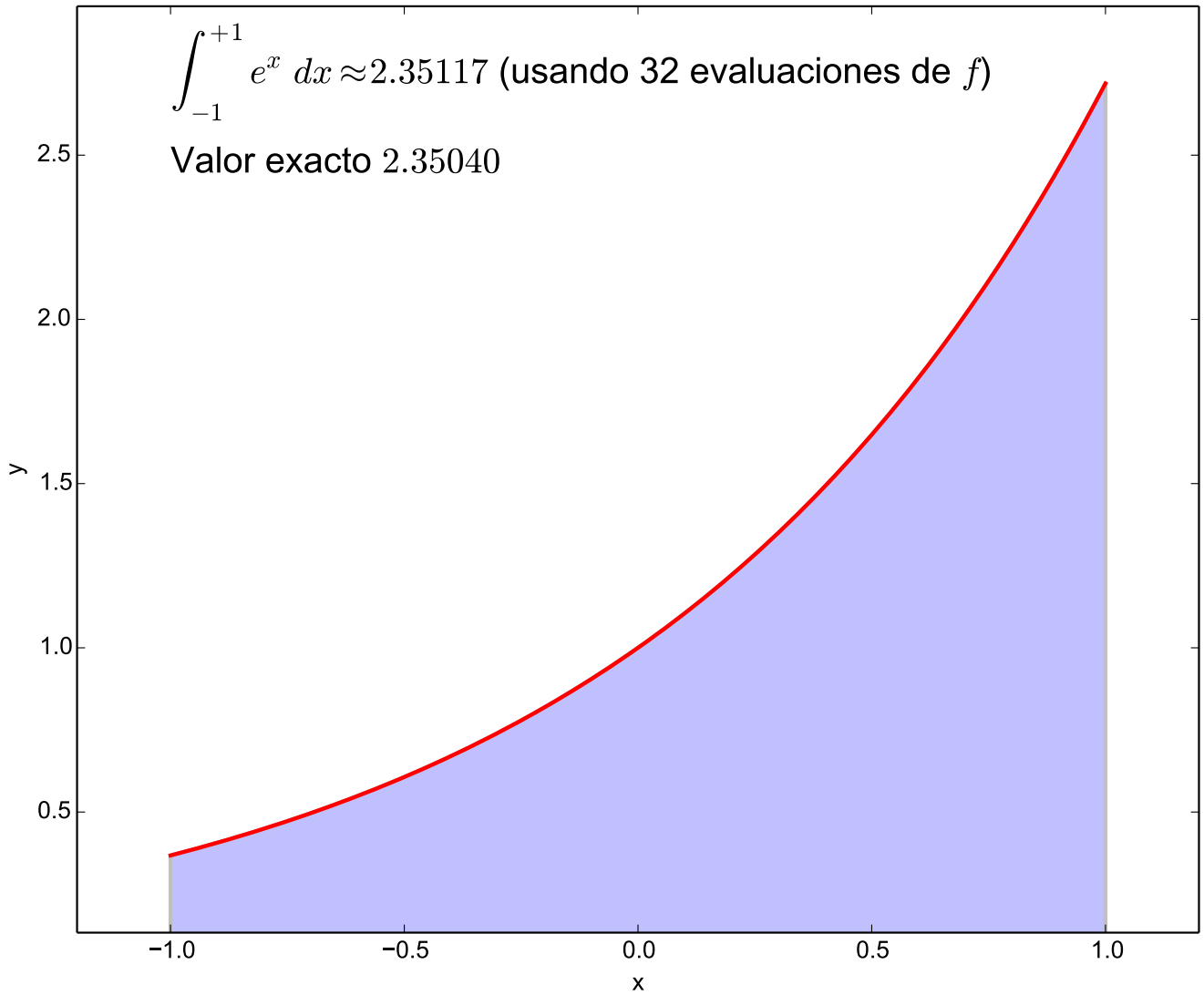
Regla del Trapecio
Error Compuesto



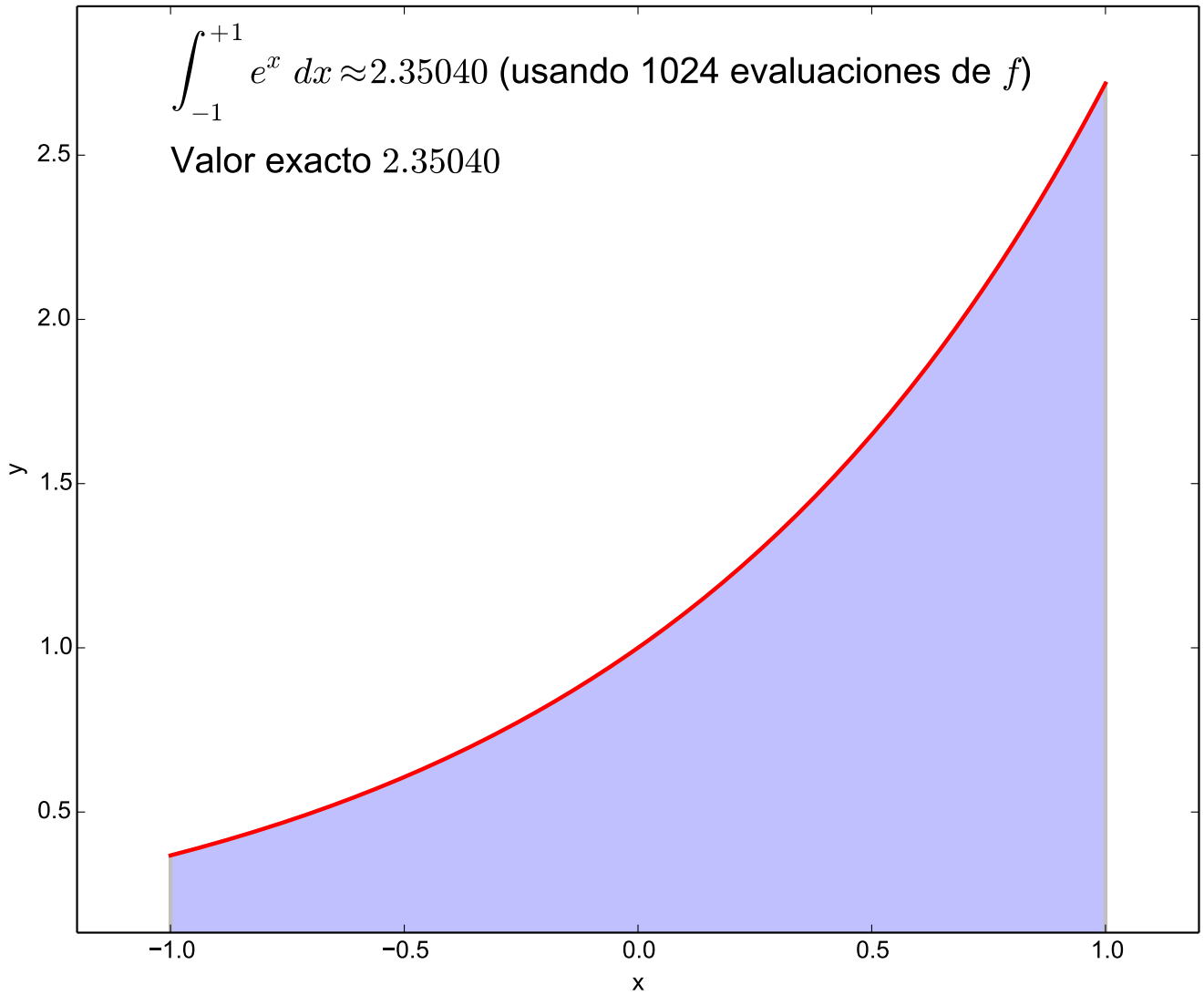
Regla del Trapecio
Error Compuesto



Regla del Trapecio
Error Compuesto



Regla del Trapecio
Error Compuesto



Regla del Trapecio
Algoritmo

```
import numpy as np

def trapezoid(f, N, a, b):
  x = np.linspace(a, b, N+1)
  dx = x[1]-x[0]
  xleft = x[:-1]
  xright = x[1:]
  int_val = 0.5 * dx * sum( f(xleft) + f(xright) )
  return int_val
```

Pregunta:
¿Es posible escribir el algoritmo como un producto punto entre 2 vectores?
Si fuera posible, ¿cuáles serían?

