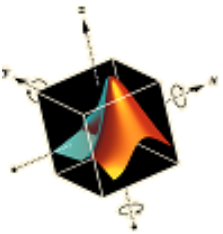
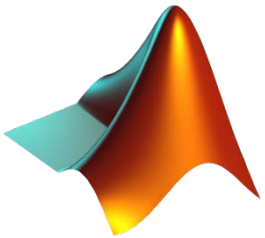
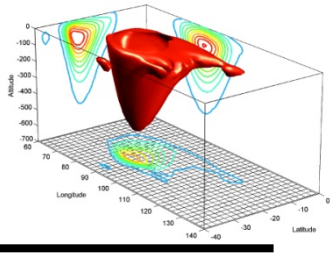




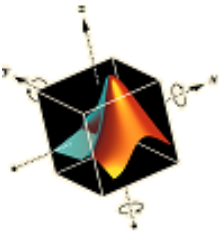
MÉTODOS NUMÉRICOS

Pedro Corcuera

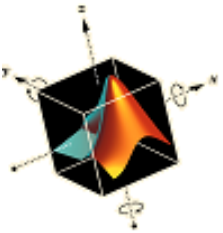
Dpto. Matemática Aplicada y
Ciencias de la Computación

Universidad de Cantabria

corcuerp@unican.es

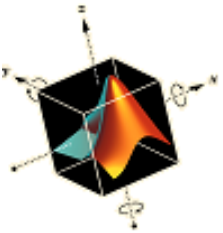


Indice

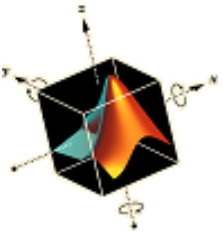


Objetivos

- Revisión de los principales métodos numéricos

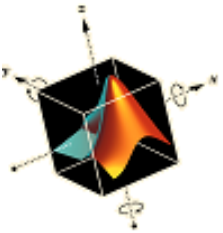


Introducción a la Computación Científica

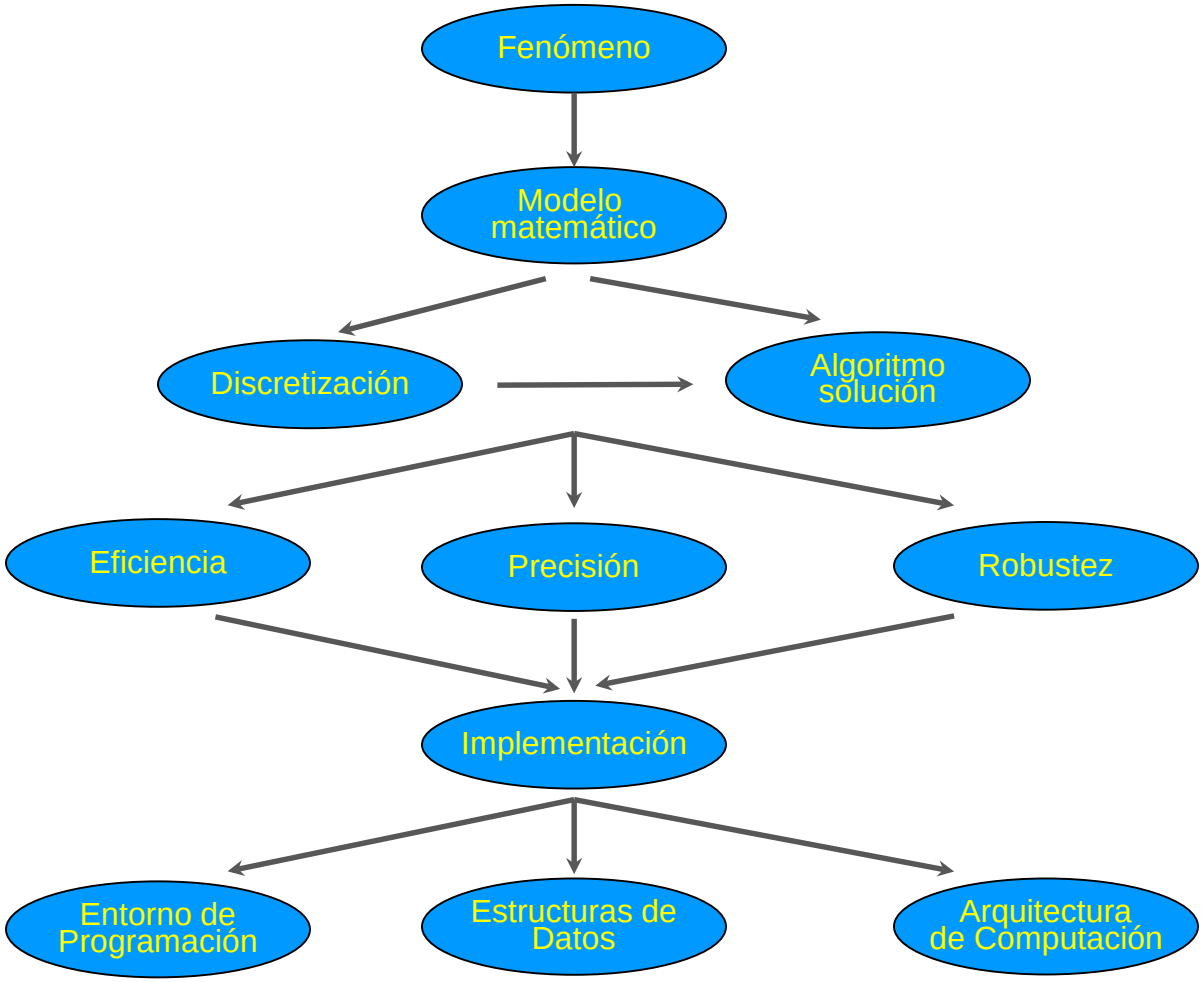


Computación científica

- Disciplina que desarrolla y estudia **algoritmos numéricos** para resolver problemas matemáticos que se presentan en la ciencia e ingeniería.
- El punto de partida es un **modelo matemático**.
- Para resolver el modelo de manera aproximada en un ordenador se discretiza y aplican algoritmos que lo resuelven de forma eficiente, precisa y fiable.
- El **análisis numérico** sustenta la teoría que hay detrás de los algoritmos.



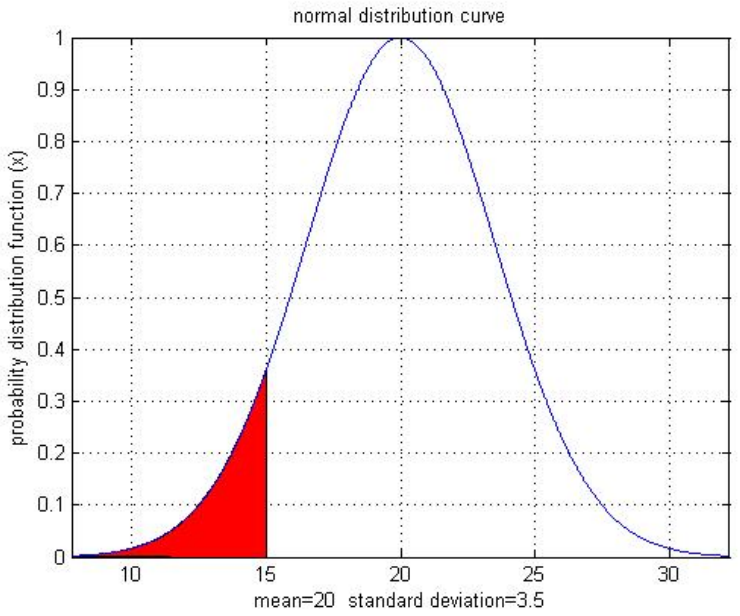
Computación científica

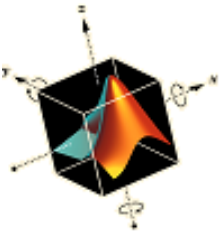


¿Por qué usar métodos numéricos?

- Para resolver problemas que no se pueden resolver de forma exacta o analítica
- Para resolver problemas que son intratables

$$\frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{u^2}{2}} du$$





¿Cómo resolver un problema de ingeniería?

Descripción del Problema

↓

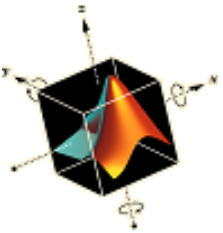
Modelo matemático

↓

Solución del modelo matemático

↓

Utilización de la solución



¿Cómo resolver un problema de ingeniería?

Descripción del problema

Definición	Fácil o difícil?

Diseño del modelo matemático

Búsqueda de solución	Control del modelo

Solución del modelo matemático

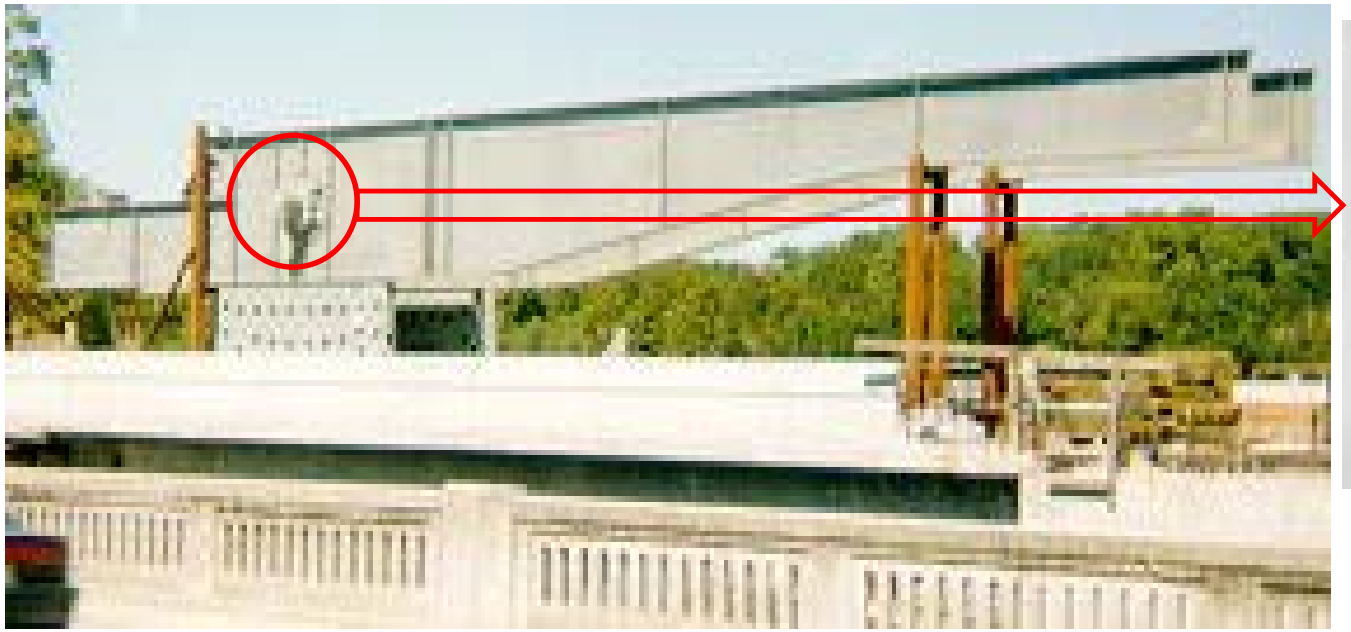
Con programa de ordenador?	Qué pasos?

Utilizacion del modelo

Comprobar resultados	Volver a hacerlo, si está mal!

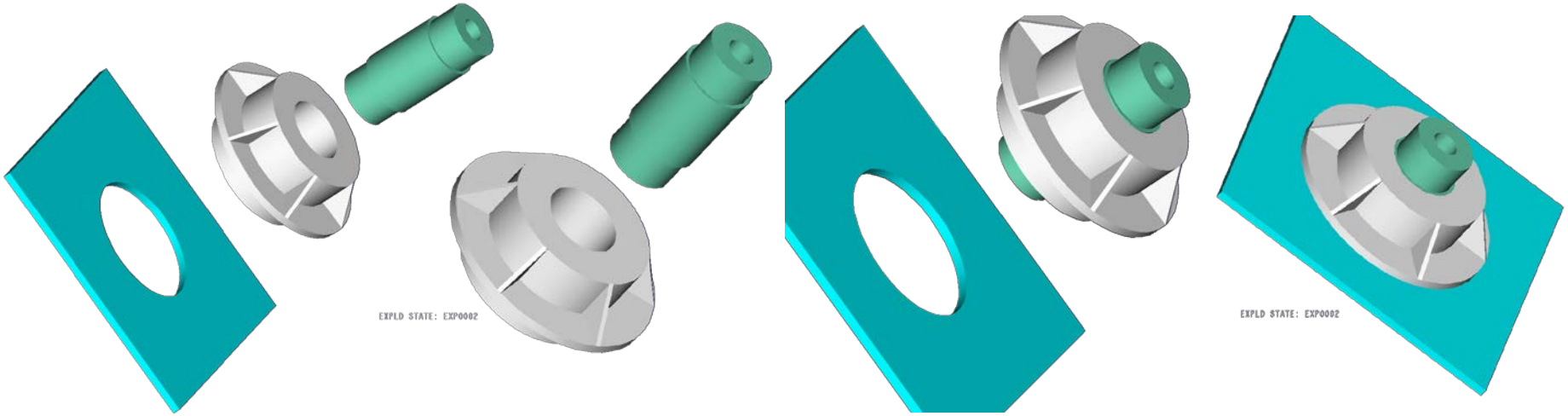
Ejemplos de soluciones de problemas de ingeniería

- Colocar un pivote en una viga



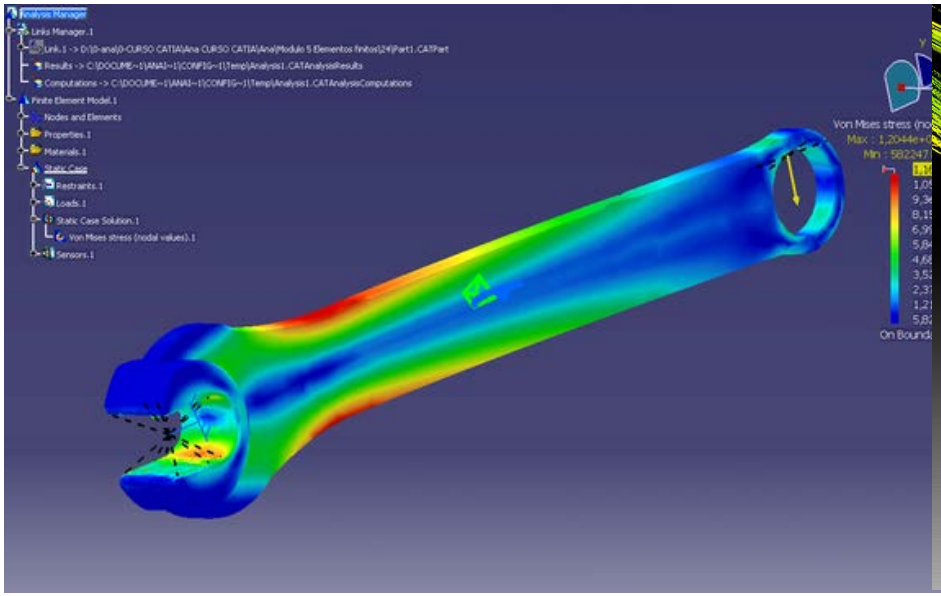
Ejemplos de soluciones de problemas de ingeniería

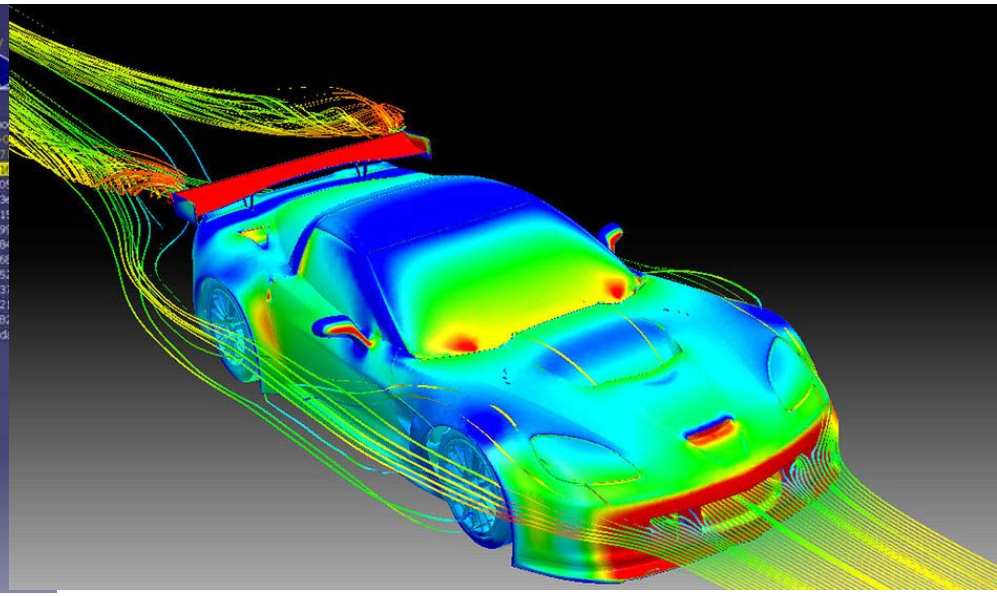
- Pasos:
  - Eje sumergido en nitrógeno líquido
  - Eje dejado a temperatura ambiente en el apoyo
  - Eje-apoyo enfriado
  - Eje-apoyo dejado a temperatura ambiente en la viga

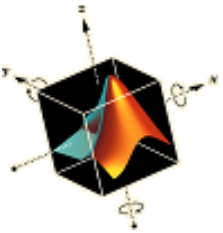


Ejemplos de aplicaciones en ingeniería

- Todas las ramas de la ingeniería
  - Journal of Computational and Applied Mathematics
  - International Journal of Numerical Methods and Applications
  - International Journal for Numerical Methods in Engineering



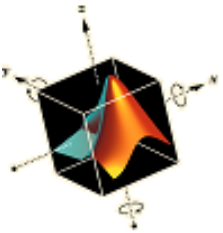




Desastres causados por errores numéricos

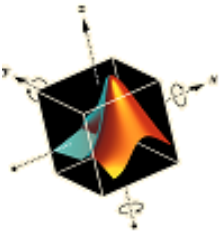
- Patriot Missile Failure
- Explosion of the Ariane 5
- EURO page: Conversion Arithmetics
- The Vancouver Stock Exchange
- Rounding error changes Parliament makeup
- The sinking of the Sleipner A offshore platform
- Tacoma bridge failure (wrong design)
- 200 million dollar typing error (typing error)
- What's 77.1 x 850? Don't ask Excel 2007
- Collection of Software Bugs

http://ta.twi.tudelft.nl/users/vuik/wi211/disasters.html



Temas

- Nonlinear Equations
- Differentiation
- Simultaneous Linear Equations
- Interpolation
- Regression
- Integration
- Ordinary Differential Equations
- Partial Differential Equations
- Optimization
- Fourier and LaplaceTransforms
- Graphs

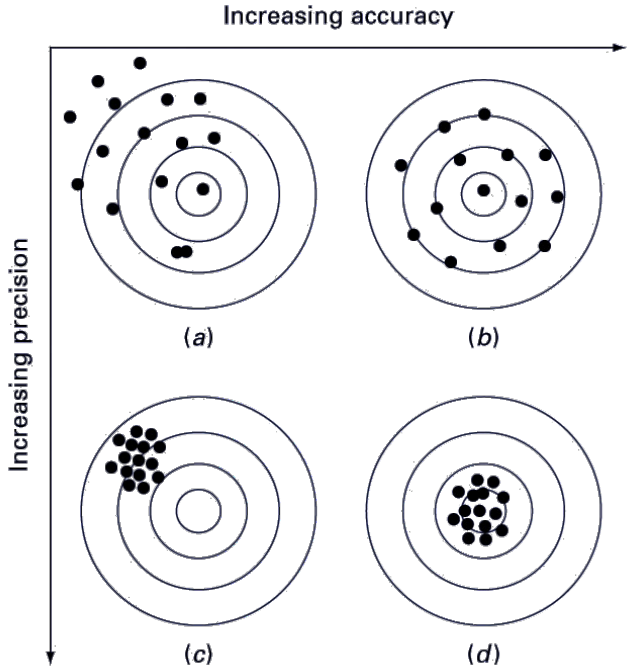


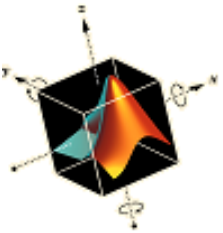
Errores

Exactitud y precisión

- La *exactitud* (accuracy) se refiere a la proximidad de un valor calculado o medido respecto del valor verdadero.
- La *precisión* (precision) se refiere a la proximidad de un valor individual calculado o medido respecto de otros.

