Integración Numérica

Métodos y Propiedades

(S)cientific (C)omputing (T)eam II-2014
ILI-286 DI-UTFSM Chile

25 de agosto de 2014

Contenido

Introducción

Definición

Integración Numérica o Cuadratura[a]:

Obtención del valor $c \in \mathbb{R}$ de una cierta integral definida $c = \int_a^b f(x)dx$ utilizando algún método numérico.

[a] Numerical Integration or Quadrature

Durante el video utilizaremos el siguiente ejemplo:

$$\int_{-1}^{+1} e^x dx = e^{+1} - e^{-1} = 2.35040...$$

Introducción

Existen 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... en estos casos resulta conveniente aplicar regla de Simpsons, Trapecio o Punto Medio, por nombrar algunas.
- La función f puede evaluarse en puntos elegidos arbitrariamente... en estos casos resulta conveniente aplicar la cuadratura gausiana.

Suma de Riemann

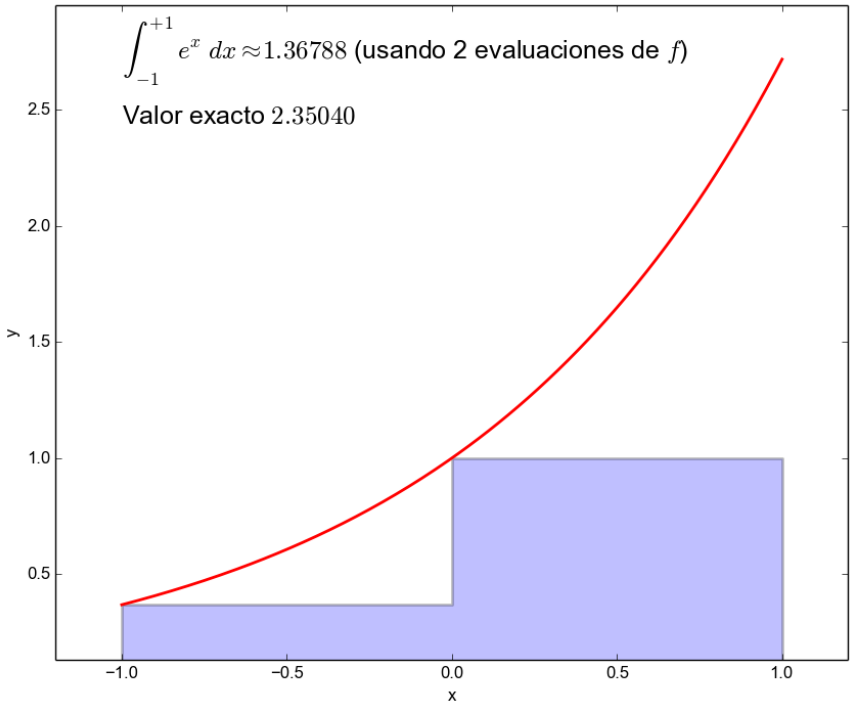
Recordemos que se dice que una función es integrable cuando las sumas izquierdas y derechas de Riemann convergen al mismo valor:

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

Suma Izquierda de Riemann

Suma Izquierda de Riemann

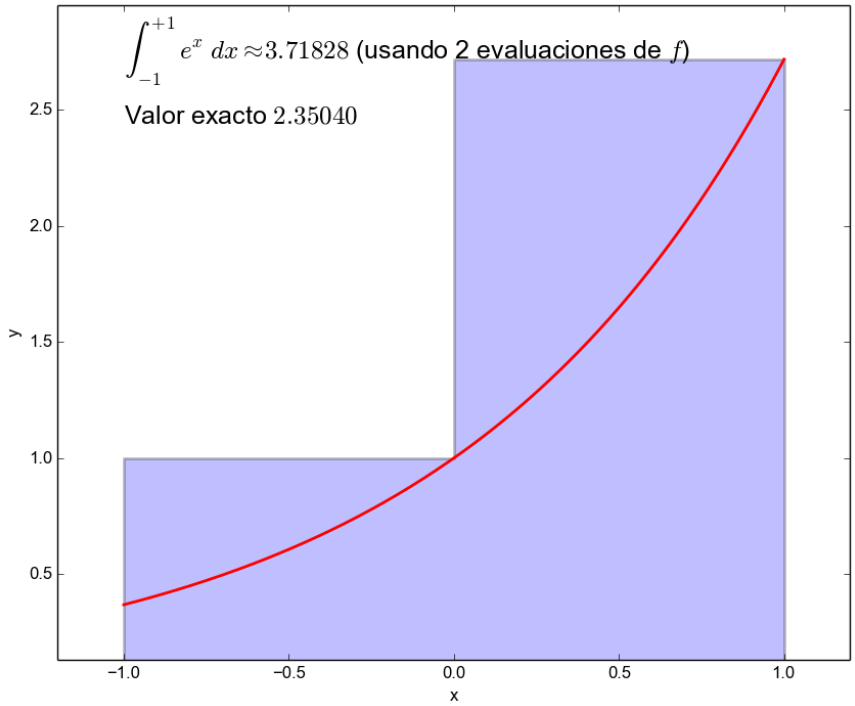
$$c = \int_a^b f(x)dx = \sup_P \sum_{i=0}^{N-1} f(x_i)(x_{i+1} - x_i)$$