Regla de Simpson
Simpson's Rule

Es posible probar que para una parábola se tiene:

$$\int_{x_0}^{x_2}(ax^2+bx+c)\mathrm{d}x = \frac{a}{3}(x_2^3-x_0^3)+\frac{b}{2}(x_2^2-x_0^2)+a(x_2-x_0)$$

$$=(x_1-x_0)\frac{1}{3}\left[(ax_0^2+bx_0+c)+4(ax_1^2+bx_1+c)+(ax_2^2+bx_2+c)\right]$$

donde $x_1=\dfrac{(x_0+x_2)}{2}$.

Si se supone que $f(x)$ es una parábola que pasa por los valores en los extremos y punto medio del intervalo, se tiene:

$$\int_{x_0}^{x_2} f(x)\mathrm{d}x \approx (x_1-x_0)\frac{1}{2}(f(x_0)+4f(x_1)+f(x_2))$$

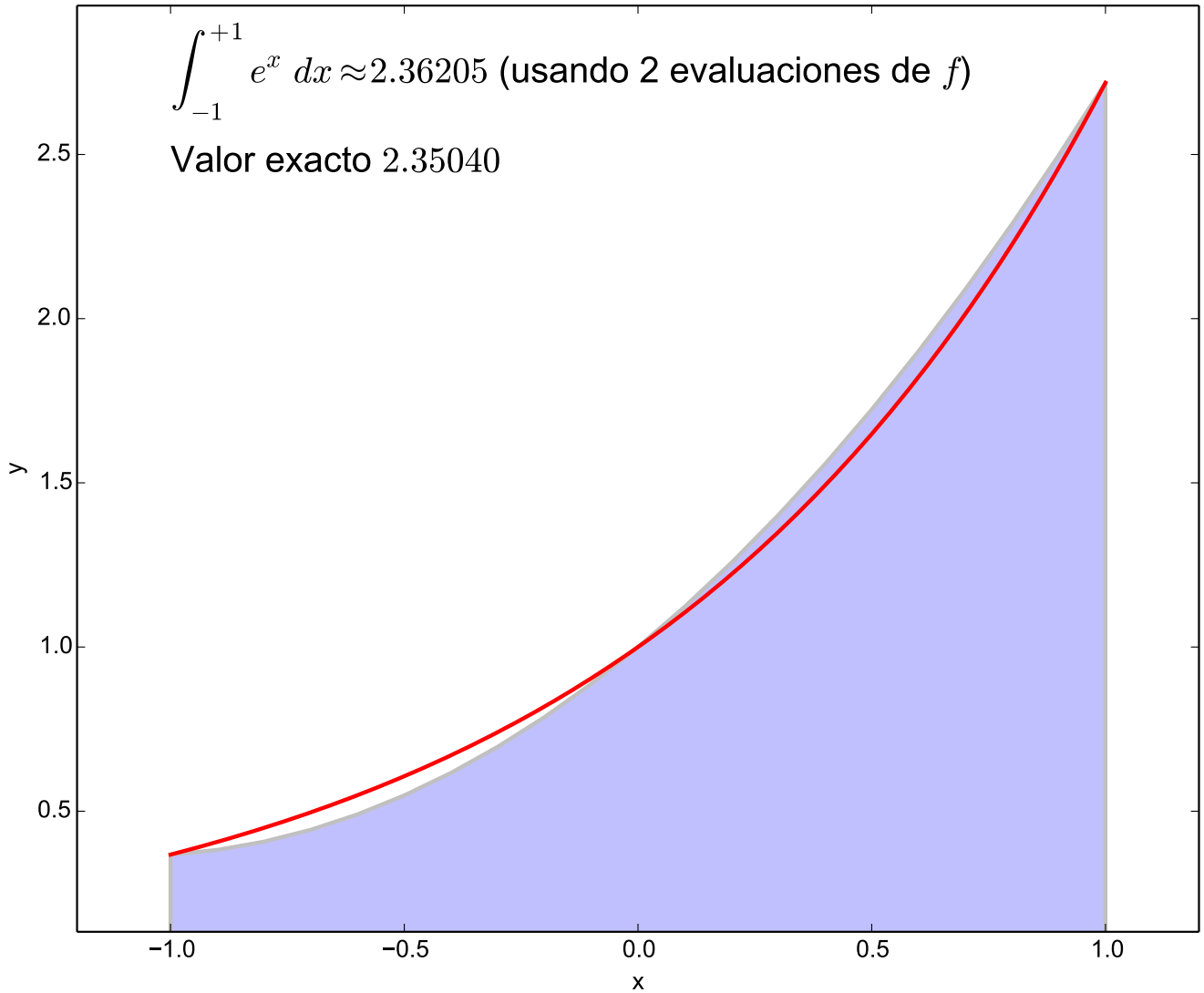
Regla de Simpson
Error Directo

La regla de Simpson tiene un error asociado:

$$\int_{x_0}^{x_2} f(x)\mathrm{d}x = \frac{h}{3}(f(x_0) + 4f(x_1) + f(x_2)) - \frac{h^5}{90}f^{(4)}(c)$$

- $h = (x_1 - x_0) = (x_2 - x_1)$ y $x_1 = \dfrac{(x_0 + x_2)}{2}$
- $c \in [x_0, x_2]$