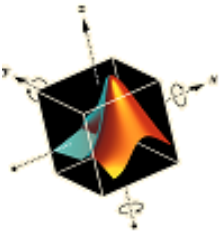
a) inaccurate and imprecise
b) accurate and imprecise
c) inaccurate and precise
d) accurate and precise





¿Por qué medir errores?

- To determine the accuracy of numerical results
- To develop stopping criteria for iterative algorithms

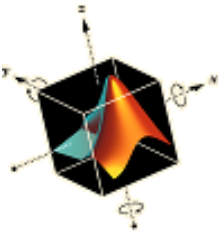


Definiciones de error

- **True error** (E_t): the difference between the true value in a calculation and the approximate value found using a numerical method etc.

  True Error (E_t)= True Value – Approximate Value

- **Absolute error** ($|E_t|$): the absolute difference between the true value and the approximation.

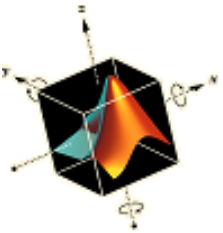


True error - ejemplo

- The derivative $f'(x)$ of a function $f(x)$ can be approximated by the equation

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

- If $f(x) = 7e^{0.5x}$ and $h = 0.3$
  a) Find the approximate value of $f'(2)$
  b) True value of
  c) True error for part (a)



True error - ejemplo

- Solution:

a) For $x = 2$ and $h = 0.3$

$$f'(2) \approx \frac{f(2+0.3) - f(2)}{0.3} = \frac{f(2.3) - f(2)}{0.3} = \frac{7e^{0.5(2.3)} - 7e^{0.5(2)}}{0.3} = \frac{22.107 - 19.028}{0.3} = 10.263$$

b) The exact value of $f'(2)$ can be found by using differential calculus.

$$f(x) = 7e^{0.5x}$$

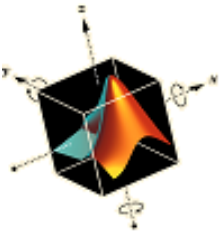
$$f'(x) = 7 \times 0.5 \times e^{0.5x} = 3.5e^{0.5x}$$

$$f'(2) = 3.5e^{0.5(2)} = 9.5140$$

c) True error is calculated as

$$E_t = \text{True Value} - \text{Approximate Value}$$

$$= 9.5140 - 10.263 = -0.722$$

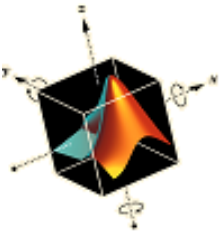


Definiciones de error

- **Relative true error**: the true error divided by the true value.

$$\text{Relative True Error } (\in_t) = \frac{\text{True Error}}{\text{True Value}}$$

- **Relative error** (ε_t): the relative true error expressed as a percentage.



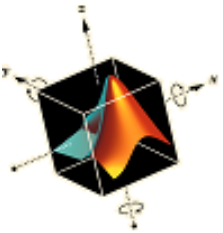
Relative true error - ejemplo

- Following from the previous example for true error, find the relative true error for $f(x) = 7e^{0.5x}$ at $f'(2)$ with $h = 0.3$
- From the previous example, $E_t = -0.722$
- Relative true error is defined as

$$\in_t = \frac{\text{True Error}}{\text{True Value}} = \frac{-0.722}{9.5140} = -0.075888$$

as a percentage

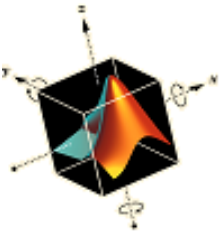
$$\varepsilon_t = -0.075888 \times 100\% = -7.5888\%$$



Error aproximado

- The previous definitions of error relied on knowing a true value. What can be done if true values are not known or are very difficult to obtain?
- If this is the case, approximations can be made to the error. **Approximate error** is defined as the difference between the present approximation and the previous approximation.

Approximate Error (E_a)= Present appr. – Previous appr.



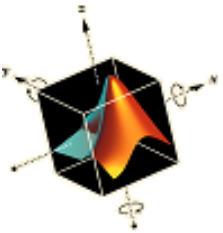
Error aproximado - ejemplo

- For $f(x) = 7e^{0.5x}$ at $x = 2$ find the following,
  a) $f'(2)$ using $h = 0.3$
  b) $f'(2)$ using $h = 0.15$
  c) Approximate error for the value of $f'(2)$ for part (b)

a) For $x = 2$ and $h = 0.3$

$$f'(x) \approx \frac{f(x+h) - f(x)}{h} \approx \frac{f(2+0.3) - f(2)}{0.3} = \frac{f(2.3) - f(2)}{0.3}$$

$$= \frac{7e^{0.5(2.3)} - 7e^{0.5(2)}}{0.3} = \frac{22.107 - 19.028}{0.3} = 10.263$$



Error aproximado - ejemplo

b) For $x = 2$ and $h = 0.15$

$$f'(x) \approx \frac{f(x+h) - f(x)}{h} \approx \frac{f(2+0.15) - f(2)}{0.15} = \frac{f(2.15) - f(2)}{0.15}$$

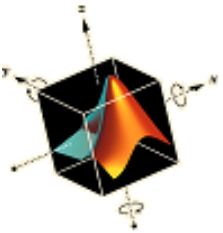
$$= \frac{7e^{0.5(2.15)} - 7e^{0.5(2)}}{0.15} = \frac{20.50 - 19.028}{0.15} = 9.8800$$

c) The approximate error E_a is

E_a = Present Approximation – Previous Approximation

$$= 9.8800 - 10.263$$

$$= -0.38300$$

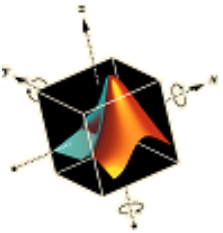


Error aproximado relativo

- The **relative approximate error** is defined as the approximate error divided by the present approximation, normally expressed as a percentage.

$$\text{Relative Approximate Error } (\in_a) = \frac{\text{Approximate Error}}{\text{Present approximation}}$$

- For iterative processes, the error can be approximated as the difference in values between sucessive iterations.



Error aproximado relativo - ejemplo

- For $f(x) = 7e^{0.5x}$ at $x = 2$ find the relative approximate error using values from $h = 0.3$ and $h = 0.15$

  Solution:

  From previous example, the approximate value of $f'(2) = 10.263$ using $h = 0.3$ and $f'(2) = 9.8800$ using $h = 0.15$

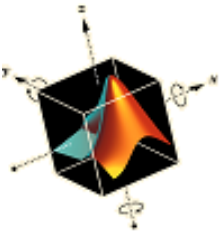
  E_a = Present Approximation – Previous Approximation

  $$= 9.8800 - 10.263 = -0.38300$$

  $$\in_a = \frac{\text{Approximate Error}}{\text{Present Approximation}} = \frac{-0.38300}{9.8800} = -0.038765$$

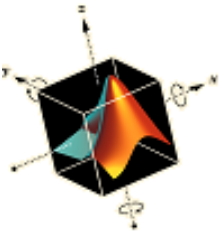
  as a percentage $\in_a = -0.038765 \times 100\% = -3.8765\%$

absolute relative approximation: $|\in_a| = 0.038765 \text{ or } 3.8765\%$



Uso del error relativo absoluto como criterio de parada

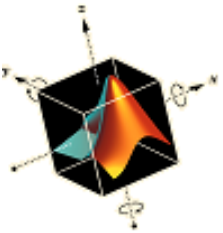
- Often, when performing calculations, we are interested in whether the absolute value of the percent relative error is lower than a prespecified tolerance ε_s. For such cases, the computation is repeated until $|\varepsilon_a| < \varepsilon_s$
- This relationship is referred to as a *stopping criterion*.
- If at least m significant digits are required to be correct in the final answer, then $|\in_a| \leq 0.5 \times 10^{2-m}\ \%$



Uso del error relativo absoluto como criterio de parada – tabla de ejemplo

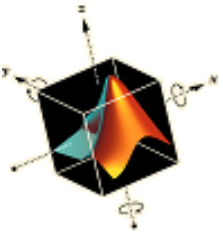
- For $f(x) = 7e^{0.5x}$ at $x = 2$ with varying step size h

h	$f'(2)$	$\lvert\epsilon_a\rvert$	m
0.3	10.263	N/A	0
0.15	9.8800	3.877%	1
0.10	9.7558	1.273%	1
0.01	9.5378	2.285%	1
0.001	9.5164	0.2249%	2



Fuentes de errores numéricos

- Round off error
- Truncation error

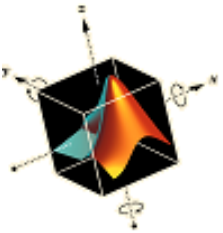


Errores de redondeo

- *Round off errors* arise because digital computers cannot represent some quantities exactly. Caused by representing a number approximately.

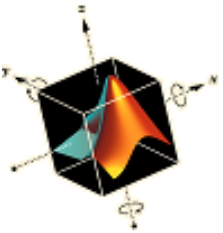
$$\frac{1}{3} \cong 0.333333 \qquad \sqrt{2} \cong 1.4142...$$

- There are two major facets of round off errors involved in numerical calculations:
  - Digital computers have size and precision limits on their ability to represent numbers.
  - Certain numerical manipulations are highly sensitive to round off errors.



Errores de redondeo con manipulaciones aritméticas

- Round off error can happen in several circumstances other than just storing numbers - for example:
  - *Large computations* - if a process performs a large number of computations, round off errors may build up to become significant
  - *Adding a Large and a Small Number* - Since the small number's mantissa is shifted to the right to be the same scale as the large number, digits are lost
  - *Smearing* - Smearing occurs whenever the individual terms in a summation are larger than the summation itself.
    - $(x + 10^{-20}) - x = 10^{-20}$ mathematically, but x = 1; $(x + 10^{-20}) - x$ gives a 0 in MATLAB!



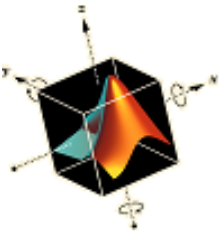
Errores de truncamiento

- *Truncation errors* are caused by truncating or approximating a mathematical procedure.
- Ejemplo: Taking only a few terms of a Maclaurin series to approximate e^x

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots\ldots\ldots\ldots\ldots$$

If only 3 terms are used:

$$Truncation\ \ Error = e^x - \left(1 + x + \frac{x^2}{2!}\right)$$



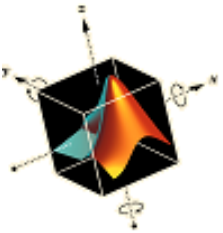
Errores de truncamiento - ejemplo

- Calculate the value of $e^{1.2}$ with an absolute relative approximate error of less than 1%.

$$e^{1.2} = 1 + 1.2 + \frac{1.2^2}{2!} + \frac{1.2^3}{3!} + \ldots\ldots\ldots\ldots\ldots$$

n	$e^{1.2}$	E_a	$\lvert\epsilon_a\rvert\%$
1	1	—	—
2	2.2	1.2	54.545
3	2.92	0.72	24.658
4	3.208	0.288	8.9776
5	3.2944	0.0864	2.6226
6	3.3151	0.020736	0.62550

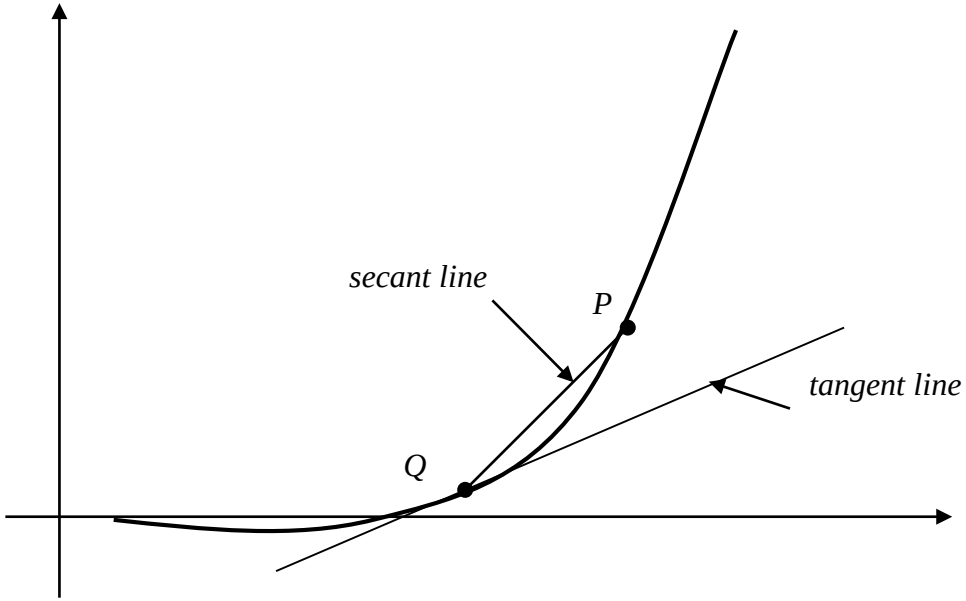
6 terms are required



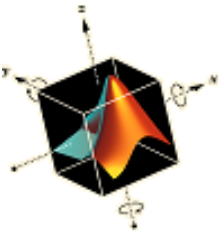
Errores de truncamiento – otro ejemplo

- Using a finite Δx to approximate $f'(x)$

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



Approximate derivative using finite Δx



Errores de truncamiento - ejemplo

- Find $f'(3)$ for $f(x) = x^2$ using $f'(x) \approx \dfrac{f(x+\Delta x) - f(x)}{\Delta x}$ and $\Delta x = 0.2$

$$f'(3) = \frac{f(3+0.2) - f(3)}{0.2} = \frac{f(3.2) - f(3)}{0.2} = \frac{3.2^2 - 3^2}{0.2} = \frac{10.24 - 9}{0.2} = \frac{1.24}{0.2} = 6.2$$

The actual value is

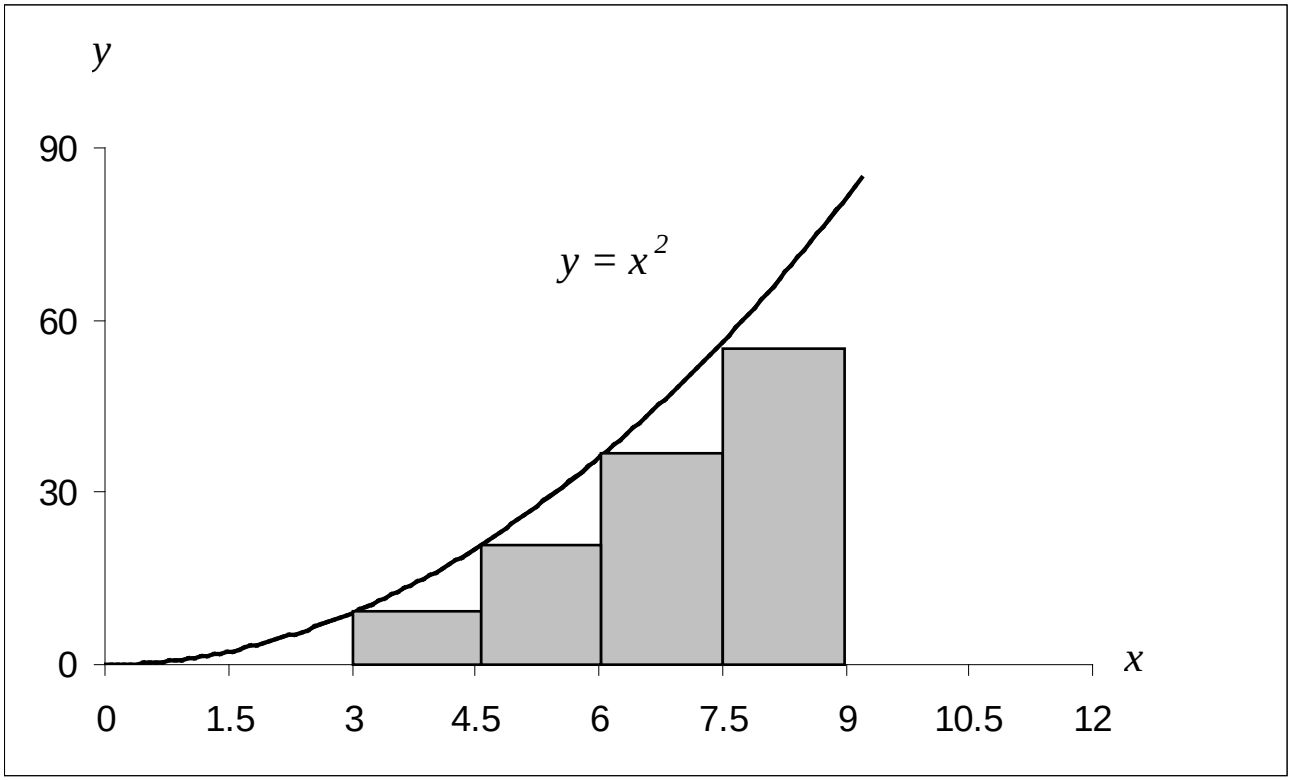
$$f'(x) = 2x, \quad f'(3) = 2 \times 3 = 6$$

Truncation error is then: $6 - 6.2 = -0.2$

Find the truncation error with $\Delta x = 0.1$

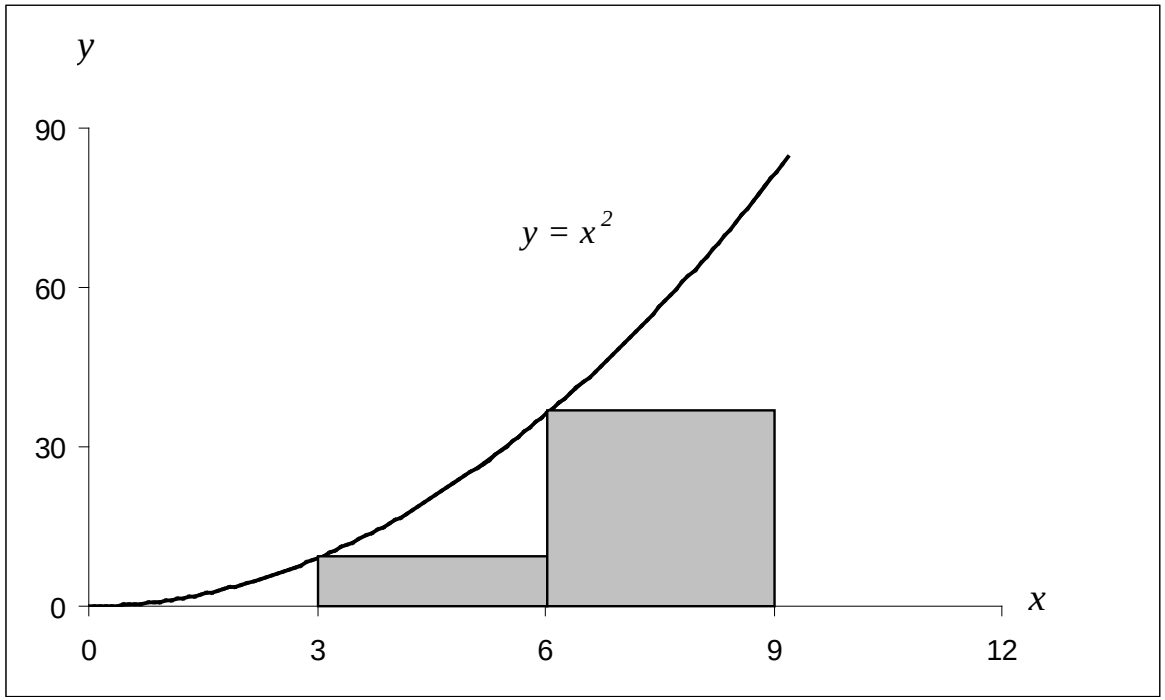
Errores de truncamiento – otro ejemplo

- Using finite rectangles to approximate an integral.