Suma de Riemann

Suma Derecha de Riemann

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



Suma de Riemann

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:

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

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

Fórmulas de Newton-Cotes

Fórmulas de Newton-Cotes

Las fórmulas de Newton-Cotes se definen para ser exactas para polinomios de grado n.
La idea central es que se tienen los puntos $x_0, x_1, ..., x_n$ y $f(x_0), f(x_1), ..., f(x_n)$ y se supone que los puntos son interpolados por polinomios... puesto que la integral de polinomios puede ser estimada fácilmente.

- La regla del punto medio se obtiene con polinomios de grado 0 (constantes).
- La regla de trapecio se obtiene con polinomios de grado 1 (rectas).
- La regla de Simpsons se obtiene con polinomios de grado 2 (parábolas).

Regla del Punto Medio

Midpoint Rule

Regla del Punto Medio

Sabemos que para una constante se tiene

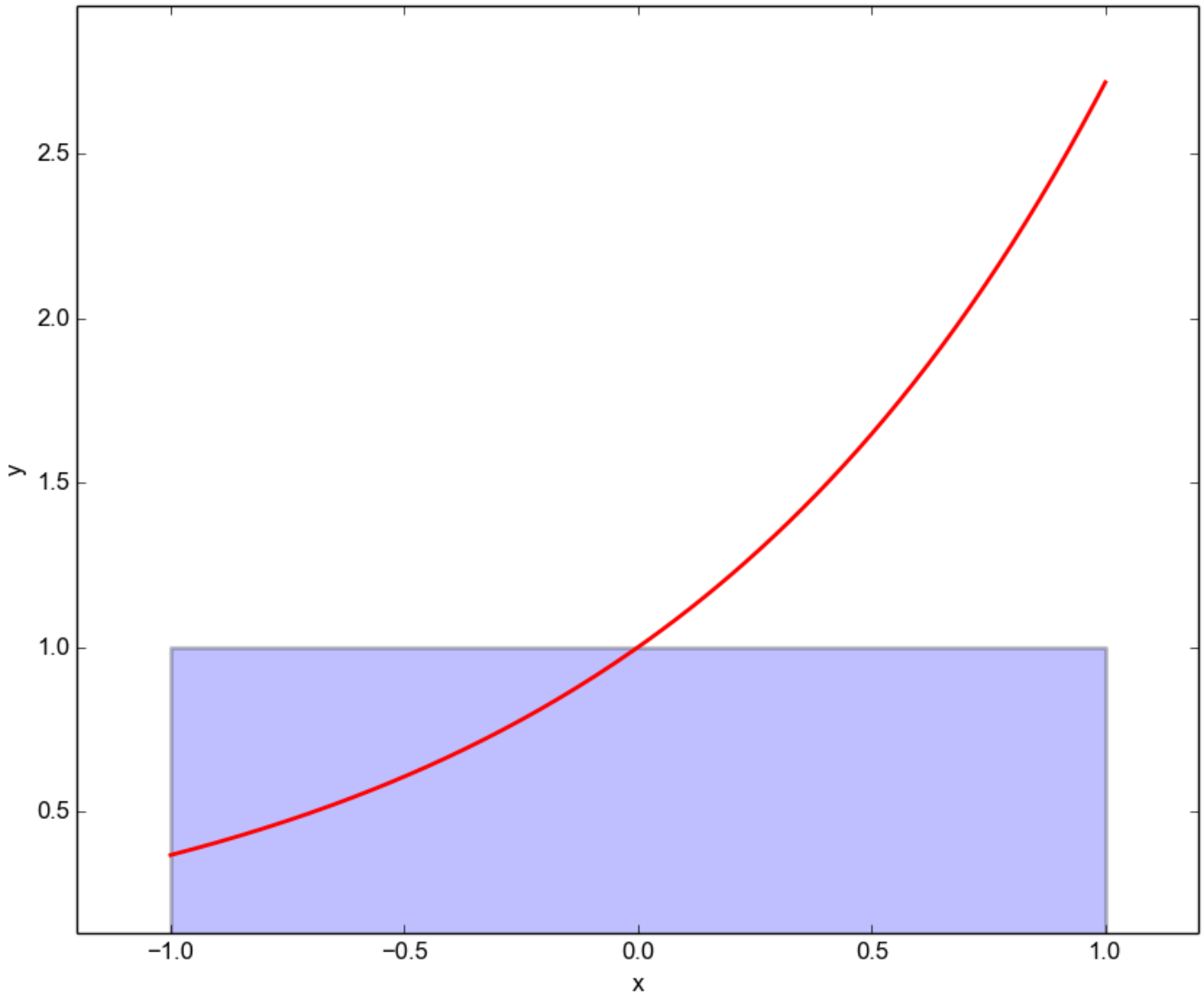
$$\int_{x_0}^{x_1} c\, dx = (x_1 - x_0)c$$