Regla de Simpson
Error Directo



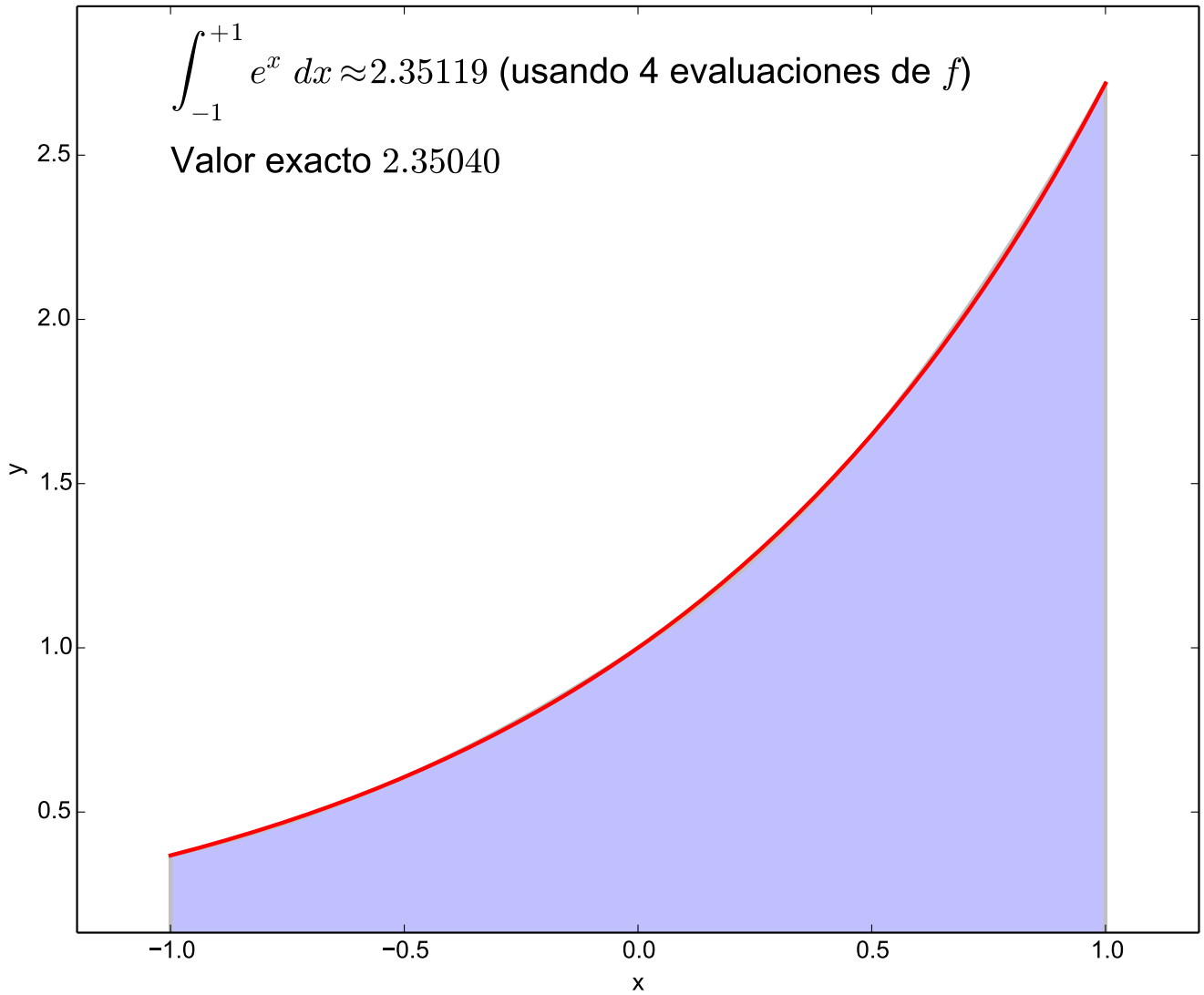
Regla de Simpson
Error Compuesto

Al aplicar a un intervalo $[a, b]$ subdividido en $N = 2m$ segmentos y $2m + 1$ puntos, $x_0, \ldots, x_N$ se tiene:

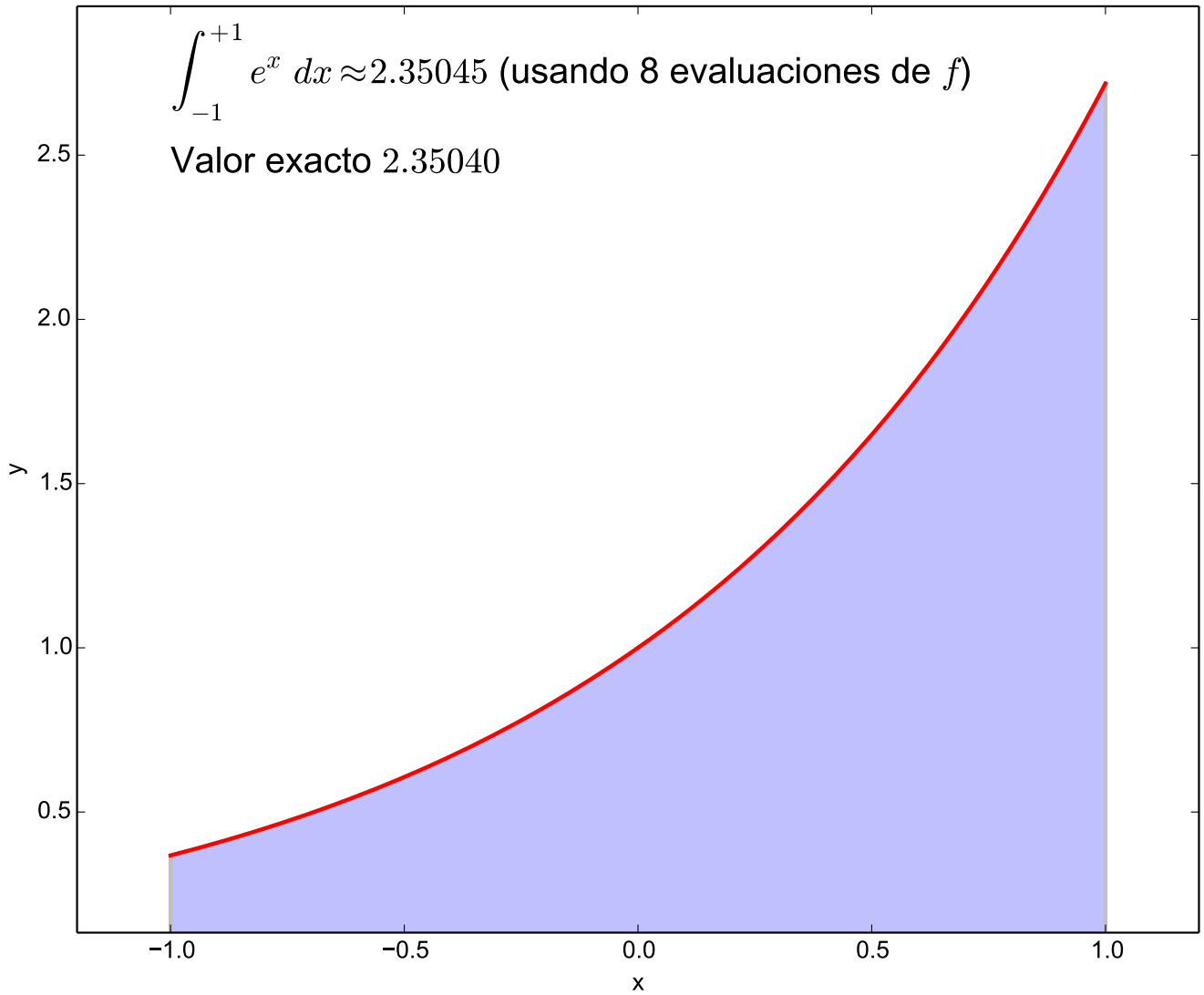
$$\int_a^b f(x)\mathrm{d}x = \frac{h}{3}\left(f(x_0) + 4\sum_{i=1}^{m} f(x_{2i-1}) + 2\sum_{i=1}^{m-1} f(x_{2i}) + f(x_N)\right) - (b-a)\frac{h^4}{180}f^{(4)}(c)$$

- $h = (x_{i+1} - x_i) = \dfrac{b-a}{N}$
- $c \in [a, b]$
- $m = \dfrac{b-a}{2h} = \dfrac{N}{2}$, i.e. N debe ser par.