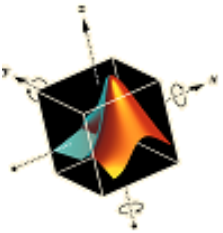


Errores de truncamiento - ejemplo

- Use two rectangles of equal width to approximate the area under the curve for $f(x) = x^2$ over the interval $[3,9]$



$$\int_3^9 x^2 dx$$



Errores de truncamiento - ejemplo

- Choosing a width of 3, we have

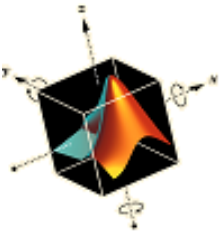
$$\int_3^9 x^2 dx = (x^2)\Big|_{x=3}(6-3) + (x^2)\Big|_{x=6}(9-6) = (3^2)3 + (6^2)3 = 27 + 108 = 135$$

Actual value is given by

$$\int_3^9 x^2 dx = \left[\frac{x^3}{3}\right]_3^9 = \left[\frac{9^3 - 3^3}{3}\right] = 234$$

Truncation error is then: $234 - 135 = 99$

Find the truncation error with 4 rectangles



Representación de números

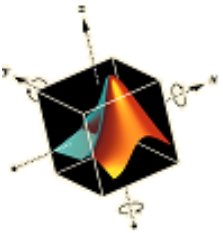
- Representación decimal

$$257.76 = 2\times10^{2} + 5\times10^{1} + 7\times10^{0} + 7\times10^{-1} + 6\times10^{-2}$$

- Base 2 (binaria)

$$(1011.0011)_2 = \begin{pmatrix} (1\times2^{3} + 0\times2^{2} + 1\times2^{1} + 1\times2^{0}) \\ + (0\times2^{-1} + 0\times2^{-2} + 1\times2^{-3} + 1\times2^{-4}) \end{pmatrix}_{10}$$

$$= 11.1875$$



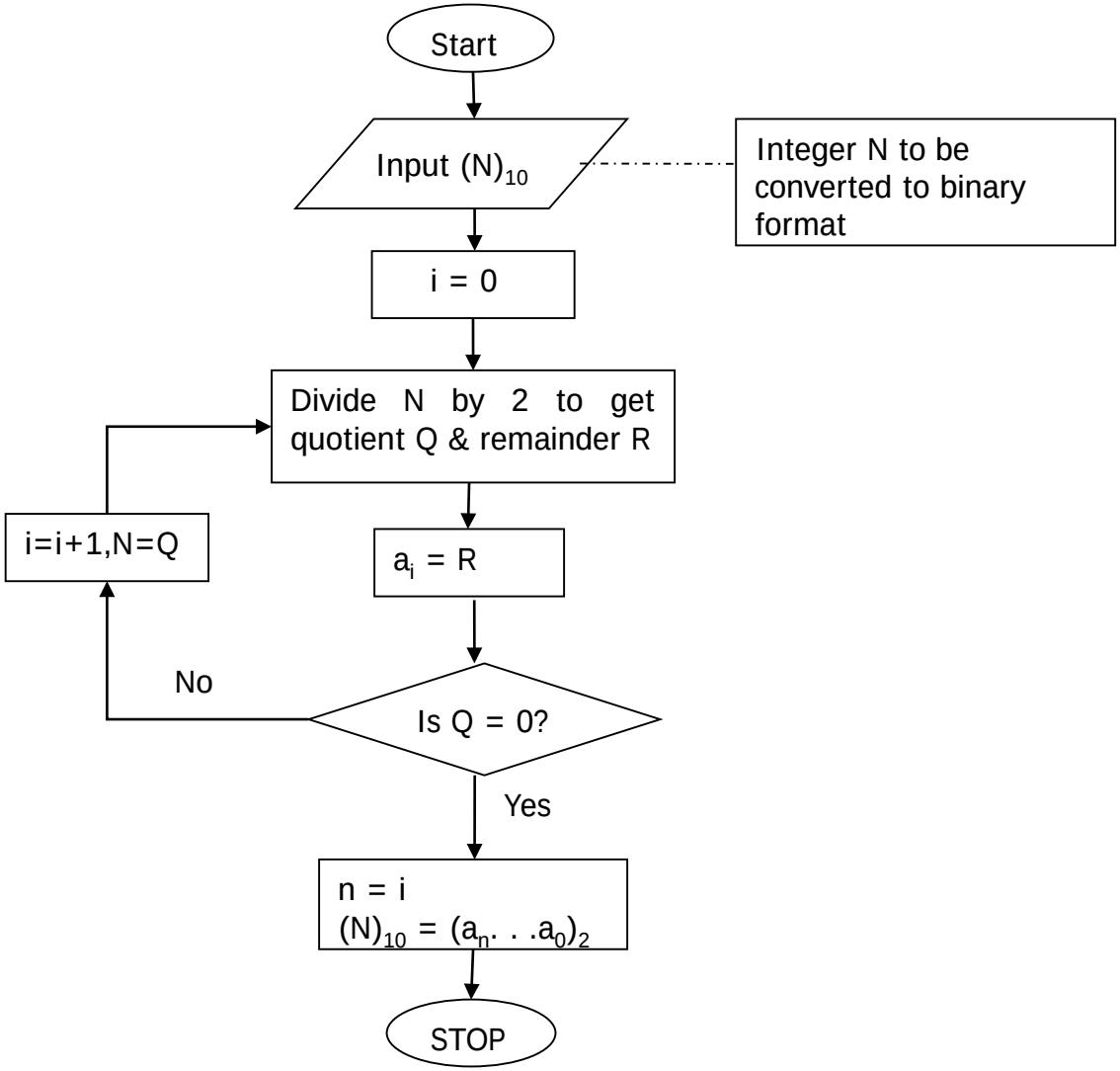
Conversión de entero base 10 a binario

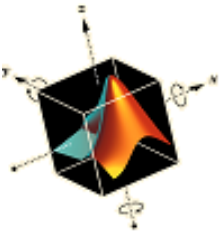
Operación	Quotient	Remainder
11/2	5	$1 = a_0$
5/2	2	$1 = a_1$
2/2	1	$0 = a_2$
1/2	0	$1 = a_3$

Hence

$$(11)_{10} = (a_3 a_2 a_1 a_0)_2$$
$$= (1011)_2$$

Conversión de entero base 10 a binario flujograma



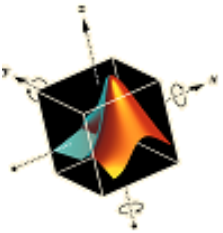


Conversión de fracción decimal a binario

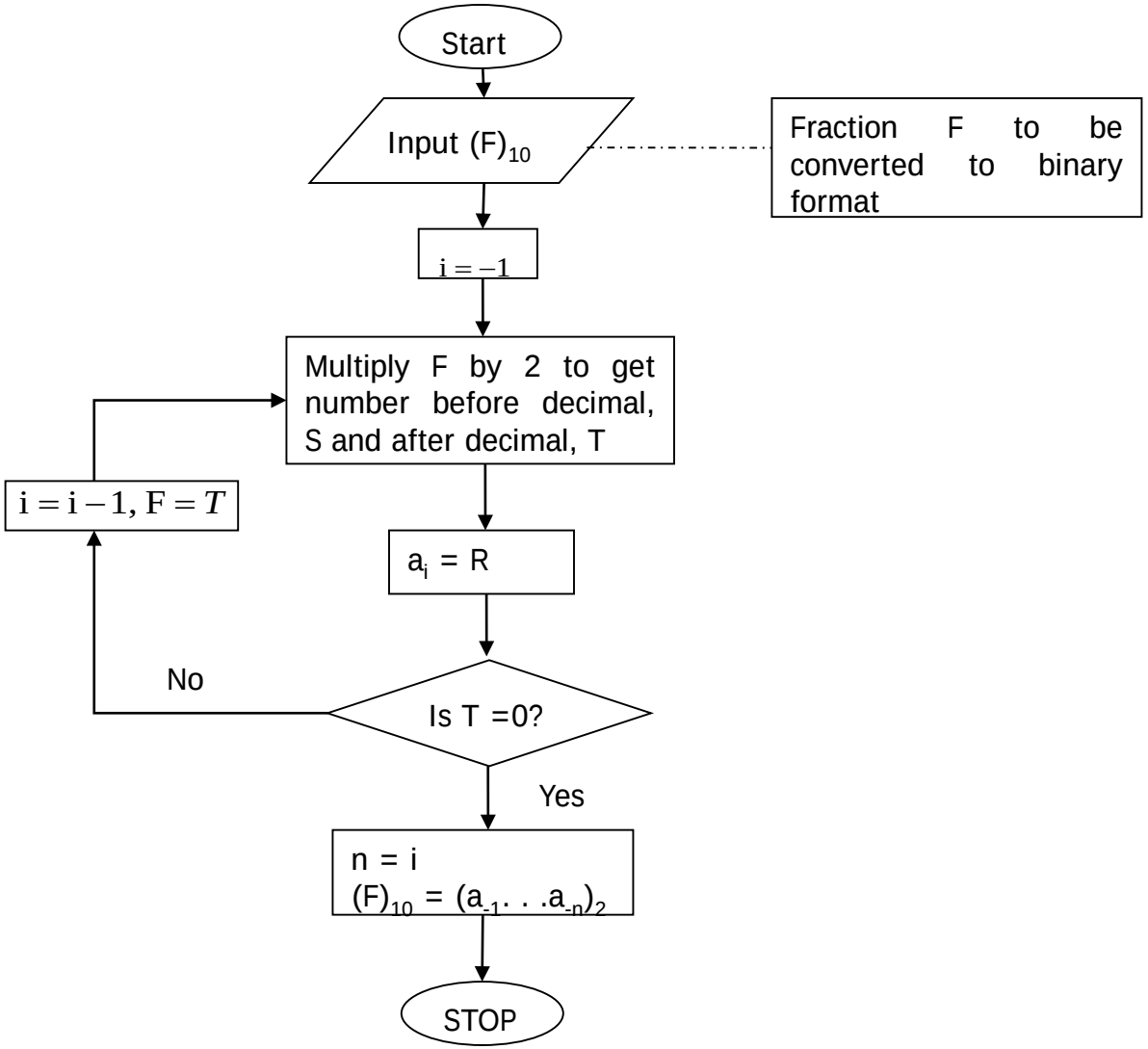
	Number	Number after decimal	Number before decimal
0.1875×2	0.375	0.375	$0 = a_{-1}$
0.375×2	0.75	0.75	$0 = a_{-2}$
0.75×2	1.5	0.5	$1 = a_{-3}$
0.5×2	1.0	0.0	$1 = a_{-4}$

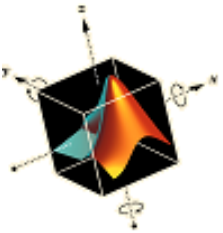
Hence

$$(0.1875)_{10} = (a_{-1}a_{-2}a_{-3}a_{-4})_2$$
$$= (0.0011)_2$$



Conversión de fracción decimal a binario flujograma





Conversión de número decimal a binario

$$(11.1875)_{10} = (\ \ ?.?\ \ )_2$$

Since

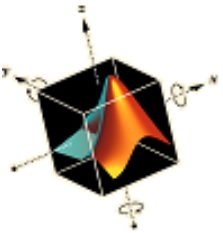
$$(11)_{10} = (1011)_2$$

and

$$(0.1875)_{10} = (0.0011)_2$$

we have

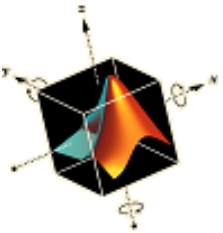
$$(11.1875)_{10} = (1011.0011)_2$$



Representación aproximada de una fracción

	Number	Number after decimal	Number before Decimal
0.3×2	0.6	0.6	$0 = a_{-1}$
0.6×2	1.2	0.2	$1 = a_{-2}$
0.2×2	0.4	0.4	$0 = a_{-3}$
0.4×2	0.8	0.8	$0 = a_{-4}$
0.8×2	1.6	0.6	$1 = a_{-5}$

$$(0.3)_{10} \approx (a_{-1}a_{-2}a_{-3}a_{-4}a_{-5})_2 = (0.01001)_2 = 0.28125$$

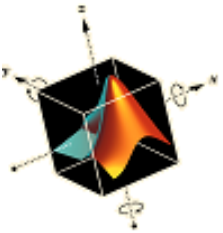


Representación punto flotante decimal

256.78 is written as $+2.5678\times10^{2}$

0.003678 is written as $+3.678\times10^{-3}$

-256.78 is written as -2.5678×10^{2}



Formato punto flotante decimal

256.78 is written as $+2.5678\times10^{2}$

0.003678 is written as $+3.678\times10^{-3}$

-256.78 is written as -2.5678×10^{2}

The form is

$$\text{sign}\times\text{mantissa}\times10^{\text{exponent}}$$

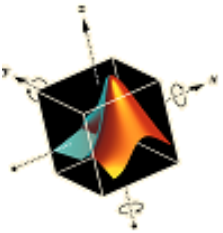
or $\sigma\times m\times10^{e}$

Example: For -2.5678×10^{2}

$\sigma=-1$

$m=2.5678$

$e=2$



Formato punto flotante números binarios

$$y = \sigma \times m \times 2^{e}$$

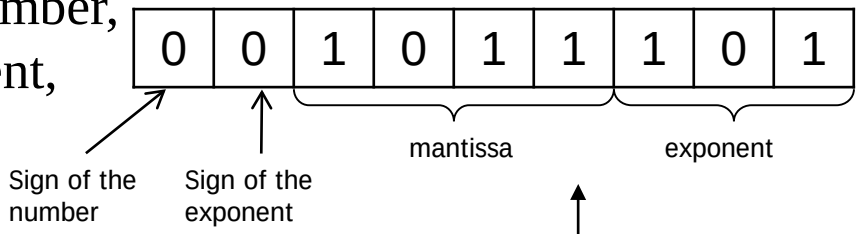
$$\sigma = \text{sign of number} \quad (0 \text{ for } + \text{ve}, 1 \text{ for - ve})$$

$$m = \text{mantissa} \quad [(1)_2 < m < (10)_2]$$

$$e = \text{integer exponent}$$

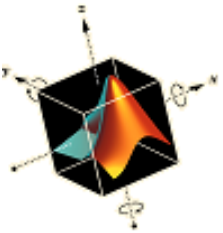
Example: 9 bit-hypothetical word

- the first bit is used for the sign of the number,
- the second bit for the sign of the exponent,
- the next four bits for the mantissa, and
- the next three bits for the exponent



$$(54.75)_{10} = (110110.11)_2 = (1.1011011)_2 \times 2^5$$

$$\cong (1.1011)_2 \times (101)_2$$

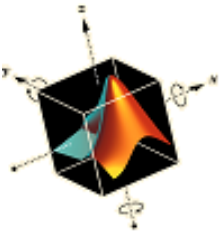


Epsilon de la máquina

- Defined as the measure of accuracy and found by difference between 1 and the next number that can be represented.
- Example: Ten bit word
  - Sign of number
  - Sign of exponent
  - Next four bits for exponent
  - Next four bits for mantissa

	0	0	0	0	0	0	0	0	0	0	$= (1)_{10}$
Next number $\longrightarrow$	0	0	0	0	0	0	0	0	0	1	$= (1.0001)_2 = (1.0625)_{10}$

$$\in_{mach} = 1.0625 - 1 = 2^{-4}$$

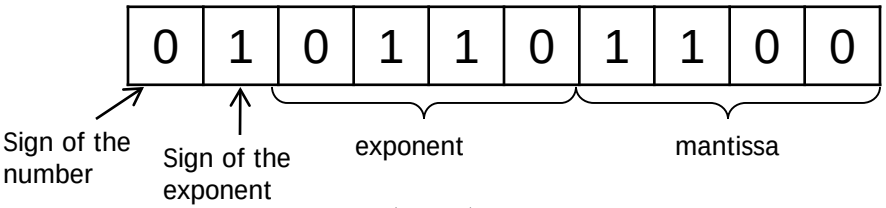


Error relativo y Epsilon de la máquina

- The absolute relative true error in representing a number will be less then the machine epsilon.
- Example: $(0.02832)_{10} \cong (1.1100)_2 \times 2^{-5}$

$$= (1.1100)_2 \times 2^{-(0110)_2}$$

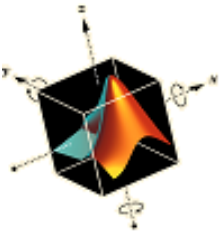
10 bit word (sign, sign of exponent, 4 for exponent, 4 for mantissa)



$$(1.1100)_2 \times 2^{-(0110)_2} = 0.0274375$$

$$\in_a = \left| \frac{0.02832 - 0.0274375}{0.02832} \right|$$

$$= 0.034472 < 2^{-4} = 0.0625$$



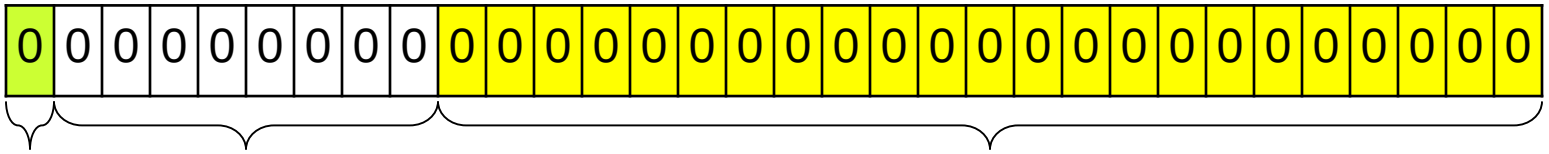
Estándar IEEE754 para representación números punto flotante

- Standardizes representation of floating point numbers on different computers in single and double precision.
- Standardizes representation of floating point operations on different computers.
- Referencia: What every computer scientist should know about floating point arithmetic

http://www.validlab.com/goldberg/paper.pdf

Estándar IEEE754 formato precisión simple

- 32 bits for single precision (s =1, e´=8, m=23)

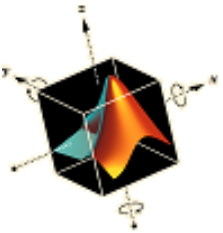


Sign (s)

Biased Exponent (e')

Mantissa (m)

$$\text{Value} = (-1)^s \times (1 \bullet m)_2 \times 2^{e'-127}$$

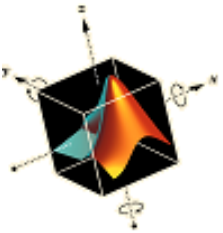


Estándar IEEE754 formato precisión simple

- 8 bits would represent $0 \le e' \le 255$
- Bias is 127; so subtract 127 from representation

$$-127 \le e \le 128$$

- Actual range of e' $1 \le e' \le 254$
- $e' = 0$ and $e' = 255$ are reserved for special numbers
- Actual range of e $-126 \le e \le 127$

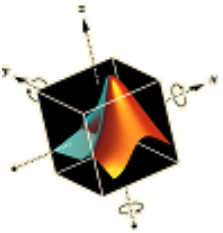


Exponentes y números especiales

$e' = 0$ — all zeros

$e' = 255$ — all ones

s	e'	m	Represents
0	all zeros	all zeros	0
1	all zeros	all zeros	-0
0	all ones	all zeros	∞
1	all ones	all zeros	$-\infty$
0 or 1	all ones	non-zero	NaN