Si suponemos que $f(x)$ es constante e igual al valor correspondiente al punto medio del intervalo, tenemos:

$$\int_{x_0}^{x_1} f(x)dx \approx (x_1 - x_0)f(w),\ w = \frac{1}{2}(x_0 + x_1)$$

Regla del Punto Medio

Midpoint Rule



Regla del Punto Medio

Midpoint Rule

Regla del Punto Medio: Error Directo

La regla del punto medio tiene el siguiente error

$$\int_{x_0}^{x_1} f(x)dx = hf(w) + \frac{h^3}{24}f''(c)$$

donde

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

Regla del Punto Medio

Midpoint Rule

Regla del Punto Medio: Error Compuesto

Al aplicar a un intervalo $[a, b]$ subdividido en m segmentos y $m+1$ puntos, $x_0, \ldots, x_{m+1}$, tenemos

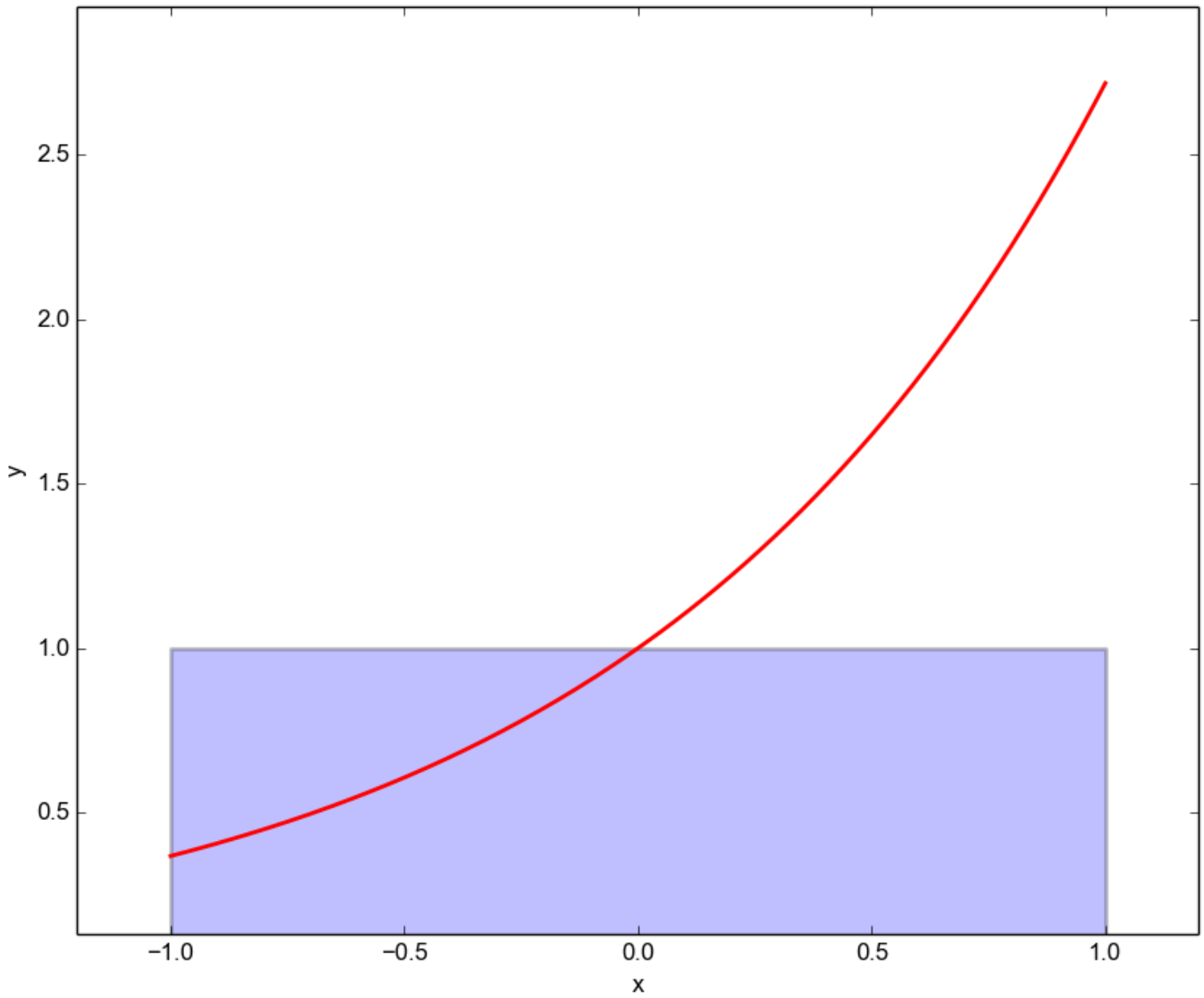
$$\int_a^b f(x)dx = \sum_{i=1}^{m} \int_{x_{i-1}}^{x_i} f(x)dx$$