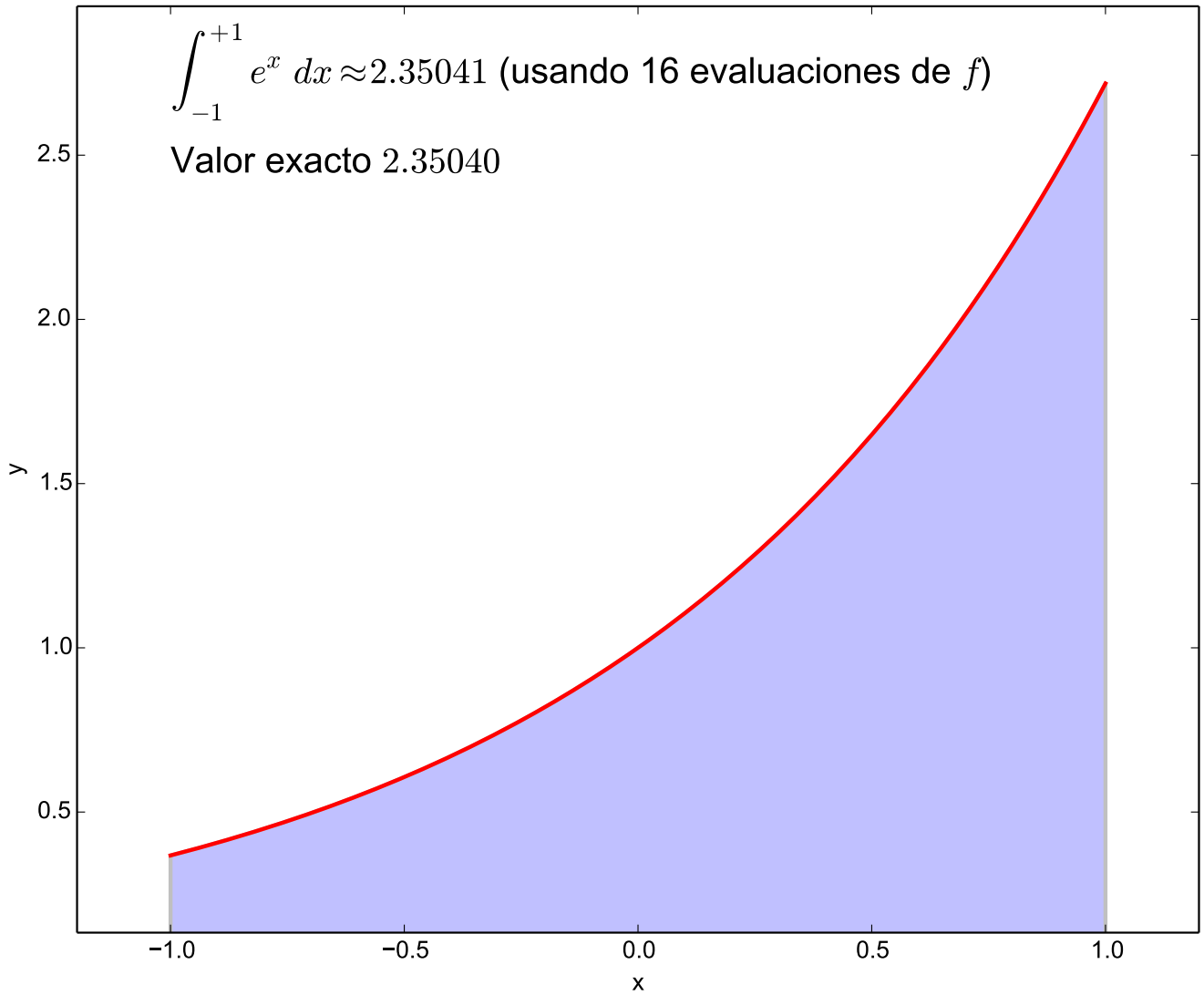
Regla de Simpson
Error Compuesto



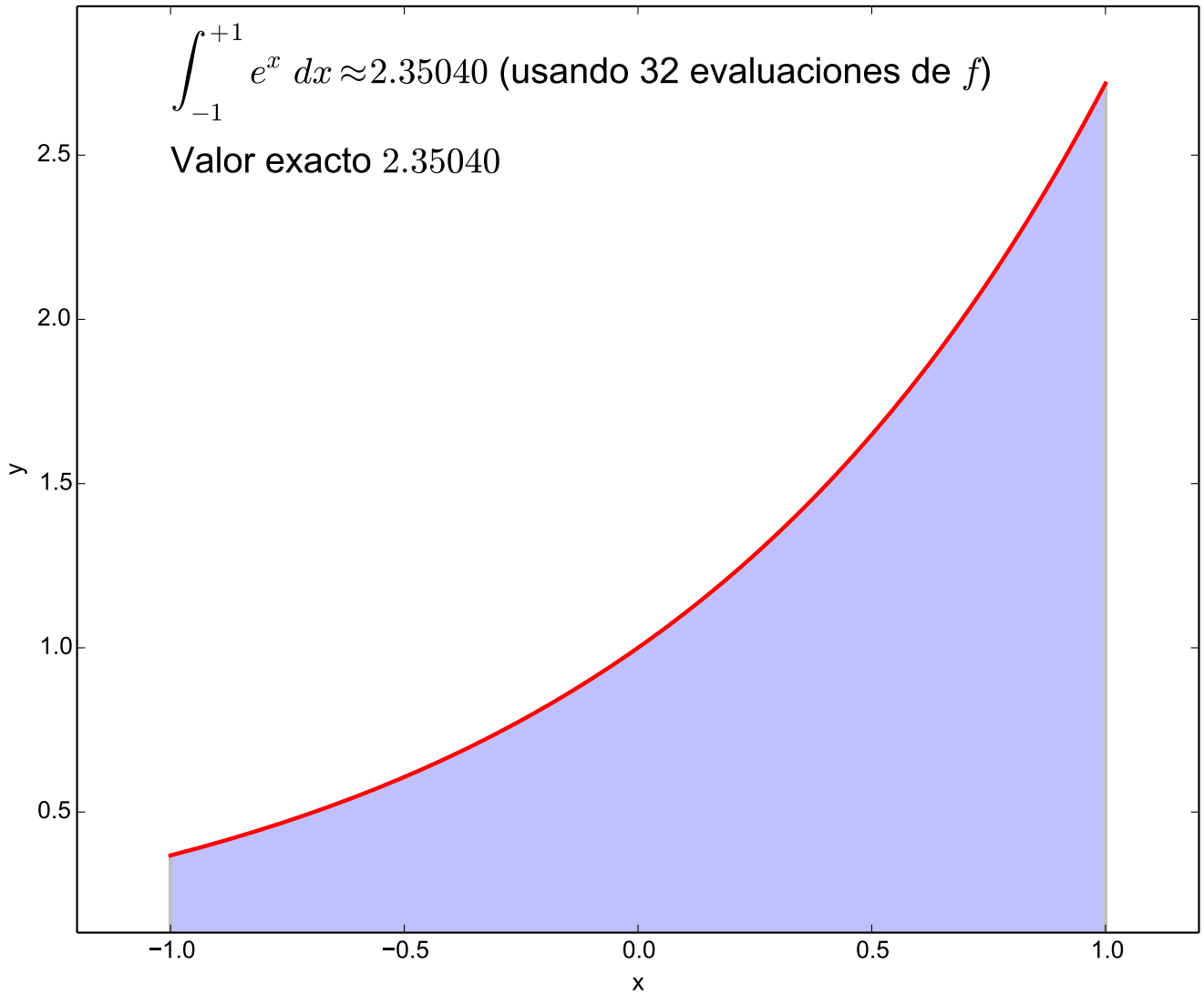
Regla de Simpson
Error Compuesto



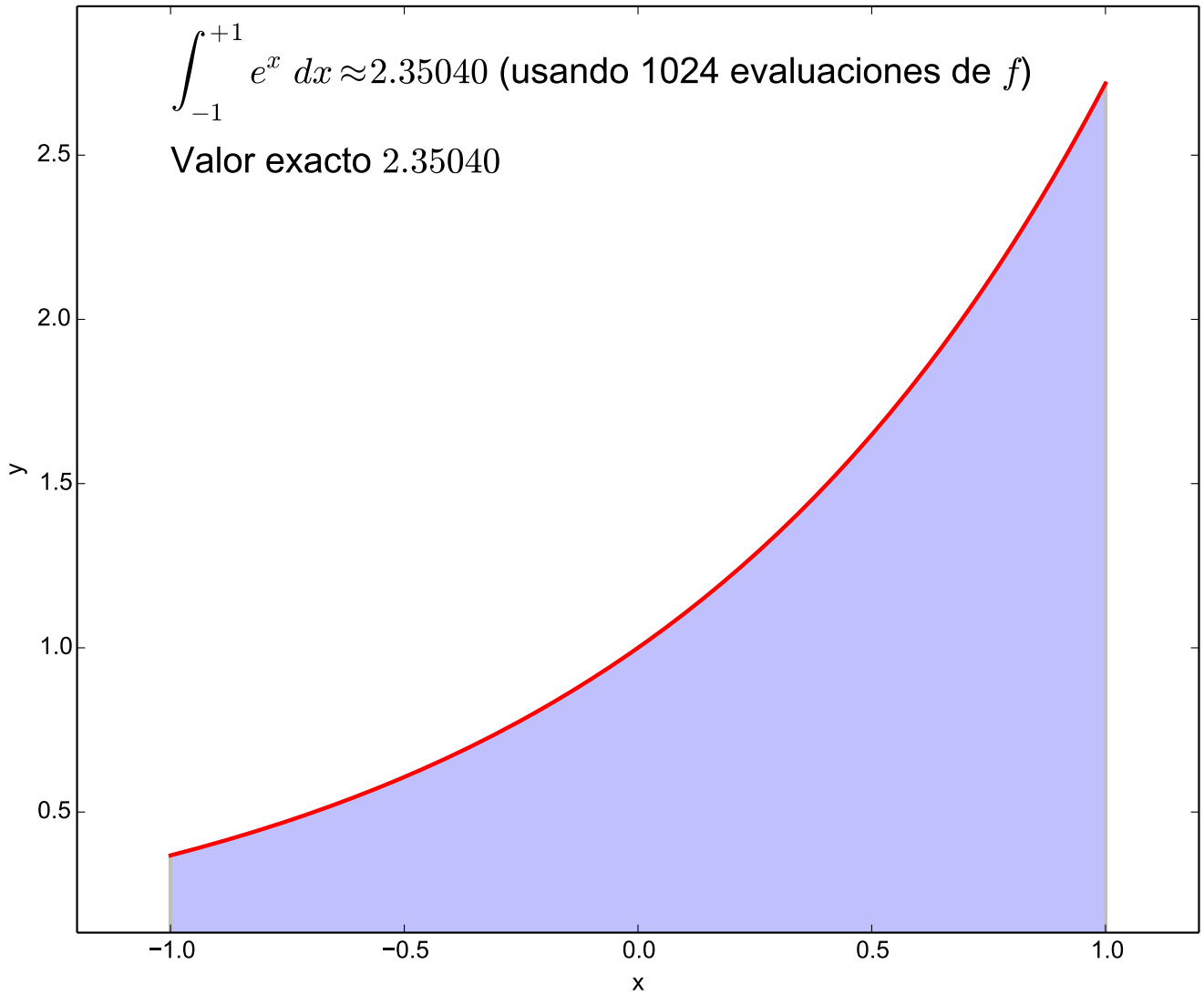
Regla de Simpson
Error Compuesto



Regla de Simpson
Error Compuesto



Regla de Simpson
Error Compuesto



Regla de Simpson
Algoritmo

```
import numpy as np

def simpsons(f, N, a, b):
  if N%2==1:
    print "Simpsons rule only applicable to even number of segments"
    return None
  x = np.linspace(a, b, N+1)
  dx = x[1]-x[0]
  xleft   = x[:-2:2]
  xmiddle = x[1::2]
  xright  = x[2::2]
  int_val = (dx/3) * sum( f(xleft) + 4*f(xmiddle) + f(xright) )
  return int_val
```

Pregunta:

¿Es posible escribir el algoritmo como un producto punto entre 2 vectores?
Si fuera posible, ¿cuáles serían?

Cuadratura Gaussiana

Motivación: si $f(x)$ pudiera ser evaluada en cualquier punto, ¿cuál es la mejor forma de aproximar la integral?

$$\int_a^b f(x)\mathrm{d}x = \sum_{i=1}^{n} w_i\, f(x_i)$$

- w_i son los pesos o ponderaciones
- x_i son los puntos donde la función será evaluada