Rango IEEE754 formato precisión simple

- The largest number by magnitude

$$(1.1.....1)_2 \times 2^{127} = 3.40 \times 10^{38}$$

- The smallest number by magnitude

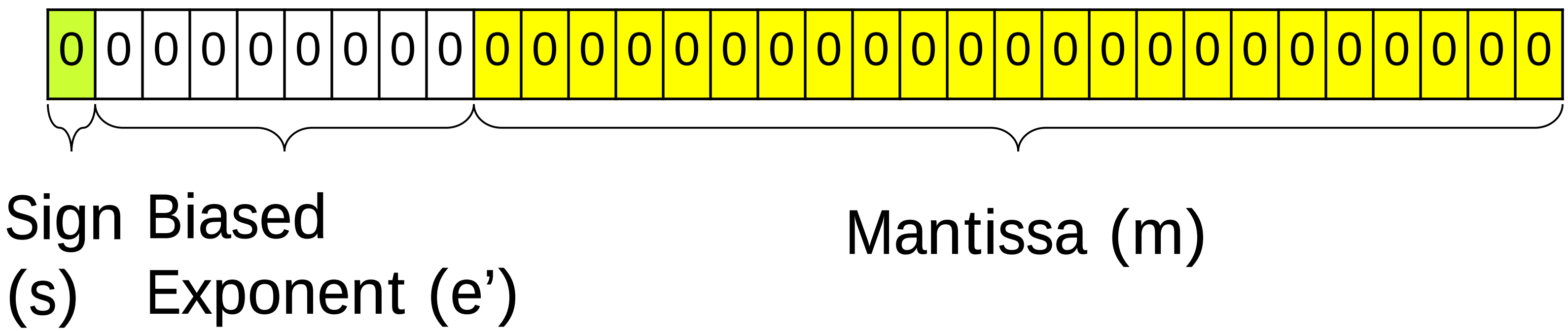
$$(1.00.....0)_2 \times 2^{-126} = 2.18 \times 10^{-38}$$

- Machine epsilon (7 decimal digits)

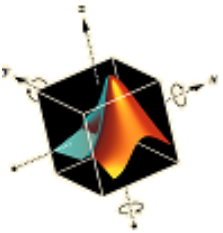
$$\varepsilon_{mach} = 2^{-23} = 1.19 \times 10^{-7}$$

Estándar IEEE754 formato doble precisión

- 64 bits for single precision (s =1, e´=11, m=52)



$$\text{Value} = (-1)^{s} \times (1 \bullet m)_2 \times 2^{e'-1023}$$

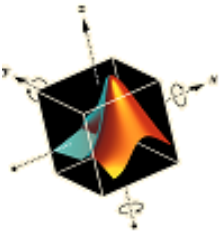


Estándar IEEE754 formato doble precisión

- 11 bits would represent $0 \leq e' \leq 2047$
- Bias is 1023; so subtract 1023 from representation

$$-1023 \leq e \leq 1024$$

- Actual range of e' $1 \leq e' \leq 2046$
- $e' = 0$ and $e' = 2047$ are reserved for special numbers
- Actual range of e $-1022 \leq e \leq 1023$

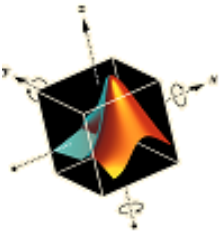


Exponentes y números especiales

$e' = 0$ — all zeros

$e' = 2047$ — all ones

s	e'	m	Represents
0	all zeros	all zeros	0
1	all zeros	all zeros	-0
0	all ones	all zeros	∞
1	all ones	all zeros	$-\infty$
0 or 1	all ones	non-zero	NaN



Rango IEEE754 formato doble precisión

- The largest number by magnitude

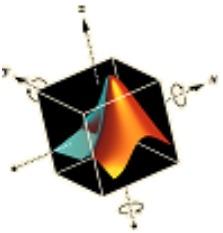
$$(1.1.....1)_2 \times 2^{1023} = 1.7997 \times 10^{308}$$

- The smallest number by magnitude

$$(1.00.....0)_2 \times 2^{-1022} = 2.2251 \times 10^{-308}$$

- Machine epsilon (15 to 16 base 10 digits)

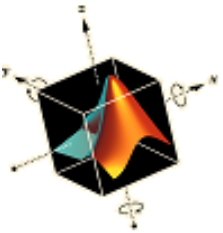
$$\varepsilon_{mach} = 2^{-52} = 2.2204 \times 10^{-16}$$



Explicación y experimentación IEEE754

http://youtu.be/L2YUAIWXlco?list=PLGnRLcmvdTqzq_TjkfAyrVtrXSNOi2e85

Experimentación: http://babbage.cs.qc.edu/IEEE-754/



Propagación de errores

- In numerical methods, the calculations are not made with exact numbers. How do these inaccuracies propagate through the calculations?
- **Example**: Find the bounds for the propagation in adding two numbers. For example if one is calculating X +Y where

$$X = 1.5 \pm 0.05 \qquad Y = 3.4 \pm 0.04$$

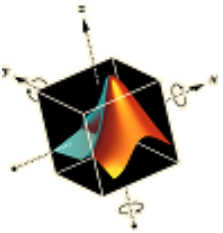
Solution Maximum possible value of $X = 1.55$ and $Y = 3.44$

Maximum possible value of $X + Y = 1.55 + 3.44 = 4.99$

Minimum possible value of $X = 1.45$ and $Y = 3.36$.

Minimum possible value of $X + Y = 1.45 + 3.36 = 4.81$

Hence $4.81 \leq X + Y \leq 4.99$.

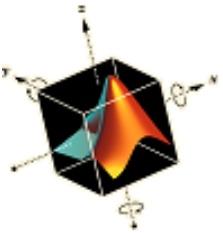


Propagación de errores en fórmulas - ejemplo

- If f is a function of several variables $X_1, X_2, X_3, \ldots\ldots, X_{n-1}, X_n$ then the maximum possible value of the error in f is

$$\Delta f \approx \left|\frac{\partial f}{\partial X_1}\Delta X_1\right| + \left|\frac{\partial f}{\partial X_2}\Delta X_2\right| + \ldots\ldots + \left|\frac{\partial f}{\partial X_{n-1}}\Delta X_{n-1}\right| + \left|\frac{\partial f}{\partial X_n}\Delta X_n\right|$$

- Example: The strain in an axial member of a square cross-section is given by $\in = \dfrac{F}{h^2 E}$
- Given $F = 72 \pm 0.9$ N
  $h = 4 \pm 0.1$ mm
  $E = 70 \pm 1.5$ GPa
- Find the maximum possible error in the measured strain.



Propagación de errores en fórmulas - ejemplo

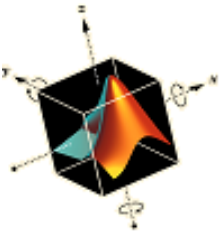
- Solution:

$$\in = \frac{72}{(4\times10^{-3})^2(70\times10^{9})}$$

$$= 64.286\times10^{-6}$$

$$= 64.286\mu$$

$$\Delta\in = \left|\frac{\partial\in}{\partial F}\Delta F\right| + \left|\frac{\partial\in}{\partial h}\Delta h\right| + \left|\frac{\partial\in}{\partial E}\Delta E\right|$$



Propagación de errores en fórmulas - ejemplo

- Solution (cont.):

$$\frac{\partial \in}{\partial F} = \frac{1}{h^2 E} \qquad \frac{\partial \in}{\partial h} = -\frac{2F}{h^3 E} \qquad \frac{\partial \in}{\partial E} = -\frac{F}{h^2 E^2}$$

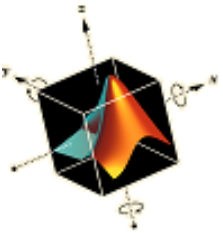
Thus $$\Delta E = \left|\frac{1}{h^2 E}\Delta F\right| + \left|\frac{2F}{h^3 E}\Delta h\right| + \left|\frac{F}{h^2 E^2}\Delta E\right|$$

$$= \left|\frac{1}{(4\times 10^{-3})^2 (70\times 10^9)} \times 0.9\right| + \left|\frac{2\times 72}{(4\times 10^{-3})^3 (70\times 10^9)} \times 0.0001\right|$$

$$+ \left|\frac{72}{(4\times 10^{-3})^2 (70\times 10^9)^2} \times 1.5\times 10^9\right|$$

$$= 5.3955\mu$$

Hence $$\in = (64.286\mu \pm 5.3955\mu)$$



Propagación de errores en fórmula - ejemplo

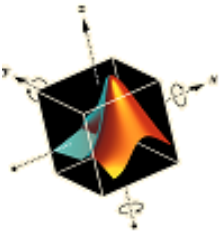
- Subtraction of numbers that are nearly equal can create unwanted inaccuracies. Using the formula for error propagation, show that this is true.
- **Solution**: Let $z = x - y$

Then $$|\Delta z| = \left|\frac{\partial z}{\partial x}\Delta x\right| + \left|\frac{\partial z}{\partial y}\Delta y\right| = |(1)\Delta x| + |(-1)\Delta y| = |\Delta x| + |\Delta y|$$

So the relative change is $$\left|\frac{\Delta z}{z}\right| = \frac{|\Delta x| + |\Delta y|}{|x - y|}$$

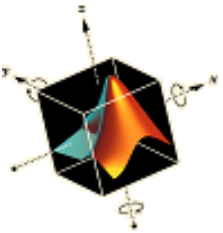
Example $x = 2 \pm 0.001$ $\quad y = 2.003 \pm 0.001$

$$\left|\frac{\Delta z}{z}\right| = \frac{|0.001| + |0.001|}{|\,2 - 2.003\,|} = 0.6667 \quad (\,66.67\%\,)$$



Teorema y Series de Taylor

- The *Taylor theorem* states that any smooth function can be approximated as a polynomial.
- The *Taylor series* provides a means to express this idea mathematically.



Series de Taylor

- The Taylor polynomial of order *n* of a function *f(x)* with *(n+1)* continuous derivatives in the domain [*x*,*x*+*h*] is given by

$$f(x+h) = f(x) + f'(x)h + f''(x)\frac{h^2}{2!} + \cdots + f^{(n)}(x)\frac{h^n}{n!} + R_n(x)$$

where the remainder is given by

$$R_n(x) = \frac{h^{n+1}}{(n+1)!} f^{(n+1)}(c)$$

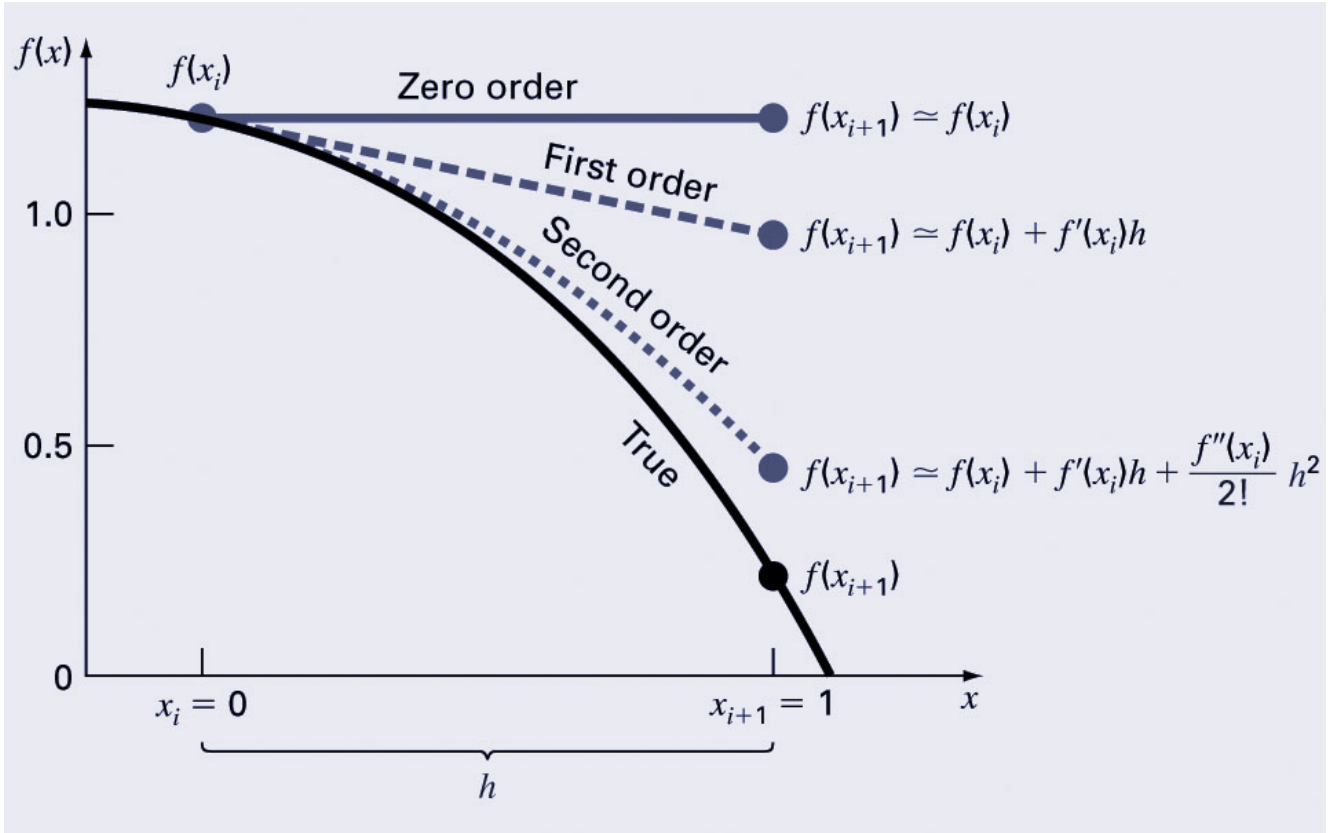
where $x < c < x+h$

that is, c is some point in the domain [x,x+h]

Series de Taylor

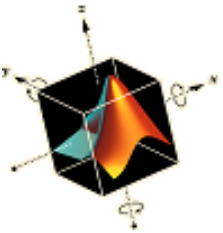
- Another form

$$f(x_{i+1}) = f(x_i) + f'(x_i)h + \frac{f''(x_i)}{2!}h^2 + \frac{f^{(3)}(x_i)}{3!}h^3 + \cdots + \frac{f^{(n)}(x_i)}{n!}h^n + R_n$$



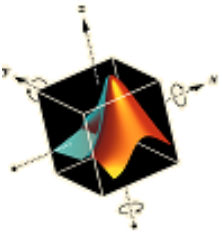
$$f(x_{i+1}) = \sum_{n=0}^{\infty} \frac{f^n(x_i)}{n!}h^2$$

$$x_{i+1} = x_i + h$$



Error de truncamiento en Series de Taylor

- In general, the *n*th order Taylor series expansion will be exact for an *n*th order polynomial.
- In other cases, the remainder term R_n is of the order of h^{n+1}, meaning:
  - The more terms are used, the smaller the error, and
  - The smaller the spacing, the smaller the error for a given number of terms.



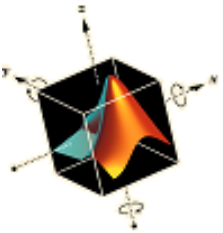
Series de Taylor - ejemplos

- Some examples of Taylor series which are widely known and used

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots$$

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots$$

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$$



Series de Taylor - ejemplo

- Find the value of $f(6)$ given that

  $f(4)=125,\ f'(4)=74,\ f''(4)=30,\ f'''(4)=6$

  and all other higher order derivatives $f(x)$ at $x=4$ are zero

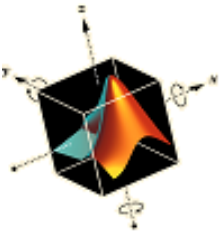
  Solution: $f(x+h)=f(x)+f'(x)h+f''(x)\frac{h^2}{2!}+f'''(x)\frac{h^3}{3!}+\cdots$

  $x=4 \qquad h=6-4=2$

  Since the higher order derivatives are zero,

$$f(4+2)=f(4)+f'(4)2+f''(4)\frac{2^2}{2!}+f'''(4)\frac{2^3}{3!}$$

$$f(6)=125+74(2)+30\left(\frac{2^2}{2!}\right)+6\left(\frac{2^3}{3!}\right)=125+148+60+8=341$$



Deducción de la serie de Maclaurin para e^x

- Derive the Maclaurin series $e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$

  the Maclaurin series is simply the Taylor series about the point x=0

  Solution:

$$f(x+h) = f(x) + f'(x)h + f''(x)\frac{h^2}{2!} + f'''(x)\frac{h^3}{3!} + f''''(x)\frac{h^4}{4} + f'''''(x)\frac{h^5}{5} + \cdots$$

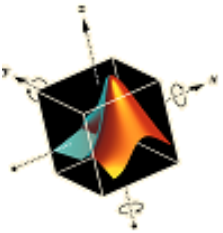
$$f(0+h) = f(0) + f'(0)h + f''(0)\frac{h^2}{2!} + f'''(0)\frac{h^3}{3!} + f''''(0)\frac{h^4}{4} + f'''''(0)\frac{h^5}{5} + \cdots$$

  Since $f(x) = e^x$, $f'(x) = e^x$, $f''(x) = e^x, \ldots, f^n(x) = e^x$ and $f^n(0) = e^0 = 1$

  the Maclaurin series is then

$$f(h) = (e^0) + (e^0)h + \frac{(e^0)}{2!}h^2 + \frac{(e^0)}{3!}h^3 \ldots = 1 + h + \frac{1}{2!}h^2 + \frac{1}{3!}h^3 \ldots$$

  so, $f(x) = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots$



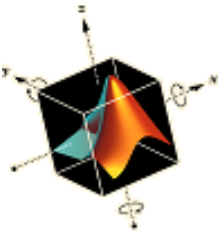
Error en Series de Taylor - ejemplo

- The Taylor series for e^x at point $x = 0$ is given by

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \frac{x^5}{5!} + \cdots$$

- It can be seen that as the number of terms used increases, the error bound decreases and hence a better estimate of the function can be found.

- How many terms would it require to get an approximation of e^1 within a magnitude of true error of less than 10^{-6}.



Error en Series de Taylor - ejemplo

- Solution:

  Using $(n+1)$ terms of Taylor series gives error bound of

$$R_n(x) = \frac{(x-h)^{n+1}}{(n+1)!} f^{(n+1)}(c)$$

$$x = 0,\ h = 1,\ f(x) = e^x$$

$$R_n(0) = \frac{(0-1)^{n+1}}{(n+1)!} f^{(n+1)}(c) = \frac{(-1)^{n+1}}{(n+1)!} e^c$$

- Since $x < c < x+h$

$$0 < c < 0+1$$

$$0 < c < 1$$

$$\frac{1}{(n+1)!} < |R_n(0)| < \frac{e}{(n+1)!}$$

- So if we want to find out how many terms it would require to get an approximation of e^1 within a magnitude of true error of less than 10^{-6}

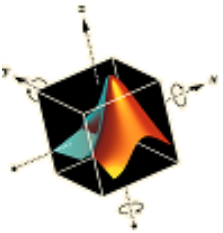
$$\frac{e}{(n+1)!} < 10^{-6}$$

$$(n+1)! > 10^6 e$$

$$(n+1)! > 10^6 \times 3$$

$$n \geq 9$$

So 9 terms or more are needed to get a true error less than 10^{-6}



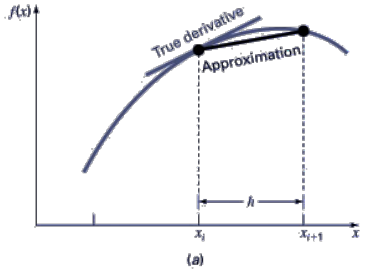
Diferenciación numérica con Series de Taylor

- The first order Taylor series can be used to calculate approximations to derivatives:
  - Given: $f(x_{i+1}) = f(x_i) + f'(x_i)h + O(h^2)$
  - Then: $f'(x_i) = \frac{f(x_{i+1}) - f(x_i)}{h} + O(h)$
- This is termed a “forward” difference because it utilizes data at *i* and *i*+1 to estimate the derivative.