$$= \sum_{i=1}^{m} hf(w_i) + \frac{(b-a)}{24} h^2 f''(c)$$

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

Regla del Punto Medio

Midpoint Rule



Regla del Punto Medio

Algoritmo

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
```

Regla del Trapecio

Trapezoid Rule

Regla del Trapecio

Sabemos que para una recta se tiene

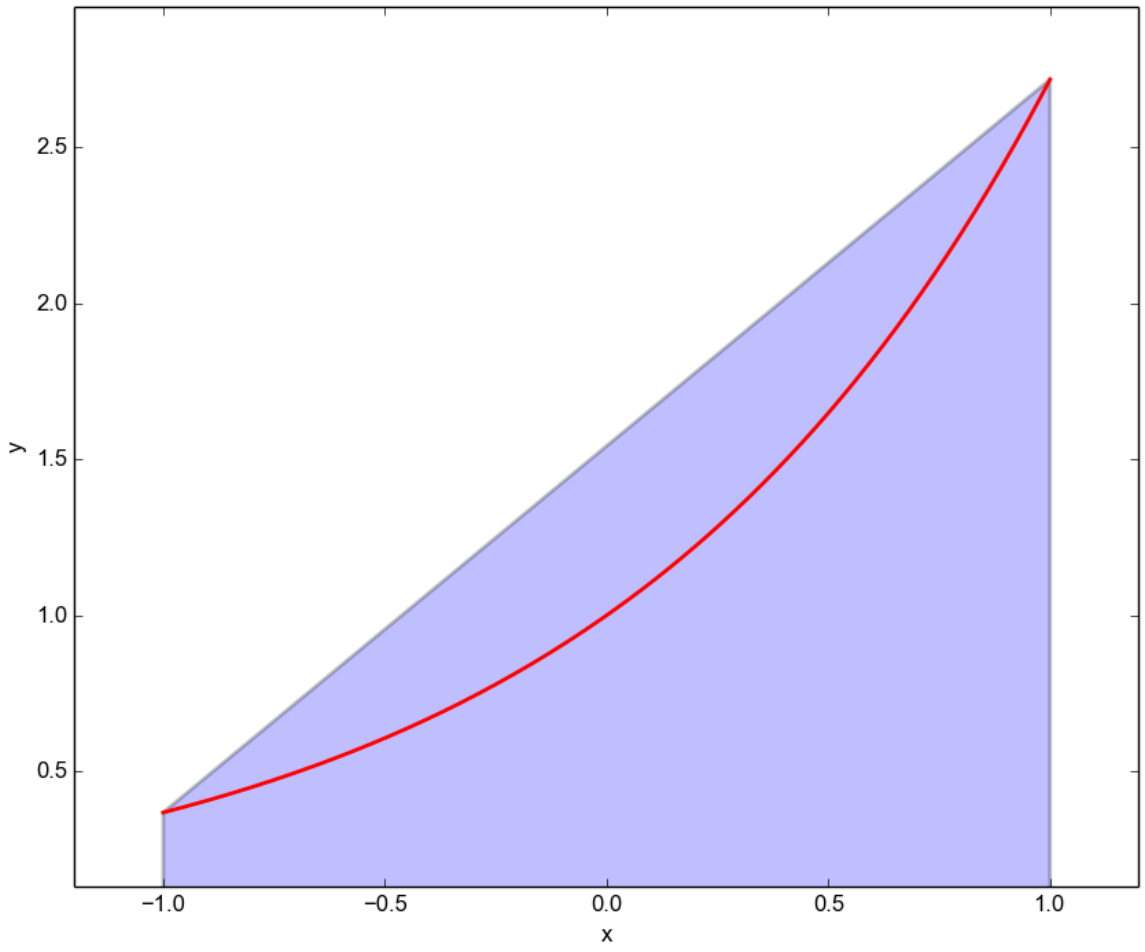
$$\int_{x_0}^{x_1} (mx + b)\, dx = (x_1 - x_0)\frac{(mx_0 + b) + (mx_1 + b)}{2}$$

Si suponemos que $f(x)$ es una recta que pasa por los valores en los extremos del intervalo, tenemos:

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

Regla del Trapecio

Trapezoid Rule



Regla del Trapecio

Trapezoid Rule

Regla del Trapecio: Error Directo