- f puede ser una función muy costosa de calcular, por lo que se busca evaluarla lo menos posible.

Cuadratura Gaussiana

Idea: considerar que $f(x) \approx Q(x) = \sum_{i=1}^{n} L_i(x)\, f(x_i)$. La dependencia de x se encuentra sólamente en los términos $L_i(x)$. Realizando la integración:

$$\begin{aligned}
\int_{-1}^{1} f(x)\mathrm{d}x &\approx \int_{-1}^{1} Q(x)\mathrm{d}x \\
&= \int_{-1}^{1} \sum_{i=1}^{n} L_i(x) f(x_i)\mathrm{d}x \\
&= \sum_{i=1}^{n} f(x_i) \underbrace{\int_{-1}^{1} L_i(x)dx}_{w_i}
\end{aligned}$$

Cuadratura Gaussiana

Considerar:

$$L_i(x) = \prod_{j=1, j\neq i}^{n} \frac{(x - x_i)}{(x_i - x_j)}$$

$$= \frac{(x - x_1)\ldots(x - x_{i-1})(x - x_{i+1})\ldots(x - x_n)}{(x_i - x_1)\ldots(x_i - x_{i-1})(x_i - x_{i+1})\ldots(x_i - x_n)}$$

Que cumple con:

- $L_i(x_i) = 1$
- $L_i(x_j) = 0, \forall\ j \neq i$

Cuadratura Gaussiana

Los puntos x_i se definen como las raíces del n-ésimo polinomio de Legendre, $p_n(x)$:

$$p_n(x) = \frac{1}{2^n\, n!} \frac{\mathrm{d}^n}{\mathrm{d}x^n} \left[(x^2-1)^n\right]$$

Los primeros polinomios de Legendre son:

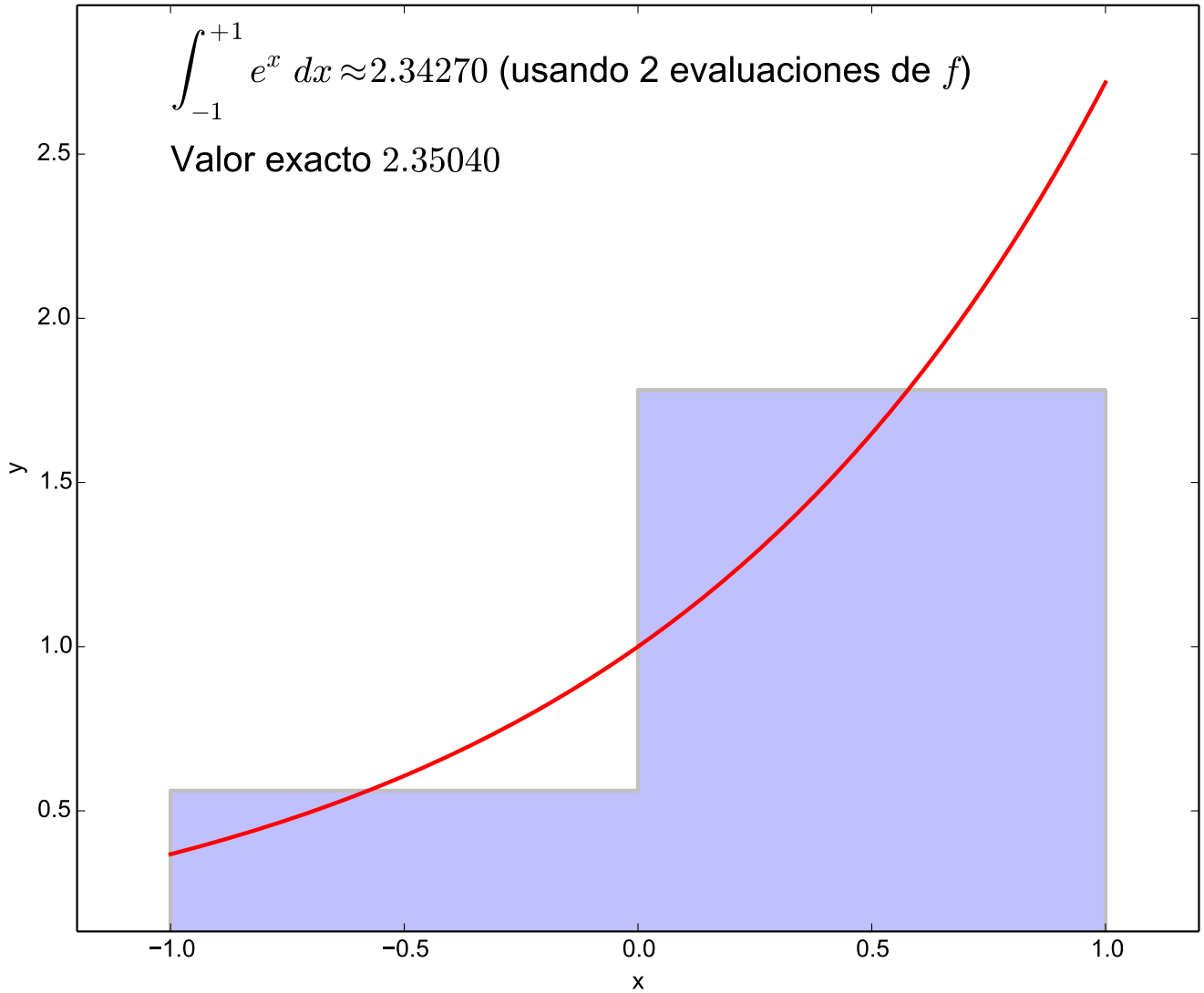
- $p_0(x) = 1$
- $p_1(x) = x$
- $p_2(x) = \frac{1}{2}(3\,x^2 - 1)$
- $p_3(x) = \frac{1}{2}(5\,x^3 - x)$
- $p_4(x) = \frac{1}{8}(35\,x^4 - 30\,x^2 + 3)$

Cuadratura Gaussiana

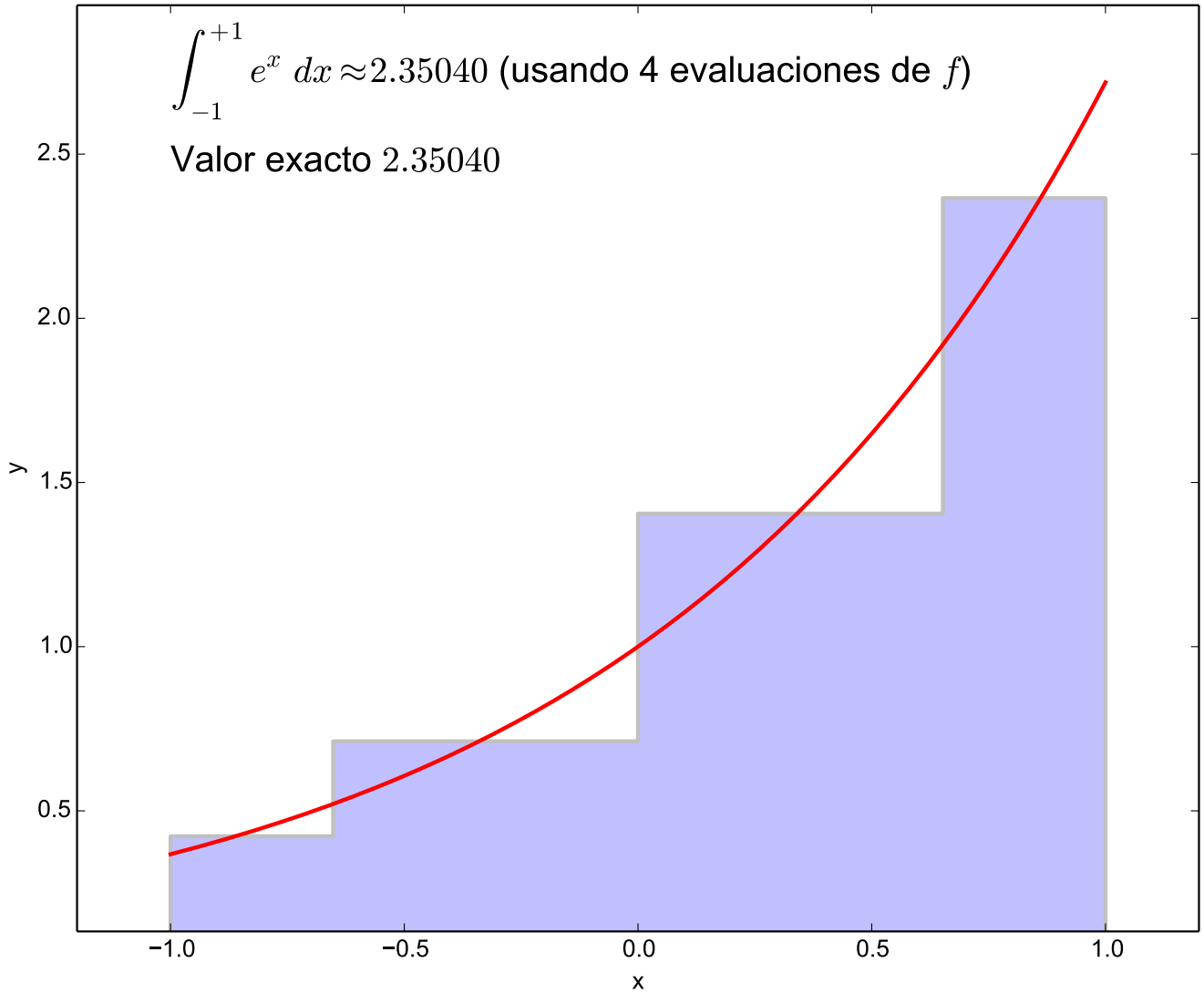
Lo anterior se resume en la siguiente tabla:

n	$p_n(x)$	x_i	w_i
2	$\frac{1}{2}(3x^2-1)$	$-\sqrt{\frac{1}{3}} \approx -0.577$	$+1.000$
		$-\sqrt{\frac{1}{3}} \approx 0.577$	$+1.000$
3	$\frac{1}{2}(5x^3-x)$	$-\sqrt{\frac{3}{5}} \approx -0.774$	$\frac{5}{9} \approx +0.555$
		0.000	$\frac{8}{9} \approx +0.888$
		$+\sqrt{\frac{3}{5}} \approx +0.774$	$\frac{5}{9} \approx +0.555$
4	$\frac{1}{8}(35x^4-30x^2+3)$	$-\sqrt{\frac{15+2\sqrt{30}}{35}} \approx -0.861$	$\frac{90-5\sqrt{30}}{180} \approx +0.347$
		$-\sqrt{\frac{15-2\sqrt{30}}{35}} \approx -0.339$	$\frac{90+5\sqrt{30}}{180} \approx +0.652$
		$+\sqrt{\frac{15-2\sqrt{30}}{35}} \approx +0.339$	$\frac{90+5\sqrt{30}}{180} \approx +0.652$
		$+\sqrt{\frac{15+2\sqrt{30}}{35}} \approx +0.861$	$\frac{90-5\sqrt{30}}{180} \approx +0.347$