Diferenciación numérica con Series de Taylor

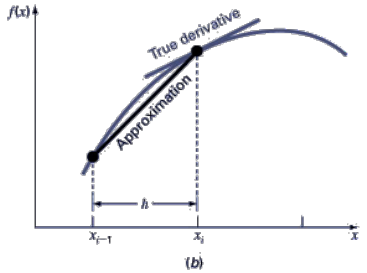
- There are also backward and centered difference approximations, depending on the points used:
- Forward:

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_i)}{h} + O(h)$$



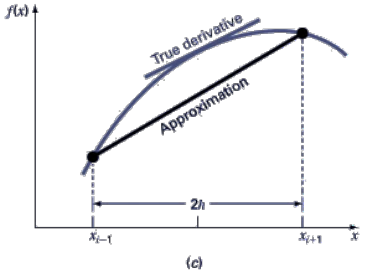
- Backward:

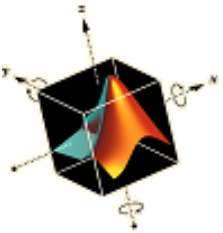
$$f'(x_i) = \frac{f(x_i) - f(x_{i-1})}{h} + O(h)$$



- Centered:

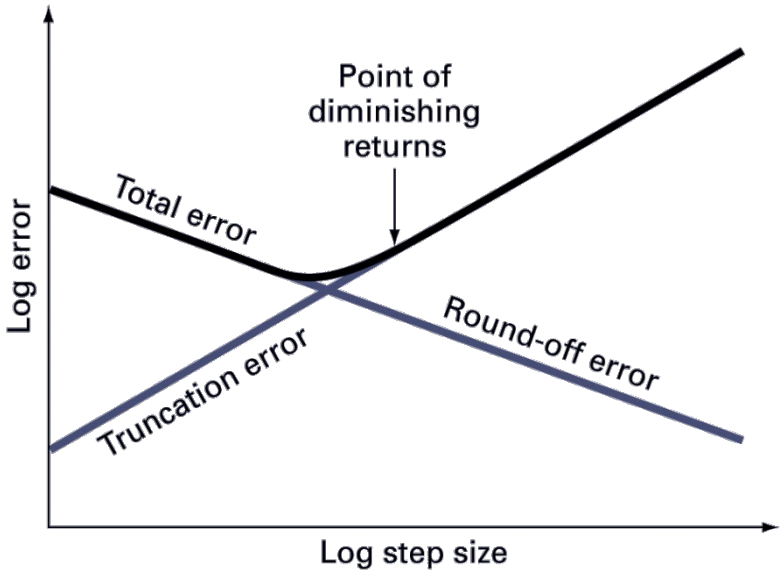
$$f'(x_i) = \frac{f(x_{i+1}) - f(x_{i-1})}{2h} + O(h^2)$$

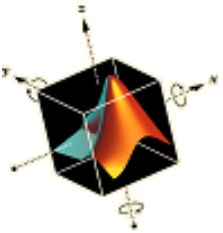




Error numérico total

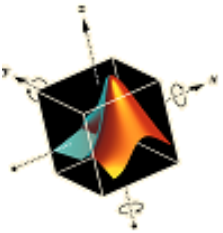
- The *total numerical error* is the summation of the truncation and round off errors.
- The truncation error generally *increases* as the step size increases, while the round off error *decreases* as the step size increases - this leads to a point of diminishing returns for step size.





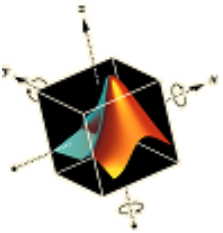
Otros errores

- *Blunders* - errors caused by malfunctions of the computer or human imperfection.
- *Model errors* - errors resulting from incomplete mathematical models.
- *Data uncertainty* - errors resulting from the accuracy and/or precision of the data.



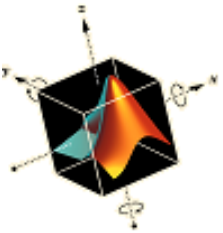
Notación asintótica O grande

- Big O notation has two main areas of application:
  - In mathematics, it is commonly used to describe how closely a finite series approximates a given function, especially in the case of a truncated Taylor series or asymptotic expansion (Infinitesimal asymptotics). It describe the error term in an approximation to a mathematical function.
  - In computer science, it is useful in the analysis of algorithms.



Notación asintótica O grande

- Big O notation (or Landau symbols) has two main areas of application:
  - In mathematics, it is commonly used to describe how closely a finite series approximates a given function, especially in the case of a truncated Taylor series or asymptotic expansion (Infinitesimal asymptotics). It describe the error term in an approximation to a mathematical function.
  - In computer science, it is useful in the analysis of algorithms such as numbers of operations.



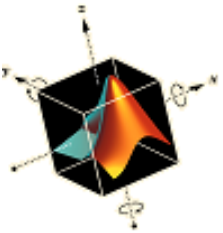
Notación asintótica O grande

- Definición 1: If two functions f, g satisfy $f(n) \leq Cg(n)$ for some constant $C > 0 \quad \forall n > N$

  we write $f(n) = O(g(n))$ as $n \to \infty$
- Definición 2: If two functions f, g that approach 0 as $x \to 0$, satisfy $f(x) \leq Cg(x)$ for some constant $C > 0 \quad \forall x < x_0$

  we write $f(x) = O(g(x))$ as $x \to 0$

  We can think of g as an asymptotic upper bound for f

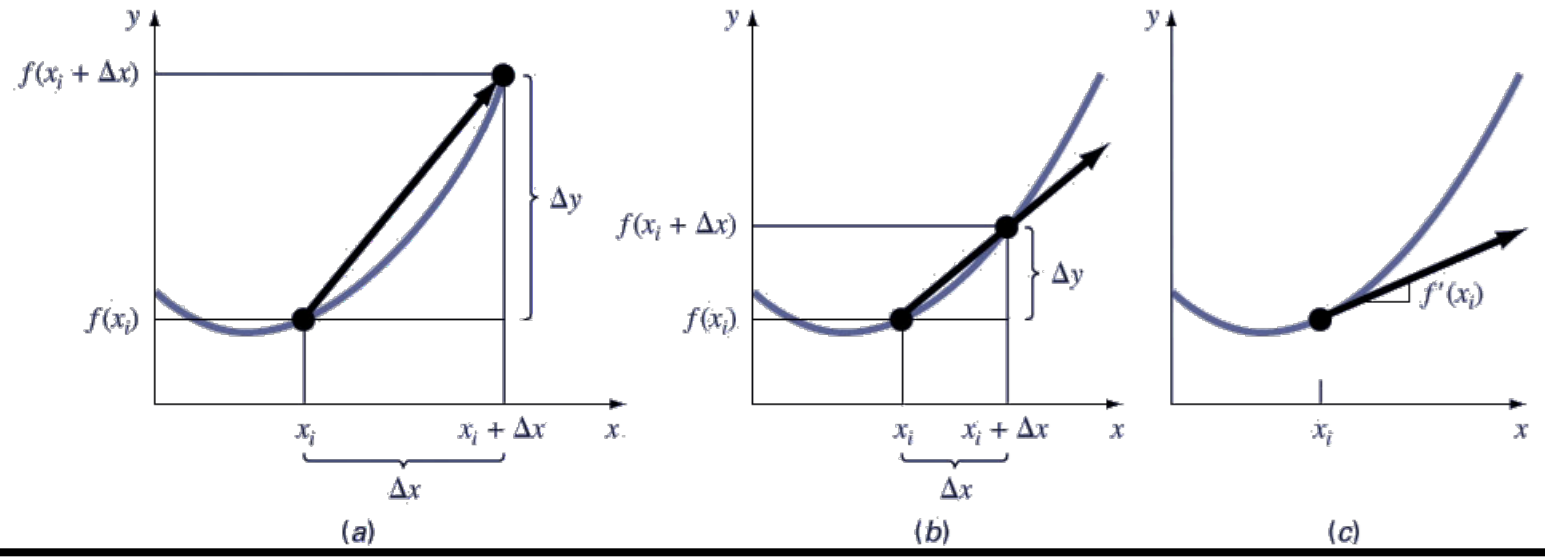


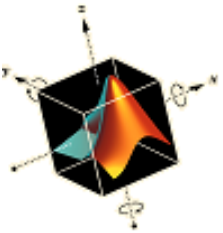
Diferenciación

Diferenciación de funciones continuas

- The mathematical definition of a derivative begins with a difference approximation: $\frac{\Delta y}{\Delta x} = \frac{f(x_i + \Delta x) - f(x_i)}{\Delta x}$

  and as Δx is allowed to approach zero, the difference becomes a derivative: $\frac{dy}{dx} = \lim_{\Delta x \to 0} \frac{f(x_i + \Delta x) - f(x_i)}{\Delta x}$





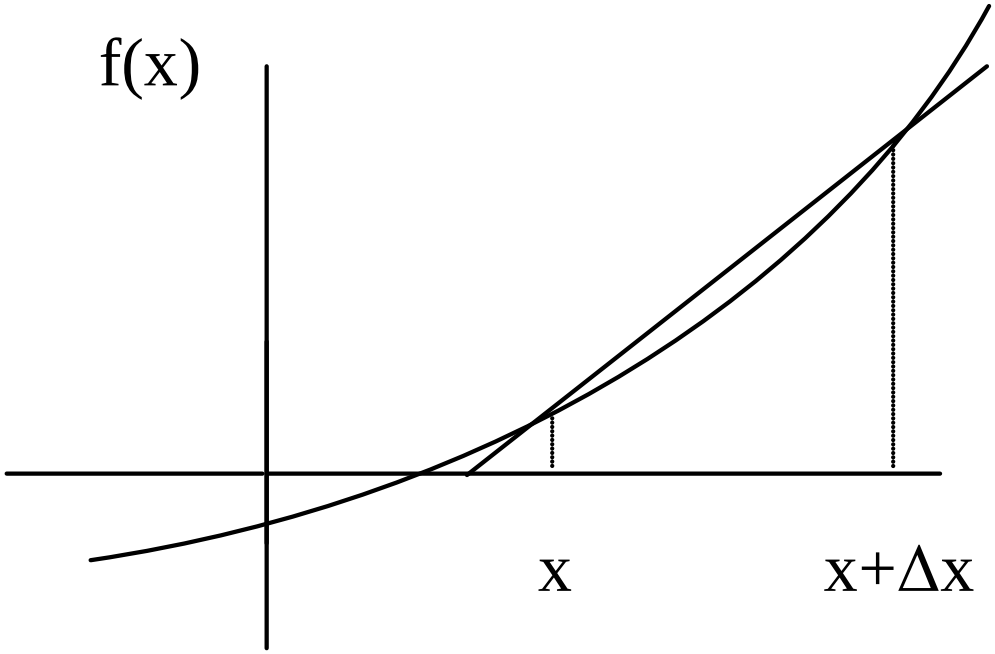
Fórmulas de diferenciación de gran exactitud

- Taylor series expansion can be used to generate high-accuracy formulas for derivatives by using linear algebra to combine the expansion around several points.
- Three categories for the formula include *forward finite-difference*, *backward finite-difference*, and *centered finite-difference*.

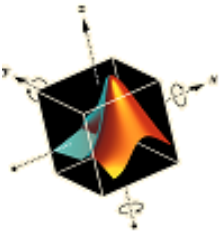
Aproximación por diferencia en adelanto

$$f'(x) = \lim_{\Delta x \to 0} \frac{f(x+\Delta x) - f(x)}{\Delta x}$$

For a finite 'Δx' $$f'(x) \approx \frac{f(x+\Delta x) - f(x)}{\Delta x}$$



Graphical Representation of forward difference approximation of first derivative.



Aproximación por diferencia en adelanto - ejemplo

- The velocity of a rocket is given by

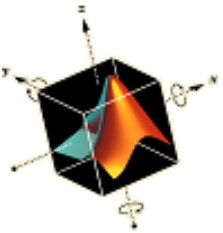
$$v(t) = 2000 \ln\left[\frac{14\times10^4}{14\times10^4 - 2100t}\right] - 9.8t, \quad 0 \le t \le 30$$

- Where 'v' is given in m/s and 't' is given in seconds.

a) Use forward difference approximation of the first derivative of $v(t)$ to calculate the acceleration at $t = 16s$. Use a step size of $\Delta t = 2s$.

b) Find the exact value of the acceleration of the rocket.

c) Calculate the absolute relative true error for part (b).



Aproximación por diferencia en adelanto - ejemplo

- Solution

a) $$a(t_i) \approx \frac{v(t_{i+1}) - v(t_i)}{\Delta t} \qquad t_i = 16 \qquad \Delta t = 2$$
$$t_{i+1} = t_i + \Delta t = 16 + 2 = 18$$

$$a(16) \approx \frac{v(18) - v(16)}{2}$$

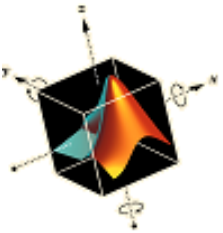
$$v(18) = 2000 \ln\left[\frac{14 \times 10^4}{14 \times 10^4 - 2100(18)}\right] - 9.8(18) = 453.02 \text{ m/s}$$

$$v(16) = 2000 \ln\left[\frac{14 \times 10^4}{14 \times 10^4 - 2100(16)}\right] - 9.8(16) = 392.07 \text{ m/s}$$

Hence
$$a(16) \approx \frac{v(18) - v(16)}{2} \approx \frac{453.02 - 392.07}{2} \approx 30.474 \text{m/s}^2$$

b) The exact value of $a(16)$ can be calculated by differentiating

$$v(t) = 2000 \ln\left[\frac{14 \times 10^4}{14 \times 10^4 - 2100t}\right] - 9.8t \quad \text{as} \quad a(t) = \frac{d}{dt}\left[v(t)\right]$$



Aproximación por diferencia en adelanto - ejemplo

Knowing that

$$\frac{d}{dt}\left[\ln(t)\right]=\frac{1}{t} \quad \text{and} \quad \frac{d}{dt}\left[\frac{1}{t}\right]=-\frac{1}{t^2}$$

$$a(t)=2000\left(\frac{14\times10^4-2100t}{14\times10^4}\right)\frac{d}{dt}\left(\frac{14\times10^4}{14\times10^4-2100t}\right)-9.8$$

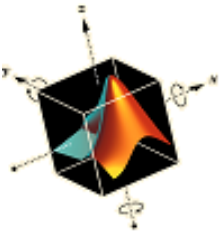
$$=2000\left(\frac{14\times10^4-2100t}{14\times10^4}\right)(-1)\left(\frac{14\times10^4}{\left(14\times10^4-2100t\right)^2}\right)(-2100)-9.8$$

$$=\frac{-4040-29.4t}{-200+3t}$$

$$a(16)=\frac{-4040-29.4(16)}{-200+3(16)}=29.674\text{m/s}^2$$

c) The absolute relative true error is

$$|\epsilon_t|=\left|\frac{\text{True Value - Approximate Value}}{\text{True Value}}\right|\cdot100=\left|\frac{29.674-30.474}{29.674}\right|x100=2.6967\ \%$$



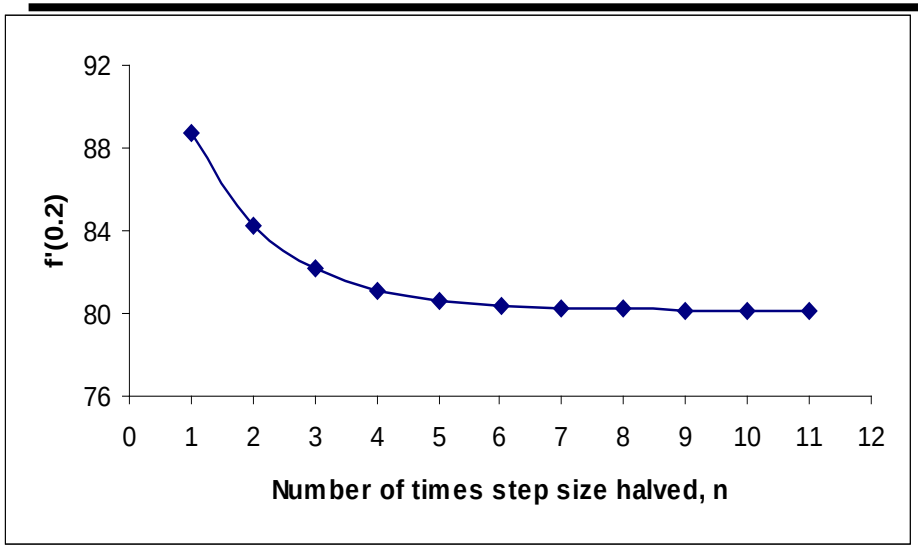
Efecto del tamaño de paso en el método de diferencia dividida en adelanto

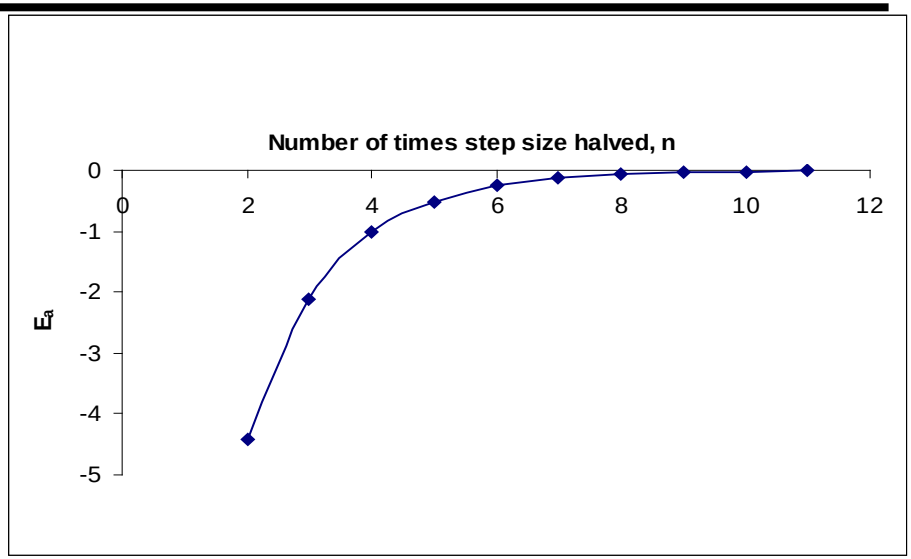
$$f(x) = 9e^{4x}$$

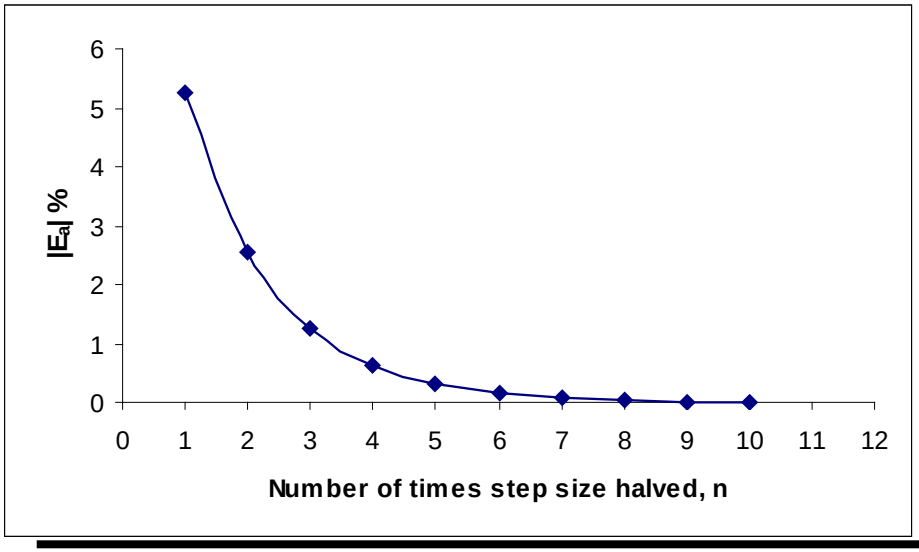
Value of´ $f'(0.2)$ using forward difference method.