$$\int_{x_0}^{x_1} f(x)dx = \frac{h}{2}(f(x_0) + f(x_1)) + \frac{h^3}{12}f''(c)$$

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

Regla del Trapecio

Trapezoid Rule

Regla del Trapecio: Error Compuesto

Al aplicar a un intervalo $[a, b]$ subdividido en m segmentos y $m+1$ puntos, $x_0, \ldots, x_{m+1}$, tenemos

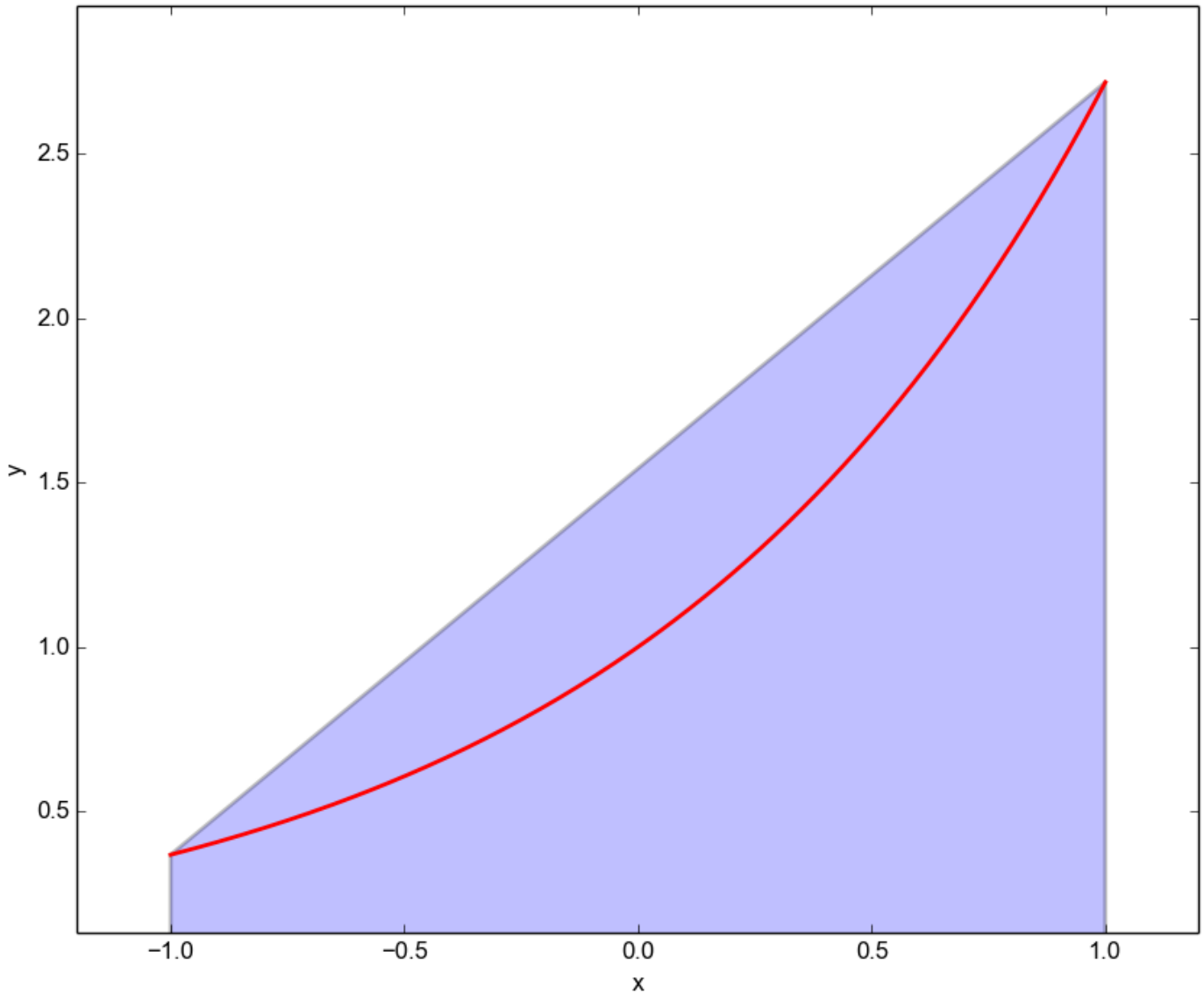
$$\int_{x_0}^{x_1} f(x)dx = \sum_{i=1}^{m} \int_{x_{i-1}}^{x_i} f(x)dx$$

$$= \frac{h}{2}\left[f(a) + f(b) + 2\sum_{i=1}^{m-1} f(x_i)\right] - (b-a)\frac{h^2}{12}f''(c)$$

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

Regla del Trapecio

Trapezoid Rule



Regla del Trapecio

Algoritmo

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
```