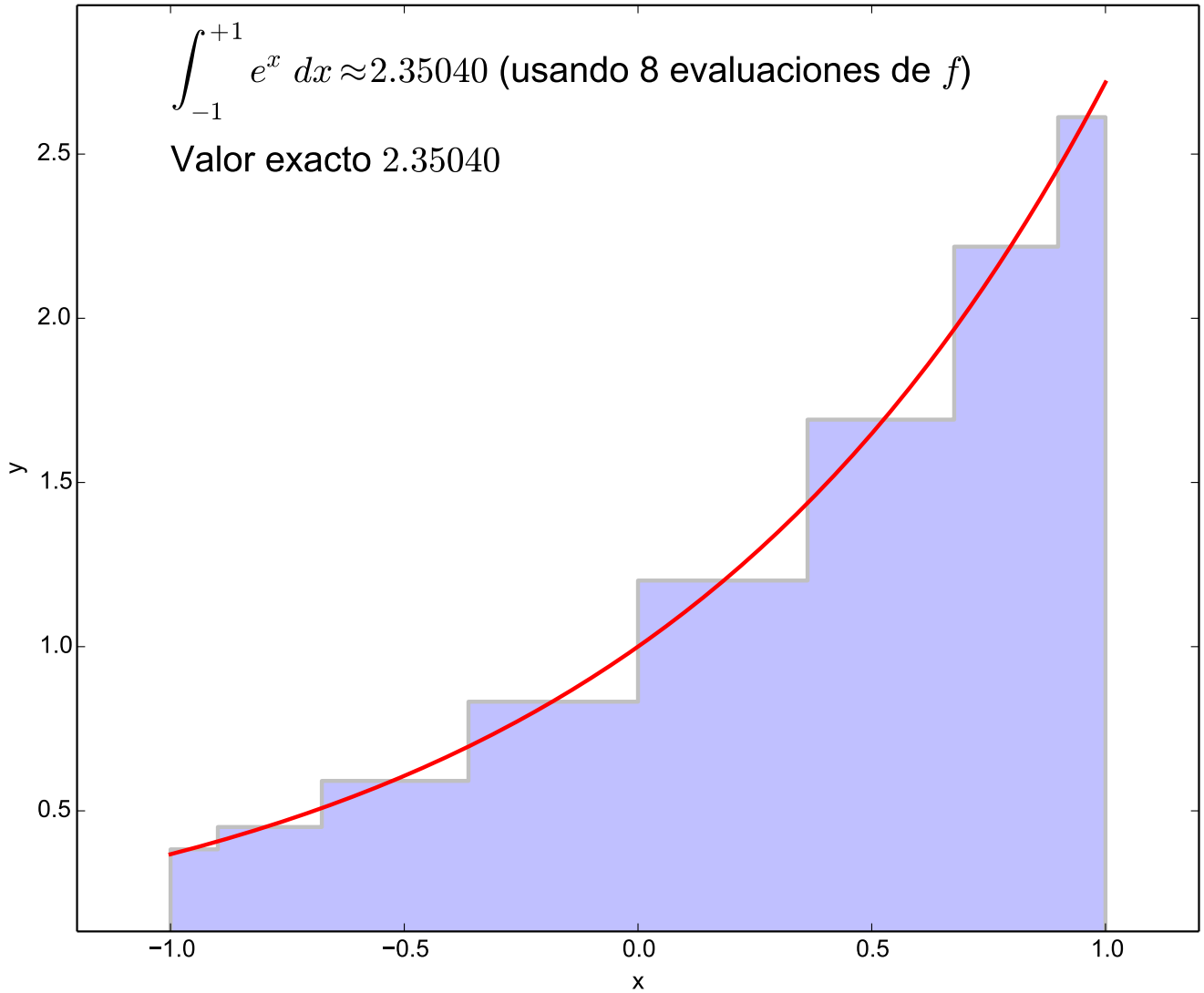
Cuadratura Gaussiana



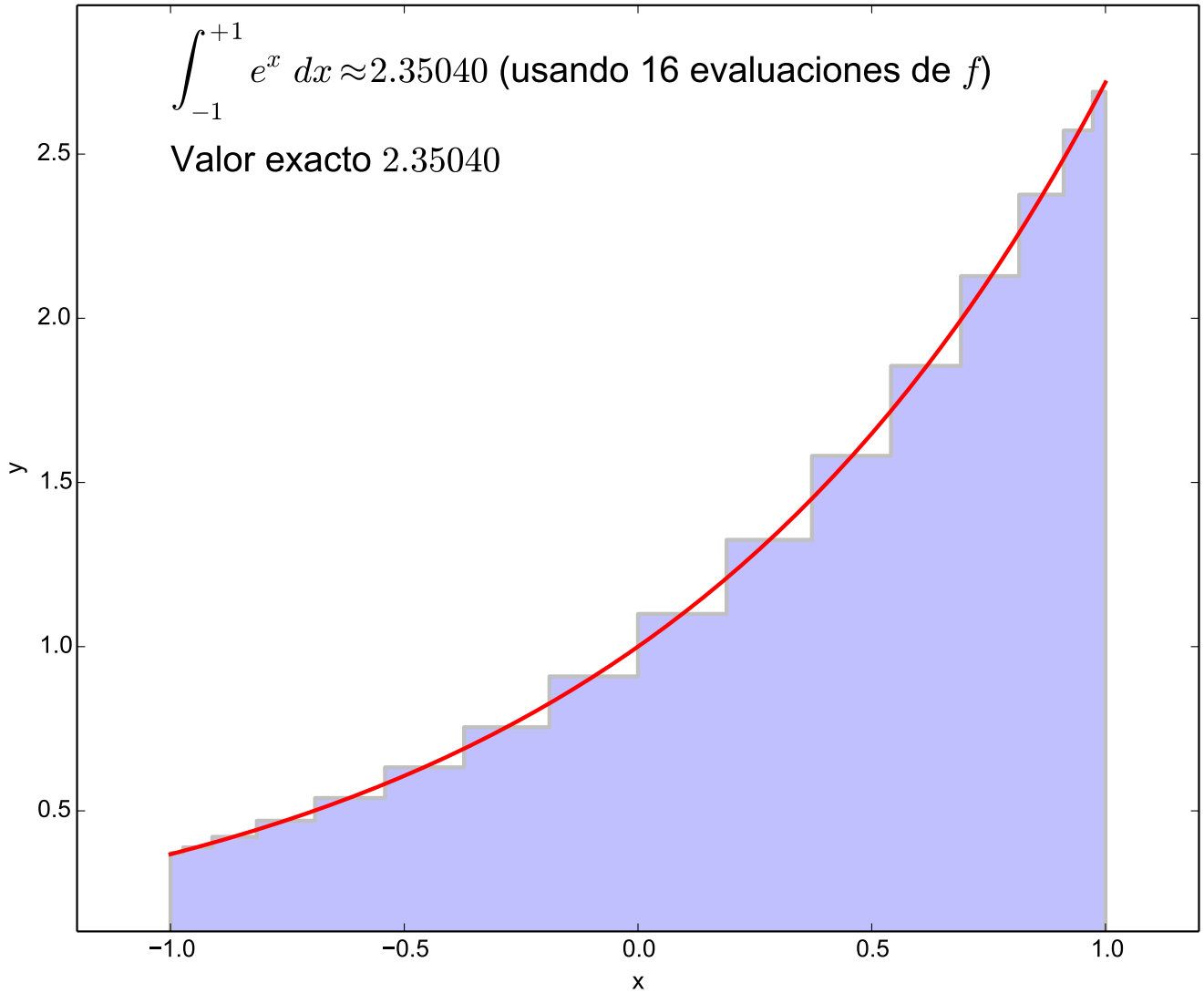
Cuadratura Gaussiana



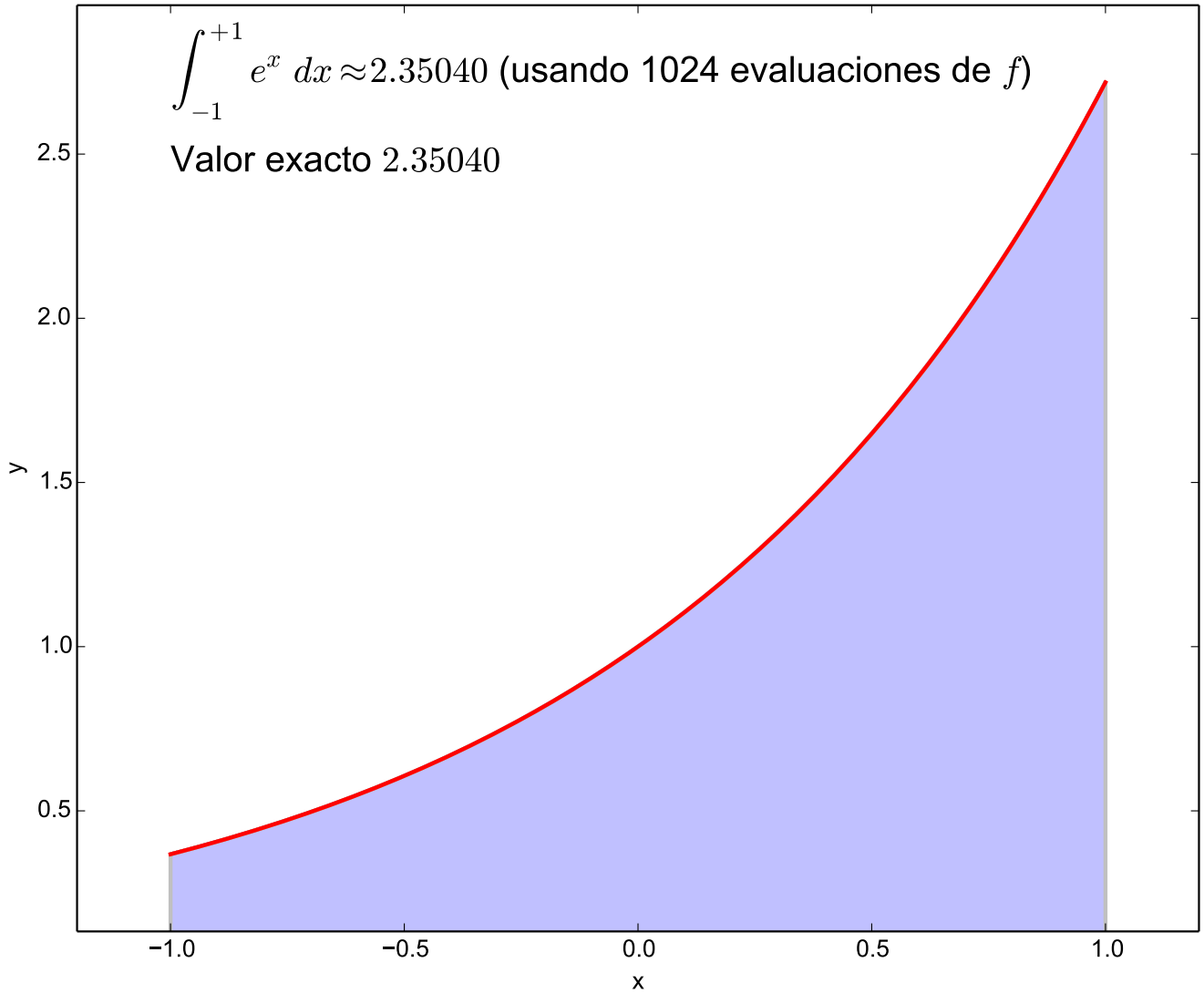
Cuadratura Gaussiana



Cuadratura Gaussiana



Cuadratura Gaussiana



Cuadratura Gaussiana

- La cuadratura Gaussiana puede resumirse como:

$$\int_{-1}^{1} f(x)\mathrm{d}x \approx \sum_{i=1}^{n} f(x_i)\, w_i$$

- ¿Qué ocurre si se quiere calcular una integral en el intervalo $[a, b]$?

$$\int_{a}^{b} f(x)\mathrm{d}x = \int_{-1}^{1} f\left(\frac{(b-a)t + b + a}{2}\right) \frac{b-a}{2}\mathrm{d}t$$

Cuadratura Gaussiana
Algoritmo

```
def gaussianquad(f, N, a, b):
    x, w = gaussian_nodes_and_weights(N, a, b)
    int_val = sum( w * f(x) )
    return int_val
def gaussian_nodes_and_weights(N, a, b):
    if N==1:
        return np.array([1]), np.array([2])
    beta = .5 / np.sqrt(1.-(2.*np.arange(1.,N))**(-2))
    T = np.diag(beta,1) + np.diag(beta,-1)
    D, V = np.linalg.eigh(T)
    x = D
    x = .5 * ( (b-a)*x + b + a) # Rescaling
    w = 2*V[0,:]**2
    w = .5*(b-a)*w
    return x, w
```

Pregunta:

¿Es posible escribir el algoritmo como un producto punto entre 2 vectores? Si fuera posible, ¿cuáles serían?

Análisis de Convergencia

- La convergencia de un método de integración depende del término de error asociado:

$$\text{Error} = C\, h^p\, f^{(q)}(c)$$

  con c una constante y $p, q \in \mathbb{N}$.