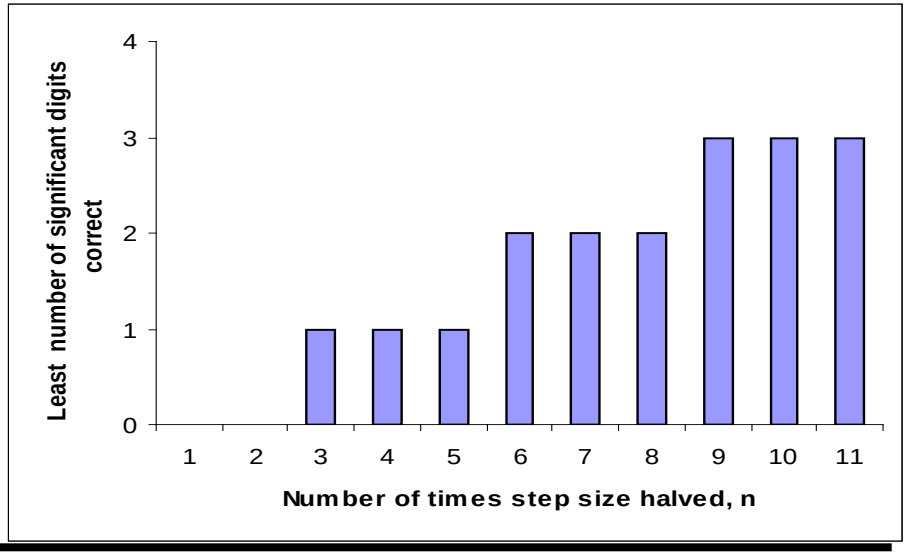
h	$f'(0.2)$	E_a	$\lvert\varepsilon_a\rvert\%$	Significant digits	E_t	$\lvert\varepsilon_t\rvert\%$
0.05	88.69336				-8.57389	10.70138
0.025	84.26239	-4.430976	5.258546	0	-4.14291	5.170918
0.0125	82.15626	-2.106121	2.563555	1	-2.03679	2.542193
0.00625	81.12937	-1.0269	1.265756	1	-1.00989	1.260482
0.003125	80.62231	-0.507052	0.628923	1	-0.50284	0.627612
0.001563	80.37037	-0.251944	0.313479	2	-0.25090	0.313152
0.000781	80.24479	-0.125579	0.156494	2	-0.12532	0.156413
0.000391	80.18210	-0.062691	0.078186	2	-0.06263	0.078166
0.000195	80.15078	-0.031321	0.039078	3	-0.03130	0.039073
9.77E-05	80.13512	-0.015654	0.019535	3	-0.01565	0.019534
4.88E-05	80.12730	-0.007826	0.009767	3	-0.00782	0.009766

Efecto del tamaño de paso en el método de diferencia dividida en adelanto

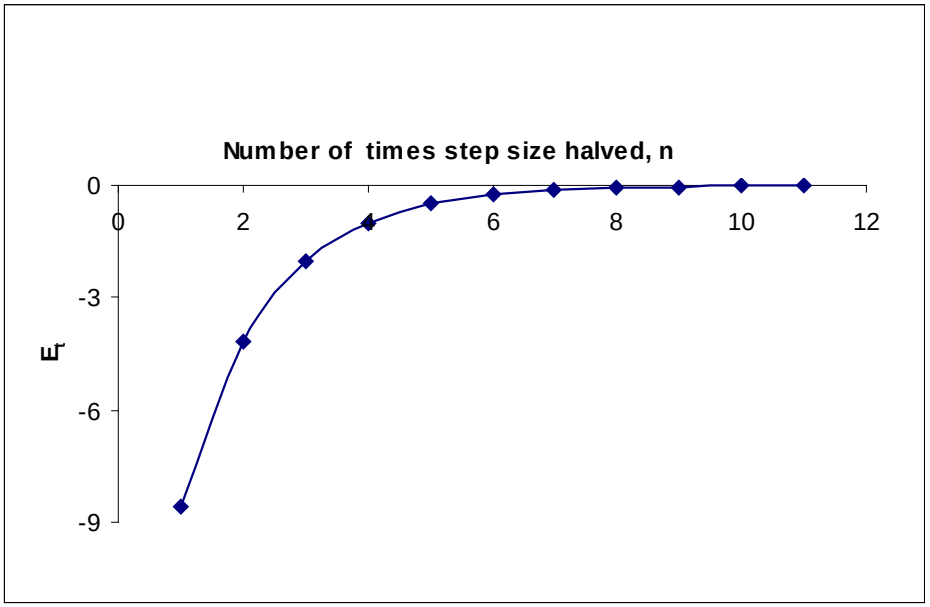


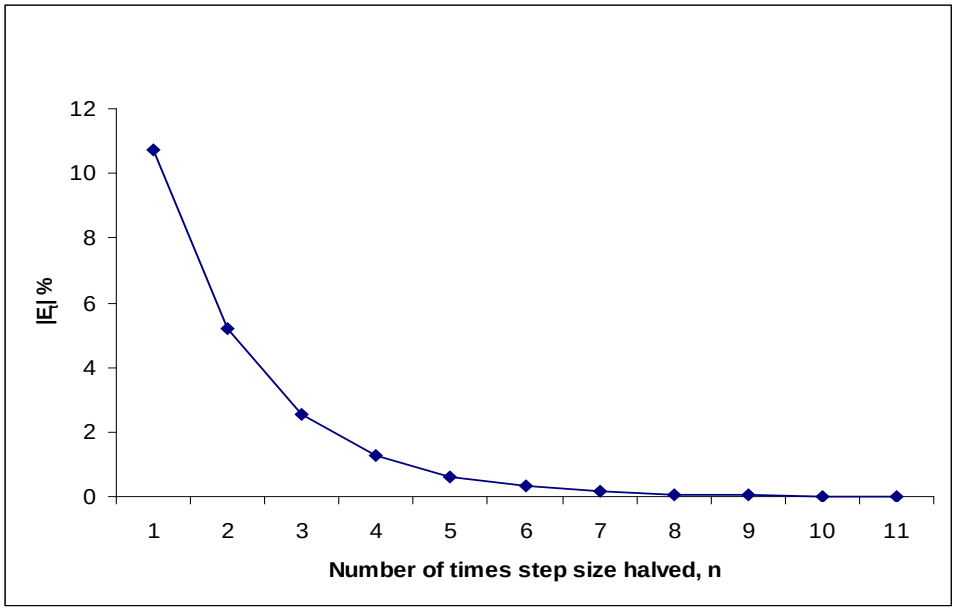


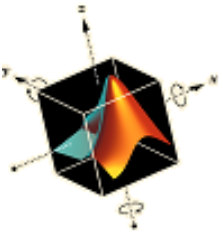




Efecto del tamaño de paso en el método de diferencia dividida en adelanto







Aproximación por diferencia en atraso

- Sabemos que $$f'(x) = \lim_{\Delta x \to 0} \frac{f(x+\Delta x) - f(x)}{\Delta x}$$
- For a finite $'\Delta x'$, $$f'(x) \approx \frac{f(x+\Delta x) - f(x)}{\Delta x}$$
- If $'\Delta x'$ is chosen as a negative number,

$$f'(x) \approx \frac{f(x-\Delta x) - f(x)}{-\Delta x} = \frac{f(x) - f(x-\Delta x)}{\Delta x}$$

Aproximación por diferencia en atraso

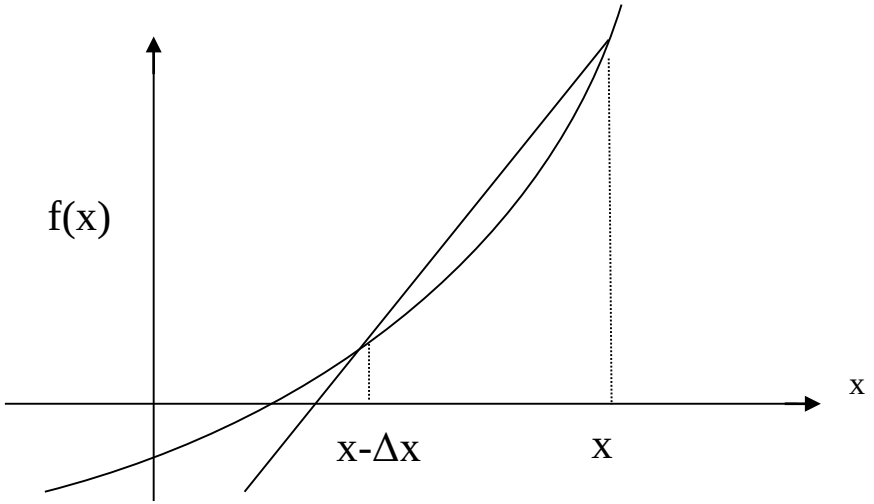
- This is a backward difference approximation as you are taking a point backward from x. To find the value of $f'(x)$ at $x = x_i$, we may choose another point 'Δx' behind as $x = x_{i-1}$. This gives

$$f'(x_i) \approx \frac{f(x_i) - f(x_{i-1})}{\Delta x}$$

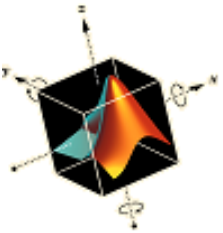
$$= \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}}$$

where

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



Graphical Representation of backward difference approximation of first derivative.



Aproximación por diferencia en atraso - ejemplo

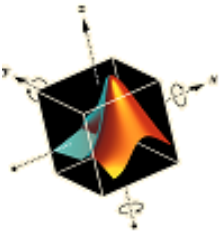
- The velocity of a rocket is given by

$$v(t) = 2000 \ln\left[\frac{14\times10^4}{14\times10^4 - 2100t}\right] - 9.8t, \quad 0 \le t \le 30$$

- Where 'v' is given in m/s and 't' is given in seconds.

a) Use backward difference approximation of the first derivative of $v(t)$ to calculate the acceleration at $t = 16s$. Use a step size of $\Delta t = 2s$.

b) Calculate the absolute relative true error for part (a).



Aproximación por diferencia en atraso - ejemplo

- Solution

a) $$a(t_i) \approx \frac{v(t_i) - v(t_{i-1})}{\Delta t} \qquad t_i = 16 \qquad \Delta t = 2$$
$$t_{i-1} = t_i - \Delta t = 16 - 2 = 14$$

$$a(16) \approx \frac{v(16) - v(14)}{2}$$

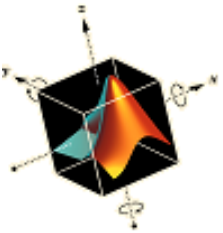
$$v(16) = 2000 \ln\left[\frac{14\times10^4}{14\times10^4 - 2100(16)}\right] - 9.8(16) = 392.07 \text{ m/s}$$

$$v(14) = 2000 \ln\left[\frac{14\times10^4}{14\times10^4 - 2100(14)}\right] - 9.8(14) = 334.24 \text{ m/s}$$

Hence
$$a(16) \approx \frac{v(16) - v(14)}{2} \approx \frac{392.07 - 334.24}{2} \approx 28.915 \text{ m/s}^2$$

b) The absolute relative true error is knowing that the exact value at $t = 16s$ is $a(16) = 29.674 \text{ m/s}^2$

$$|\epsilon_t| = \left|\frac{29.674 - 28.915}{29.674}\right| \cdot 100 = 2.5584\%$$



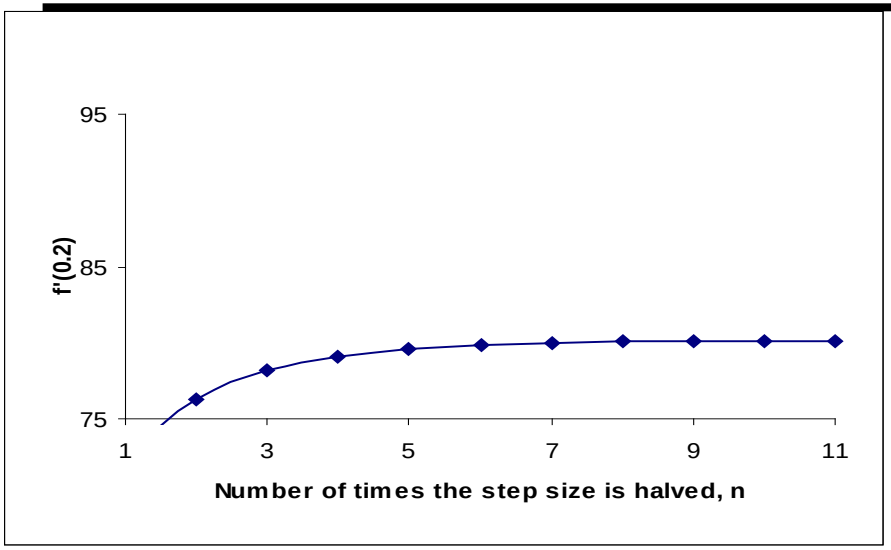
Efecto del tamaño de paso en el método de diferencia dividida en atraso

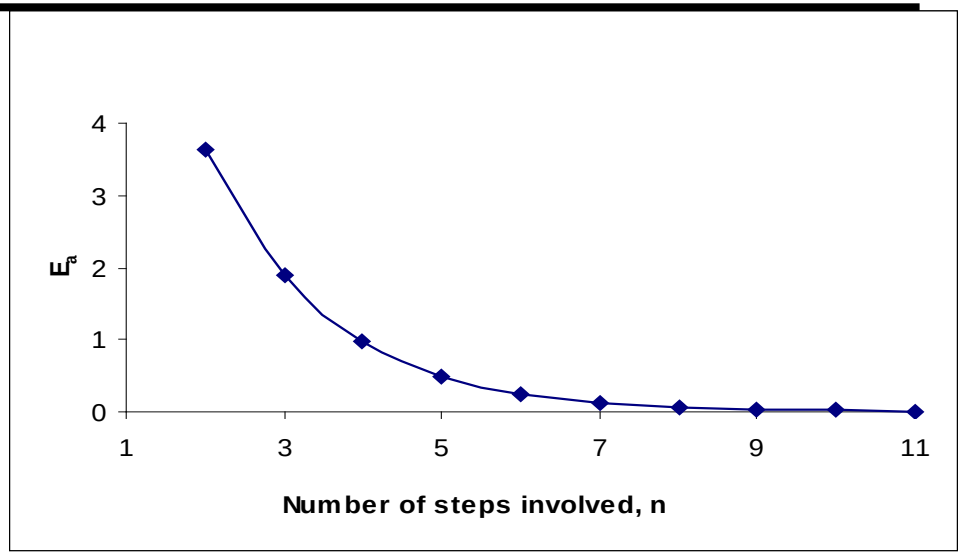
$$f(x) = 9e^{4x}$$

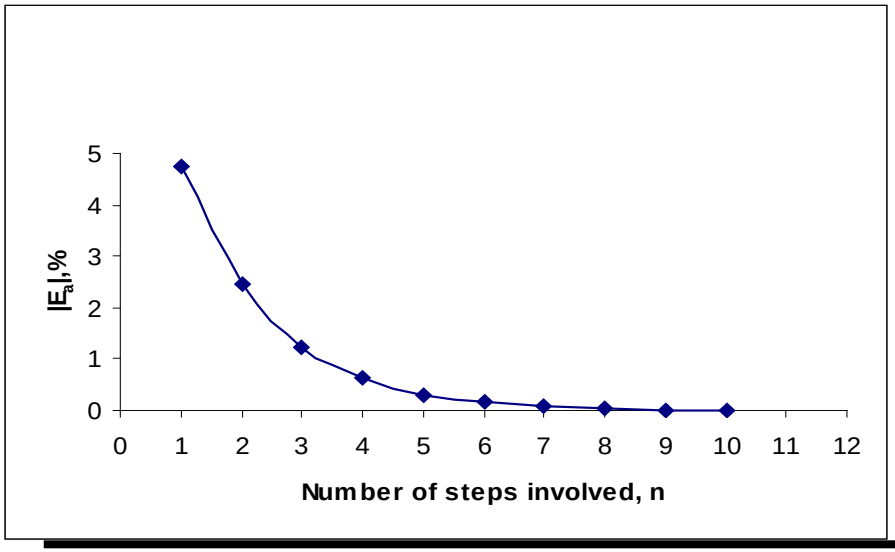
Value of´ $f'(0.2)$ using forward difference method.

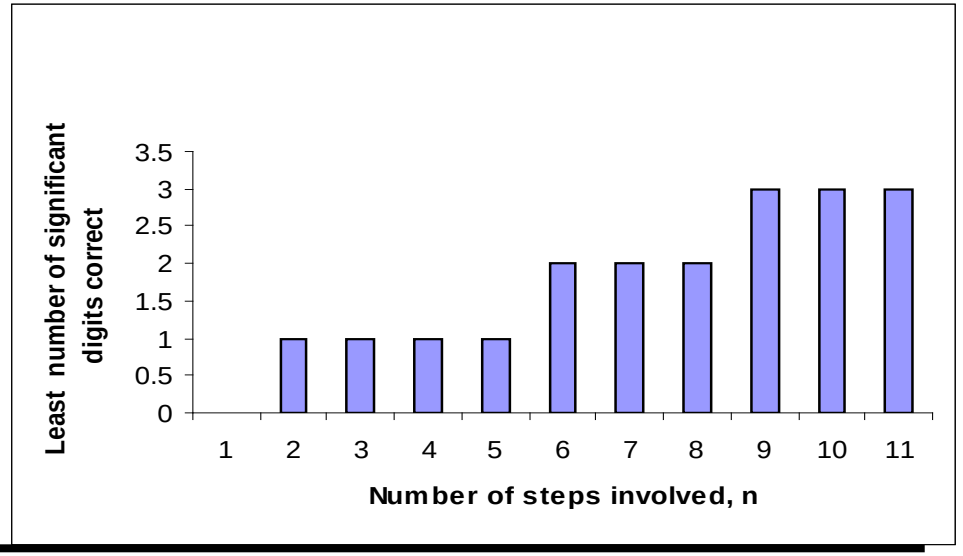
h	$f'(0.2)$	E_a	$\lvert\varepsilon_a\rvert\%$	Significant digits	E_t	$\lvert\varepsilon_t\rvert\%$
0.05	72.61598				7.50349	9.365377
0.025	76.24376	3.627777	4.758129	1	3.87571	4.837418
0.0125	78.14946	1.905697	2.438529	1	1.97002	2.458849
0.00625	79.12627	0.976817	1.234504	1	0.99320	1.239648
0.003125	79.62081	0.494533	0.62111	1	0.49867	0.622404
0.001563	79.86962	0.248814	0.311525	2	0.24985	0.31185
0.000781	79.99442	0.124796	0.156006	2	0.12506	0.156087
0.000391	80.05691	0.062496	0.078064	2	0.06256	0.078084
0.000195	80.08818	0.031272	0.039047	3	0.03129	0.039052
9.77E-05	80.10383	0.015642	0.019527	3	0.01565	0.019529
4.88E-05	80.11165	0.007823	0.009765	3	0.00782	0.009765

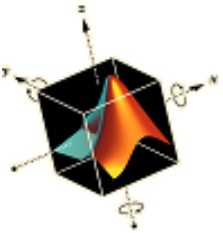
Efecto del tamaño de paso en el método de diferencia dividida en atraso