Regla de Simpson

Simpson's Rule

Regla de Simpson

Es posible probar que para una parábola se tiene

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

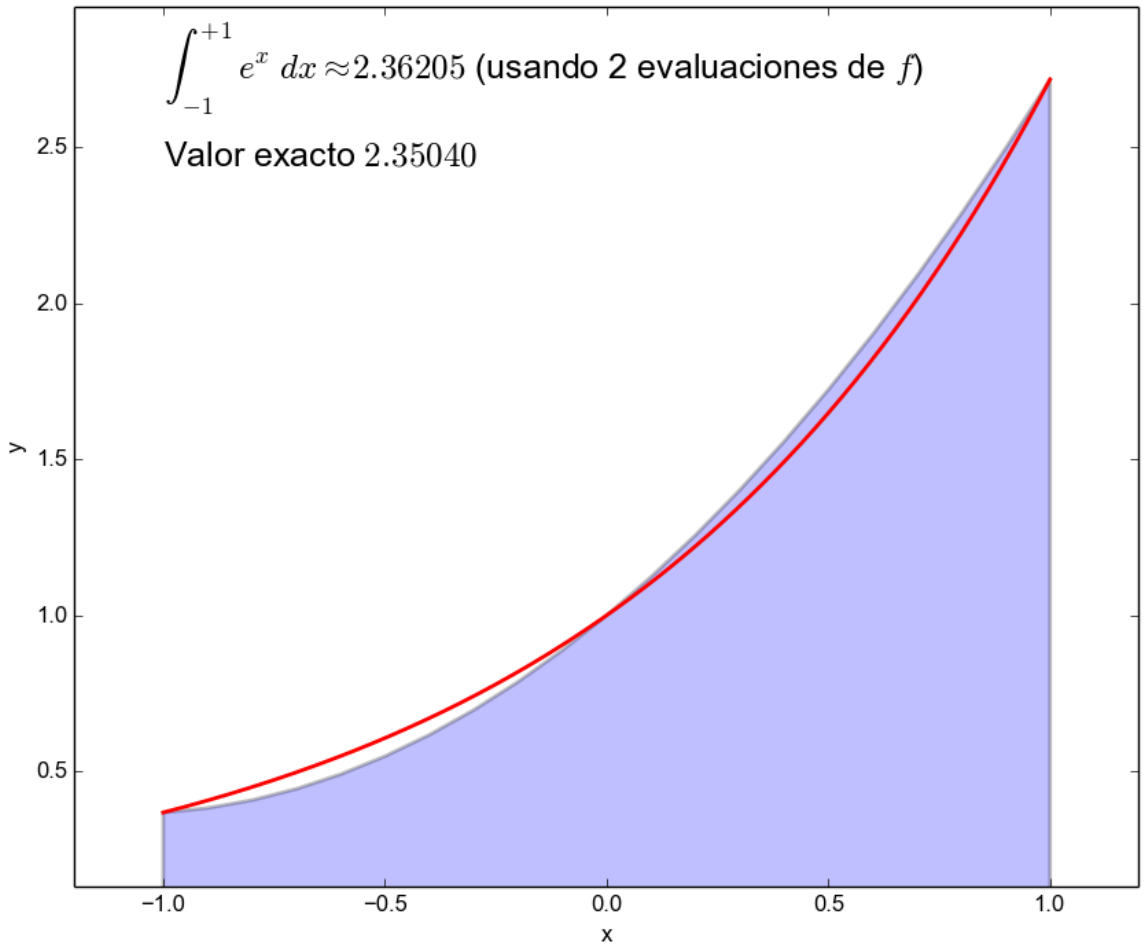
$$= (x_1 - x_0)\frac{1}{3}\left[(ax_0^2 + bx_0 + c) + 4(ax_1^2 + bx_1 + c) + (ax_2^2 + bx_2 + c\right.$$

donde $x_1 = (x_0 + x_2)/2$.
Si suponemos que $f(x)$ es una parábola que pasa por los valores en los extremos y punto medio del intervalo, tenemos:

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

Regla de Simpson

Simpson's Rule



Regla de Simpson

Simpson's Rule

Regla de Simpson: Error Directo

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

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

Regla de Simpson

Simpson's Rule

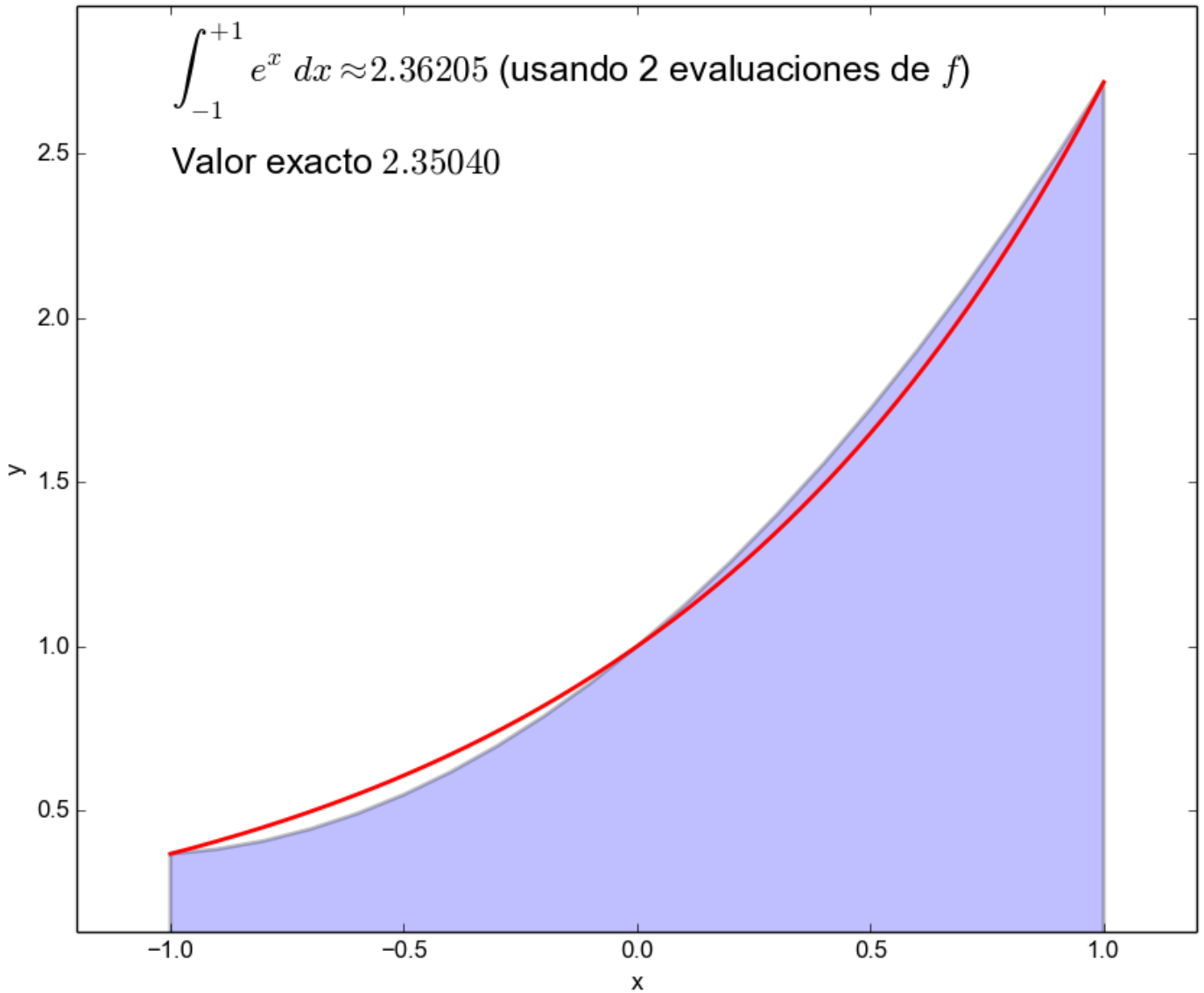
Regla de Simpson: Error Compuesto

$$\int_a^b f(x)dx = \frac{h}{3}(f(x_0) + \sum_{i=1}^{N-1} 6f(x_i) + f(x_N)) - (b-a)\frac{h^4}{90}f^{(4)}(c)$$