- La convergencia está relacionada con **el grado de precisión** y el **orden de convergencia** de un método.

Análisis de Convergencia

Grado de Precisión

Para un método de integración, corresponde al mayor entero k para el cual todos los polinomios de grado k o menor, son integrados exactamente (sin error).

- Las fórmulas de Newton-Cotes de grado n tienen grado de precisión n para n impar y $n+1$ para n par.
- Cuadratura Gaussiana tiene orden de precisión $2n+1$ cuando se usan $n+1$ puntos.
- ¿Qué parte del término del error está relacionada con el grado de precisión?

Análisis de Convergencia

Orden de convergencia

Para un valor de h pequeño ($h < 1$), corresponde a la mayor potencia p presente en el término del error. El orden de convergencia muestra como decrece el error a medida que se aumenta el número de puntos utilizados.

Otros Métodos

Otros métodos utilizados para integración numérica consideran:

- Métodos de **Newton-Cotes de orden superior**.
- **Método de Romberg:** de gran precisión, pero aplicable sólo cuando el intervalo está subdividido en $N = 2^n + 1$ puntos equiespaciados.
- **Métodos adaptativos:** subdividen el intervalo de manera no regular dependiendo de la función, de manera de sacar el máximo provecho a la información disponible.

Referencias

Numerical Analysis, Timothy Sauer, Second Edition, Pearson, 2012.
Chapter 5: Numerical Differentiation and Integration.