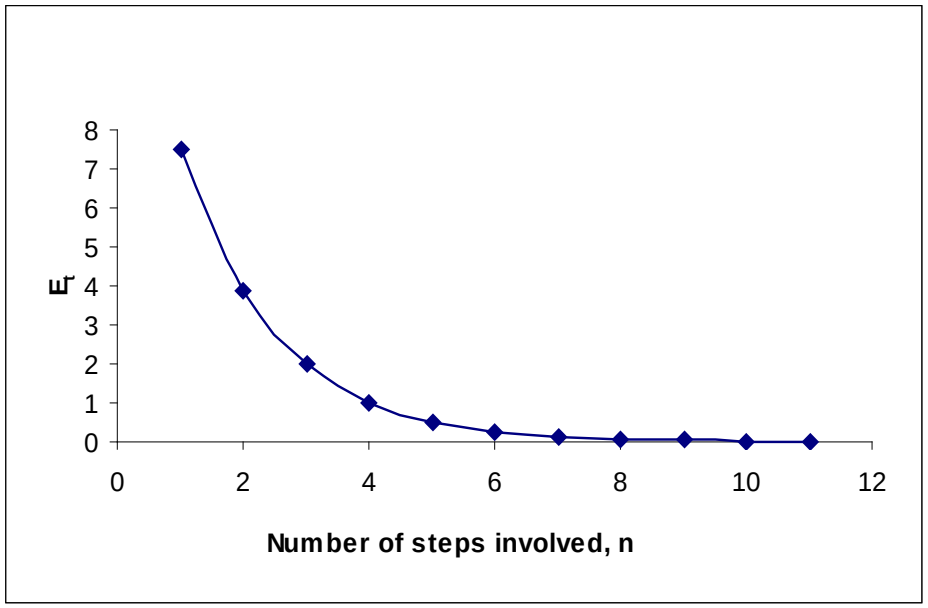


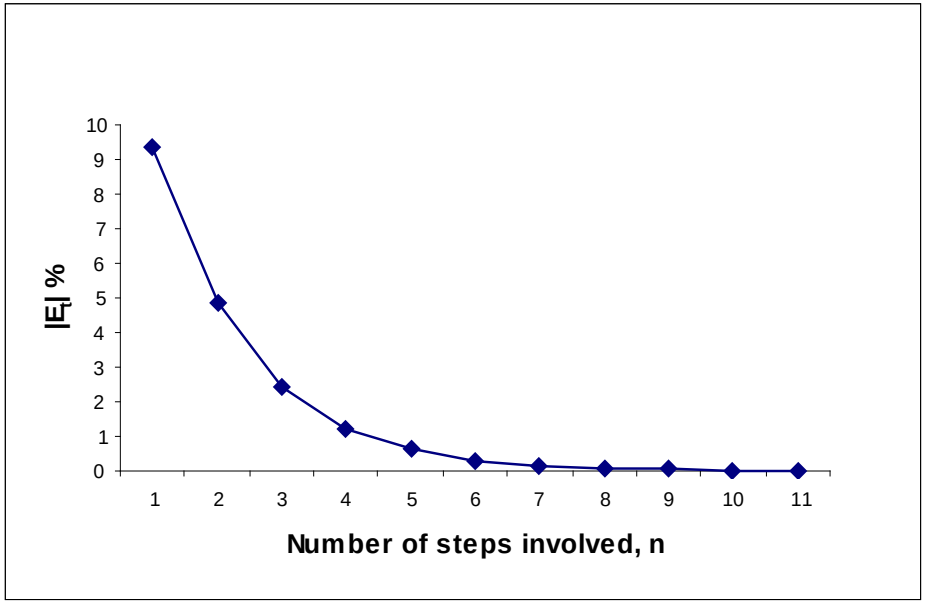


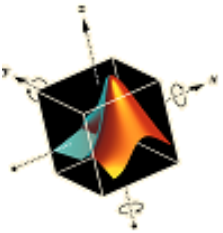




Efecto del tamaño de paso en el método de diferencia dividida en atraso







Obtención de la adad a partir de las series de Taylor

- Taylor's theorem says that if you know the value of a function f at a point x_i and all its derivatives at that point, provided the derivatives are continuous between x_i and x_{i+1}, then

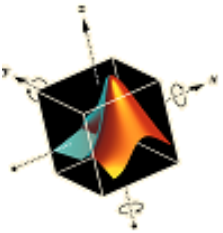
$$f(x_{i+1}) = f(x_i) + f'(x_i)(x_{i+1} - x_i) + \frac{f''(x_i)}{2!}(x_{i+1} - x_i)^2 + \dots$$

Substituting for convenience $\Delta x = x_{i+1} - x_i$

$$f(x_{i+1}) = f(x_i) + f'(x_i)\Delta x + \frac{f''(x_i)}{2!}(\Delta x)^2 + \dots$$

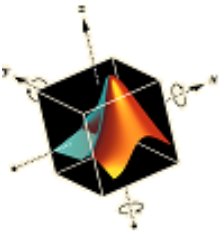
$$f'(x_i) = \frac{f(x_{i+1}) - f(x_i)}{\Delta x} - \frac{f''(x_i)}{2!}(\Delta x) + \dots$$

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_i)}{\Delta x} + O(\Delta x)$$



Obtención de la adad a partir de las series de Taylor

- The $O(\Delta x)$ term shows that the error in the approximation is of the order of Δx. It is easy to derive from Taylor series the formula for backward divided difference approximation of the first derivative.
- As shown above, both forward and backward divided difference approximation of the first derivative are accurate on the order of $O(\Delta x)$.
- Can we get better approximations? Yes, another method is called the **Central difference approximation** of the first derivative.



Obtención de la adc a partir de las series de Taylor

- From Taylor series

$$f(x_{i+1}) = f(x_i) + f'(x_i)\Delta x + \frac{f''(x_i)}{2!}(\Delta x)^2 + \frac{f'''(x_i)}{3!}(\Delta x)^3 + \ldots \qquad (1)$$

$$f(x_{i-1}) = f(x_i) - f'(x_i)\Delta x + \frac{f''(x_i)}{2!}(\Delta x)^2 - \frac{f'''(x_i)}{3!}(\Delta x)^3 + \ldots \qquad (2)$$

Subtracting equation (2) from equation (1)

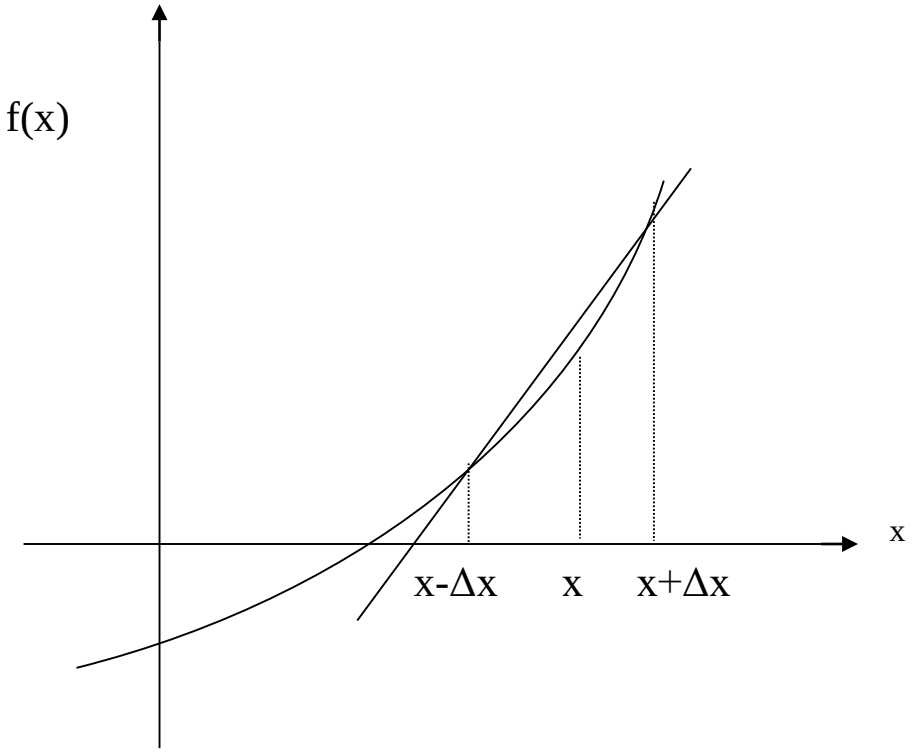
$$f(x_{i+1}) - f(x_{i-1}) = f'(x_i)(2\Delta x) + \frac{2f'''(x_i)}{3!}(\Delta x)^3 + \ldots$$

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_{i-1})}{2\Delta x} - \frac{f'''(x_i)}{3!}(\Delta x)^2 + \ldots$$

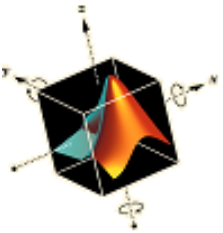
$$f'(x_i) = \frac{f(x_{i+1}) - f(x_{i-1})}{2\Delta x} + O(\Delta x)^2$$

Obtención de la adc a partir de las series de Taylor

- Hence showing that we have obtained a more accurate formula as the error is of the order of $O(\Delta x)^2$



Graphical Representation of central difference approximation of first derivative



Aproximación por diferencia central - ejemplo

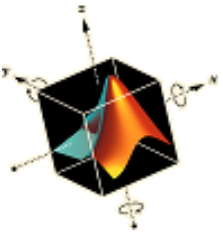
- The velocity of a rocket is given by

$$v(t) = 2000 \ln\left[\frac{14\times10^4}{14\times10^4 - 2100t}\right] - 9.8t, \quad 0 \le t \le 30$$

- Where 'v' is given in m/s and 't' is given in seconds.

a) Use central divided difference approximation of the first derivative of $v(t)$ to calculate the acceleration at $t = 16s$. Use a step size of $\Delta t = 2s$.

b) Calculate the absolute relative true error for part (a).



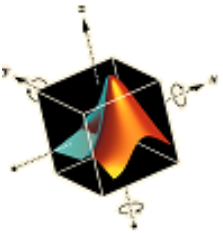
Aproximación por diferencia central - ejemplo

- Solution

a)
$$a(t_i) \approx \frac{v(t_{i+1}) - v(t_{i-1})}{2\Delta t} \qquad t_i = 16 \qquad \Delta t = 2$$
$$t_{i-1} = t_i - \Delta t = 16 - 2 = 14$$
$$t_{i+1} = t_i + \Delta t = 16 + 2 = 18$$
$$a(16) \approx \frac{v(18) - v(14)}{2 \cdot 2}$$
$$v(18) = 2000 \ln\left[\frac{14\times10^4}{14\times10^4 - 2100(18)}\right] - 9.8(18) = 453.02 \text{ m/s}$$
$$v(14) = 2000 \ln\left[\frac{14\times10^4}{14\times10^4 - 2100(14)}\right] - 9.8(14) = 334.24 \text{ m/s}$$

Hence
$$a(16) \approx \frac{v(18) - v(14)}{4} \approx \frac{453.02 - 334.24}{4} \approx 29.694 \text{ m/s}^2$$

b) The absolute relative true error is knowing that the exact value at $t = 16s$ is $a(16) = 29.674 \text{ m/s}^2$
$$|\epsilon_t| = \left|\frac{29.674 - 29.694}{29.674}\right| \cdot 100 = 0.069157 \ \%$$



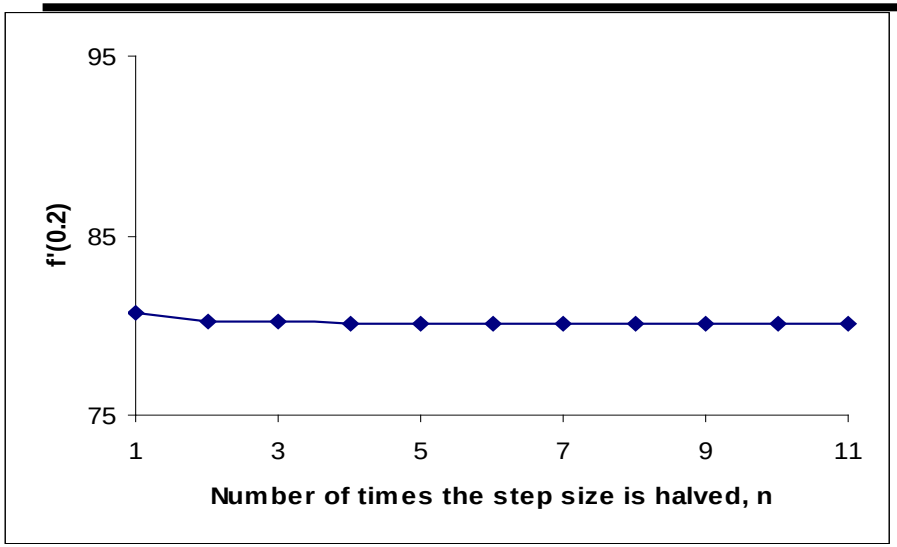
Efecto del tamaño de paso en el método de diferencia dividida central

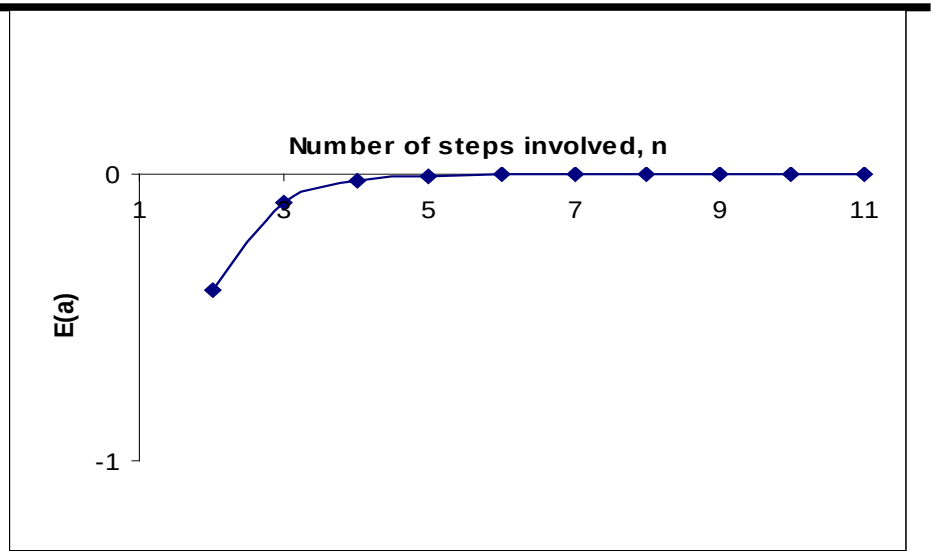
$$f(x) = 9e^{4x}$$

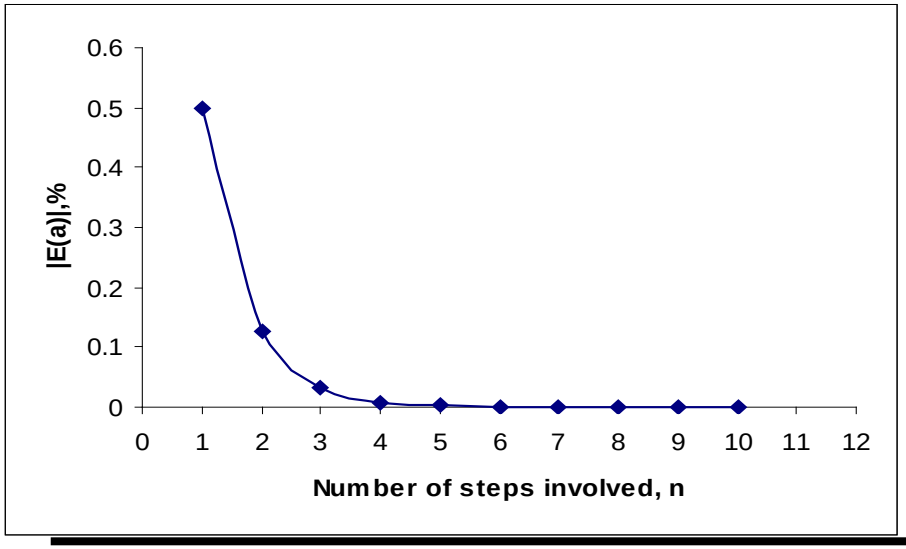
Value of $f'(0.2)$ using forward difference method.

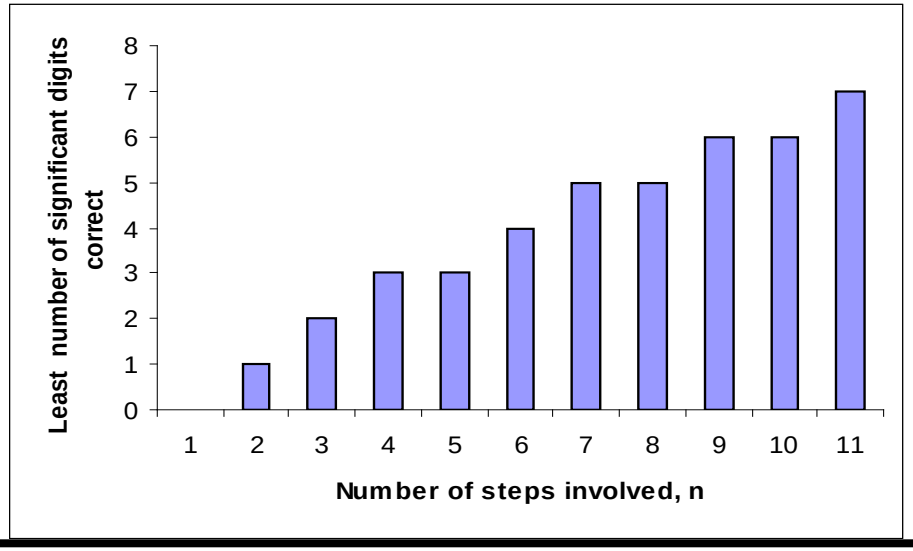
h	$f'(0.2)$	E_a	$\lvert\varepsilon_a\rvert\,\%$	Significant digits	E_t	$\lvert\varepsilon_t\rvert\,\%$
0.05	80.65467				-0.53520	0.668001
0.025	80.25307	-0.4016	0.500417	1	-0.13360	0.16675
0.0125	80.15286	-0.100212	0.125026	2	-0.03339	0.041672
0.00625	80.12782	-0.025041	0.031252	3	-0.00835	0.010417
0.003125	80.12156	-0.00626	0.007813	3	-0.00209	0.002604
0.001563	80.12000	-0.001565	0.001953	4	-0.00052	0.000651
0.000781	80.11960	-0.000391	0.000488	5	-0.00013	0.000163
0.000391	80.11951	-9.78E-05	0.000122	5	-0.00003	4.07E-05
0.000195	80.11948	-2.45E-05	3.05E-05	6	-0.00001	1.02E-05
9.77E-05	80.11948	-6.11E-06	7.63E-06	6	0.00000	2.54E-06
4.88E-05	80.11947	-1.53E-06	1.91E-06	7	0.00000	6.36E-07

Efecto del tamaño de paso en el método de diferencia dividida central

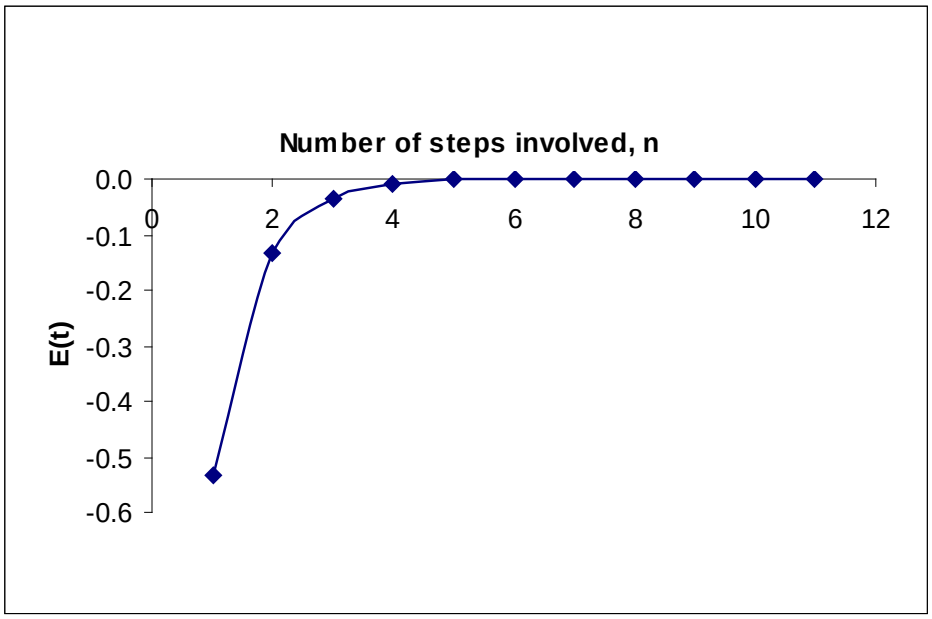


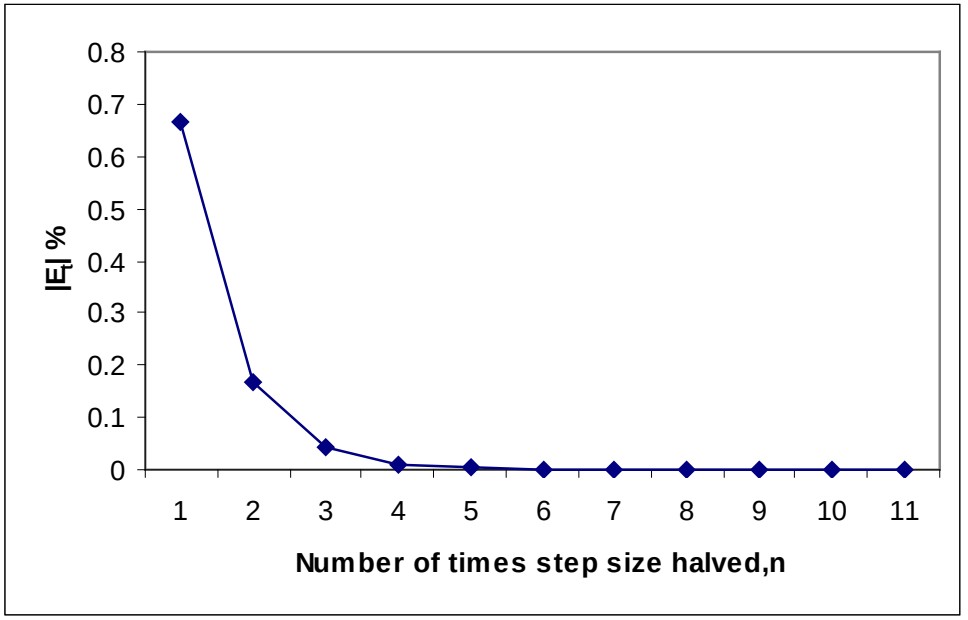


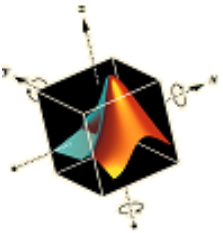




Efecto del tamaño de paso en el método de diferencia dividida central





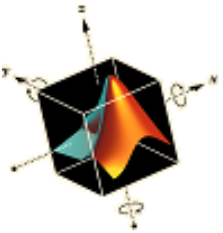


Comparación de métodos adad, adat, adc

- The results from the three difference approximations are given in the following table.