- $h = (x_{i+1} - x_i)$.
- $c \in [a, b]$

Regla de Simpson

Simpson's Rule



Regla del Simpsons

Algoritmo

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
```

Cuadratura Gaussiana

Gaussian Quadrature

Cuadratura Gaussiana

Si f podemos evaluarla en cualquier punto ¿cuál es la mejor forma de aproximar la integral?

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

- 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 buscamos evaluarla lo menos posible.

Cuadratura Gaussiana

Idea

Idea

Si consideramos que $f(x) \approx Q(x) = \sum_{i=1}^{n} L_i(x) f(x_i)$, la dependencia en x se encuentra sólamente en los términos $L_i(x)$. Al realizar la integración,

$$\int_{-1}^{1} f(x)dx \approx \int_{-1}^{1} Q(x)dx = \int_{-1}^{1} \sum_{i=1}^{n} L_i(x) f(x_i) dx$$

$$= \sum_{i=1}^{n} f(x_i) \underbrace{\int_{-1}^{1} L_i(x)dx}_{w_i}$$

Lo unico que queda es elegir apropiadamente los L_i

Idea

Tomamos

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

Que tiene las siguientes propiedades:

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

Cuadratura Gaussiana

Idea

Polinomio de Legendre

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

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

Que tiene las siguientes propiedades:

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

Cuadratura Gaussiana

Cuadro resumen

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

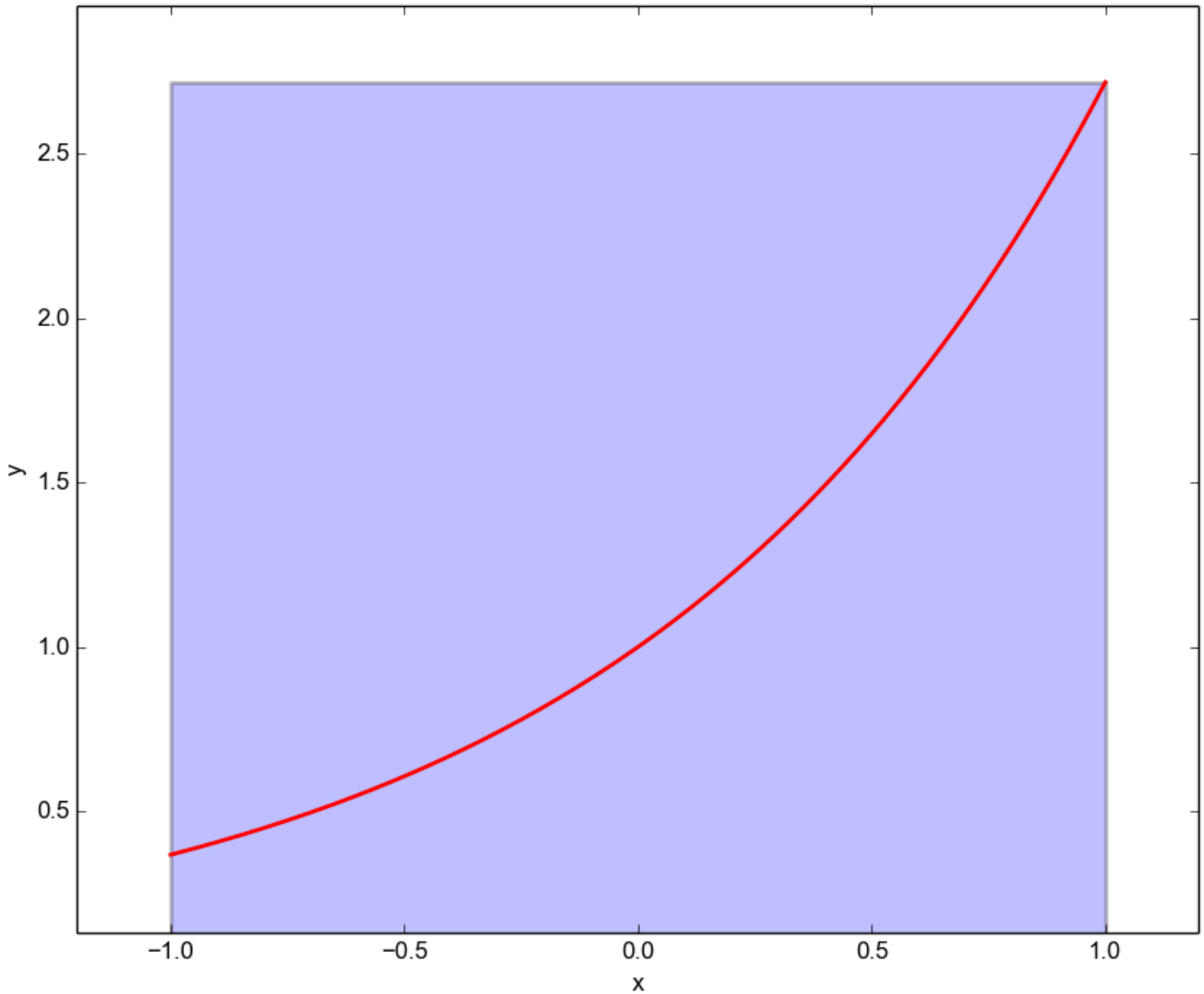
Cuadro resumen

Teorema

La cuadratura gaussiana usando polinomio de grado n de Legendre en $[-1, 1]$ tiene precisión $2n - 1$.

Cuadratura Gaussiana

Cuadro resumen



Cuadratura Gaussiana

Algoritmo

Algoritmo

```
import numpy as np

def gaussianquad(f, N, a, b):
  x, w = gaussian_nodes_and_weights(N)
  int_val = sum( w * f(x) )
  return int_val

def gaussian_nodes_and_weights(N):
  if N==1:
    return np.array([1]), np.array([2])
  beta = .5 / np.sqrt( 1. - (2.*np.arange(1.,N))*
  T = np.diag(beta,1) + np.diag(beta,-1)
  D, V = np.linalg.eigh(T)
  x = D
```

Otros Métodos

Existen otros métodos que no cubriremos por tiempo. Entre éstos se cuentan:

- 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.