Summary of *a* (16) using different divided difference approximations

Type of Difference Approximation	$a(16)$ (m/s^2)	$\lvert\epsilon_t\rvert\%$
Forward	30.475	2.6967
Backward	28.915	2.5584
Central	29.695	0.069157

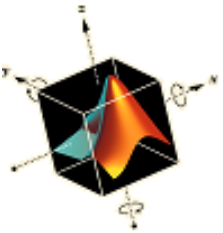


Obtención del valor de la derivada con una tolerancia específica

- In real life, one would not know the exact value of the derivative – so how would one know how accurately they have found the value of the derivative.
- A simple way would be to start with a step size and keep on halving the step size until the absolute relative approximate error is within a pre-specified tolerance.
- Take the example of finding $v'(t)$ for

$$v(t) = 2000 \ln\left[\frac{14\times10^4}{14\times10^4 - 2100t}\right] - 9.8t$$

at $t = 16$ using the backward divided difference scheme.

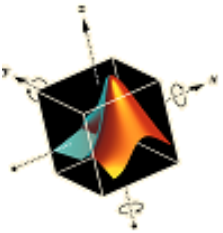


Obtención del valor de la derivada con una tolerancia específica

- The values obtained using the backward difference approximation method and the corresponding absolute relative approximate errors are given in the following table.

Δt	$v'(t)$	$\lvert\in_a\rvert\%$
2	28.915	
1	29.289	1.2792
0.5	29.480	0.64787
0.25	29.577	0.32604
0.125	29.625	0.16355

one can see that the absolute relative approximate error decreases as the step size is reduced. At $\Delta t = 0.125$ the absolute relative approximate error is 0.16355%, meaning that at least 2 significant digits are correct in the answer.



Aproximación por diferencia finita de derivadas de mayor orden

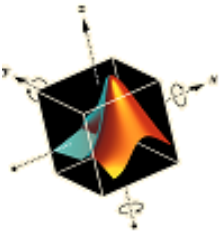
- One can use Taylor series to approximate a higher order derivative.
- For example, to approximate $f''(x)$, the Taylor series for

$$f(x_{i+2}) = f(x_i) + f'(x_i)(2\Delta x) + \frac{f''(x_i)}{2!}(2\Delta x)^2 + \frac{f'''(x_i)}{3!}(2\Delta x)^3 + \ldots \qquad (3)$$

where $x_{i+2} = x_i + 2\Delta x$

$$f(x_{i+1}) = f(x_i) + f'(x_i)(\Delta x) + \frac{f''(x_i)}{2!}(\Delta x)^2 + \frac{f'''(x_i)}{3!}(\Delta x)^3 \ldots \qquad (4)$$

where $x_{i-1} = x_i - \Delta x$



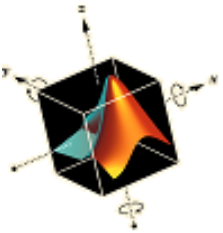
Aproximación por diferencia finita de derivadas de mayor orden

- Subtracting 2 times equation (4) from equation (3) gives

$$f(x_{i+2}) - 2f(x_{i+1}) = -f(x_i) + f''(x_i)(\Delta x)^2 + f'''(x_i)(\Delta x)^3 \ldots$$

$$f''(x_i) = \frac{f(x_{i+2}) - 2f(x_{i+1}) + f(x_i)}{(\Delta x)^2} - f'''(x_i)(\Delta x) + \ldots$$

$$f''(x_i) \cong \frac{f(x_{i+2}) - 2f(x_{i+1}) + f(x_i)}{(\Delta x)^2} + O(\Delta x) \qquad (5)$$



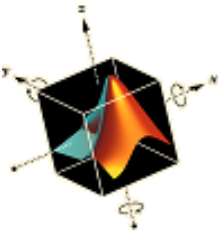
Aproximación por diferencia finita de segunda derivada - ejemplo

- The velocity of a rocket is given by

$$v(t) = 2000 \ln\left[\frac{14\times10^4}{14\times10^4 - 2100t}\right] - 9.8t, \quad 0 \le t \le 30$$

- Where $'v'$ is given in m/s and $'t'$ is given in seconds.

Use forward difference approximation of the second derivative of $v(t)$ to calculate the jerk at $t = 16s$. Use a step size of $\Delta t = 2s$.



Aproximación por diferencia finita de segunda derivada - ejemplo

- Solution

$$j(t_i) \approx \frac{v(t_{i+2}) - 2v(t_{i+1}) + v(t_i)}{(\Delta t)^2}$$

$$t_i = 16 \qquad \Delta t = 2$$

$$t_{i+1} = t_i + \Delta t = 16 + 2 = 18$$

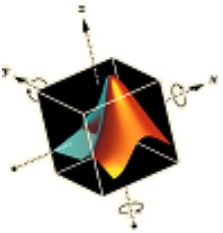
$$t_{i+2} = t_i + 2(\Delta t) = 16 + 2(2) = 20$$

$$j(16) \approx \frac{v(20) - 2v(18) + v(16)}{(2)^2}$$

$$v(20) = 2000 \ln\left[\frac{14 \times 10^4}{14 \times 10^4 - 2100(20)}\right] - 9.8(20) = 517.35 \text{m/s}$$

$$v(18) = 2000 \ln\left[\frac{14 \times 10^4}{14 \times 10^4 - 2100(18)}\right] - 9.8(18) = 453.02 m/s$$

$$v(16) = 2000 \ln\left[\frac{14 \times 10^4}{14 \times 10^4 - 2100(16)}\right] - 9.8(16) = 392.07 \text{m/s}$$



Aproximación por diferencia en adelanto - ejemplo

$$j(16) \approx \frac{517.35 - 2(453.02) + 392.07}{4} \approx 0.84515 \text{m/s}^3$$

The exact value of $j(16)$ can be calculated by differentiating

$v(t) = 2000 \ln\left[\frac{14\times10^4}{14\times10^4 - 2100t}\right] - 9.8t$ twice as $a(t) = \frac{d}{dt}[v(t)]$ and $j(t) = \frac{d}{dt}[a(t)]$

Knowing that $\frac{d}{dt}[\ln(t)] = \frac{1}{t}$ and $\frac{d}{dt}\left[\frac{1}{t}\right] = -\frac{1}{t^2}$

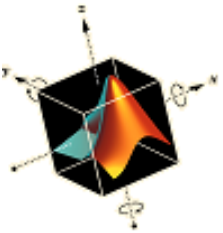
$$a(t) = 2000\left(\frac{14\times10^4 - 2100t}{14\times10^4}\right)\frac{d}{dt}\left(\frac{14\times10^4}{14\times10^4 - 2100t}\right) - 9.8$$

$$= 2000\left(\frac{14\times10^4 - 2100t}{14\times10^4}\right)(-1)\left(\frac{14\times10^4}{(14\times10^4 - 2100t)^2}\right)(-2100) - 9.8 = \frac{-4040 - 29.4t}{-200 + 3t}$$

Similarly it can be shown that

$$j(t) = \frac{d}{dt}[a(t)] = \frac{18000}{(-200 + 3t)^2} \qquad j(16) = \frac{18000}{[-200 + 3(16)]^2} = 0.77909 \text{ m/s}^3$$

The absolute relative true error is: $|\epsilon_t| = \left|\frac{0.77909 - 0.84515}{0.77909}\right| \cdot 100 = 8.4797\ \%$



Aproximación por diferencia en adelanto - ejemplo

Knowing that

$$\frac{d}{dt}\left[\ln(t)\right]=\frac{1}{t} \quad \text{and} \quad \frac{d}{dt}\left[\frac{1}{t}\right]=-\frac{1}{t^2}$$

$$a(t)=2000\left(\frac{14\times10^4-2100t}{14\times10^4}\right)\frac{d}{dt}\left(\frac{14\times10^4}{14\times10^4-2100t}\right)-9.8$$

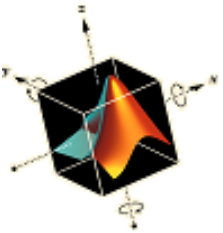
$$=2000\left(\frac{14\times10^4-2100t}{14\times10^4}\right)(-1)\left(\frac{14\times10^4}{\left(14\times10^4-2100t\right)^2}\right)(-2100)-9.8$$

$$=\frac{-4040-29.4t}{-200+3t}$$

$$a(16)=\frac{-4040-29.4(16)}{-200+3(16)}=29.674\text{m/s}^2$$

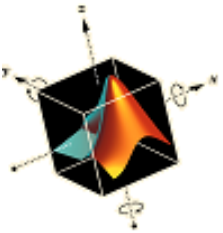
c) The absolute relative true error is

$$|\epsilon_t|=\left|\frac{\text{True Value - Approximate Value}}{\text{True Value}}\right|\cdot100=\left|\frac{29.674-30.474}{29.674}\right|x100=2.6967\ \%$$



Exactitud de derivadas de mayor orden

- The formula given by equation (5) is a forward difference approximation of the second derivative and has the error of the order of (Δx).
- Can we get a formula that has a better accuracy? We can get the central difference approximation of the second derivative.



Exactitud de derivadas de mayor orden

- The Taylor series for

$$f(x_{i+1}) = f(x_i) + f'(x_i)\Delta x + \frac{f''(x_i)}{2!}(\Delta x)^2 + \frac{f'''(x_i)}{3!}(\Delta x)^3 + \frac{f''''(x_i)}{4!}(\Delta x)^4 \ldots \quad (6)$$

where $x_{i+1} = x_i + \Delta x$

$$f(x_{i-1}) = f(x_i) - f'(x_i)\Delta x + \frac{f''(x_i)}{2!}(\Delta x)^2 - \frac{f'''(x_i)}{3!}(\Delta x)^3 + \frac{f''''(x_i)}{4!}(\Delta x)^4 \ldots \quad (7)$$

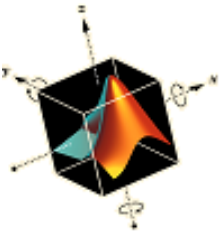
where $x_{i-1} = x_i - \Delta x$

Adding equations (6) and (7), gives

$$f(x_{i+1}) + f(x_{i-1}) = 2f(x_i) + f''(x_i)(\Delta x)^2 + f''''(x_i)\frac{(\Delta x)^4}{12}$$

$$f''(x_i) = \frac{f(x_{i+1}) - 2f(x_i) + f(x_{i-1})}{(\Delta x)^2} - \frac{f''''(x_i)(\Delta x)^2}{12}$$

$$f''(x_i) = \frac{f(x_{i+1}) - 2f(x_i) + f(x_{i-1})}{(\Delta x)^2} + 0(\Delta x)^2$$



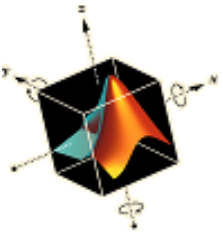
Aproximación por diferencia finita central de segunda derivada - ejemplo

- The velocity of a rocket is given by

$$v(t) = 2000 \ln\left[\frac{14\times10^4}{14\times10^4 - 2100t}\right] - 9.8t, \quad 0 \le t \le 30$$

- Where 'v' is given in m/s and 't' is given in seconds.

Use central difference approximation of the second derivative of $v(t)$ to calculate the jerk at $t = 16s$. Use a step size of $\Delta t = 2s$.



Aproximación por diferencia finita de segunda derivada - ejemplo

- Solution

$$j(t_i) \approx \frac{v(t_{i+1}) - 2v(t_i) + v(t_{i-1})}{(\Delta t)^2}$$

$t_i = 16 \qquad \Delta t = 2$

$t_{i+1} = t_i + \Delta t = 16 + 2 = 18$

$t_{i-1} = t_i - \Delta t = 16 - 2 = 14$

$$j(16) \approx \frac{v(18) - 2v(16) + v(14)}{(2)^2}$$

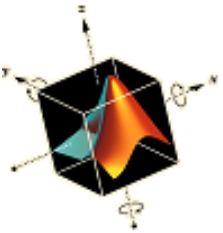
$$v(18) = 2000 \ln\left[\frac{14 \times 10^4}{14 \times 10^4 - 2100(18)}\right] - 9.8(18) = 453.02 \text{ m/s}$$

$$v(16) = 2000 \ln\left[\frac{14 \times 10^4}{14 \times 10^4 - 2100(16)}\right] - 9.8(16) = 392.07 \text{ m/s}$$

$$v(14) = 2000 \ln\left[\frac{14 \times 10^4}{14 \times 10^4 - 2100(14)}\right] - 9.8(14) = 334.24 \text{ m/s}$$

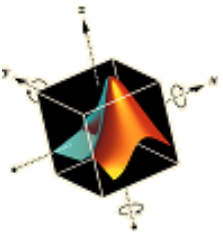
$$j(16) \approx \frac{v(18) - 2v(16) + v(14)}{(2)^2} \approx \frac{453.02 - 2(392.07) + 334.24}{4} \approx 0.77969 \text{m/s}^3$$

The absolute relative true error is $|\epsilon_t| = \left|\frac{0.77908 - 0.78}{0.77908}\right| \times 100 = 0.077992\%$



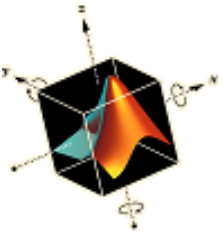
Fórmulas de diferencia finita en adelanto

	Error
First Derivative	
$f'(x_i) = \frac{f(x_{i+1}) - f(x_i)}{h}$	$O(h)$
$f'(x_i) = \frac{-f(x_{i+2}) + 4f(x_{i+1}) - 3f(x_i)}{2h}$	$O(h^2)$
Second Derivative	
$f''(x_i) = \frac{f(x_{i+2}) - 2f(x_{i+1}) + f(x_i)}{h^2}$	$O(h)$
$f''(x_i) = \frac{-f(x_{i+3}) + 4f(x_{i+2}) - 5f(x_{i+1}) + 2f(x_i)}{h^2}$	$O(h^2)$
Third Derivative	
$f'''(x_i) = \frac{f(x_{i+3}) - 3f(x_{i+2}) + 3f(x_{i+1}) - f(x_i)}{h^3}$	$O(h)$
$f'''(x_i) = \frac{-3f(x_{i+4}) + 14f(x_{i+3}) - 24f(x_{i+2}) + 18f(x_{i+1}) - 5f(x_i)}{2h^3}$	$O(h^2)$
Fourth Derivative	
$f''''(x_i) = \frac{f(x_{i+4}) - 4f(x_{i+3}) + 6f(x_{i+2}) - 4f(x_{i+1}) + f(x_i)}{h^4}$	$O(h)$
$f''''(x_i) = \frac{-2f(x_{i+5}) + 11f(x_{i+4}) - 24f(x_{i+3}) + 26f(x_{i+2}) - 14f(x_{i+1}) + 3f(x_i)}{h^4}$	$O(h^2)$



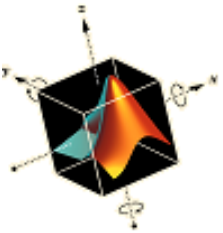
Fórmulas de diferencia finita en atraso

	Error
First Derivative	
$f'(x_i) = \frac{f(x_i) - f(x_{i-1})}{h}$	$O(h)$
$f'(x_i) = \frac{3f(x_i) - 4f(x_{i-1}) + f(x_{i-2})}{2h}$	$O(h^2)$
Second Derivative	
$f''(x_i) = \frac{f(x_i) - 2f(x_{i-1}) + f(x_{i-2})}{h^2}$	$O(h)$
$f''(x_i) = \frac{2f(x_i) - 5f(x_{i-1}) + 4f(x_{i-2}) - f(x_{i-3})}{h^2}$	$O(h^2)$
Third Derivative	
$f'''(x_i) = \frac{f(x_i) - 3f(x_{i-1}) + 3f(x_{i-2}) - f(x_{i-3})}{h^3}$	$O(h)$
$f'''(x_i) = \frac{5f(x_i) - 18f(x_{i-1}) + 24f(x_{i-2}) - 14f(x_{i-3}) + 3f(x_{i-4})}{2h^3}$	$O(h^2)$
Fourth Derivative	
$f''''(x_i) = \frac{f(x_i) - 4f(x_{i-1}) + 6f(x_{i-2}) - 4f(x_{i-3}) + f(x_{i-4})}{h^4}$	$O(h)$
$f''''(x_i) = \frac{3f(x_i) - 14f(x_{i-1}) + 26f(x_{i-2}) - 24f(x_{i-3}) + 11f(x_{i-4}) - 2f(x_{i-5})}{h^4}$	$O(h^2)$



Fórmulas de diferencia finita centrada

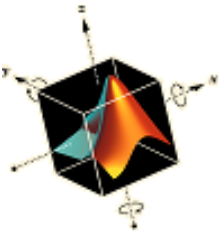
	Error
First Derivative	
$f'(x_i) = \dfrac{f(x_{i+1}) - f(x_{i-1})}{2h}$	$O(h^2)$
$f'(x_i) = \dfrac{-f(x_{i+2}) + 8f(x_{i+1}) - 8f(x_{i-1}) + f(x_{i-2})}{12h}$	$O(h^4)$
Second Derivative	
$f''(x_i) = \dfrac{f(x_{i+1}) - 2f(x_i) + f(x_{i-1})}{h^2}$	$O(h^2)$
$f''(x_i) = \dfrac{-f(x_{i+2}) + 16f(x_{i+1}) - 30f(x_i) + 16f(x_{i-1}) - f(x_{i-2})}{12h^2}$	$O(h^4)$
Third Derivative	
$f'''(x_i) = \dfrac{f(x_{i+2}) - 2f(x_{i+1}) + 2f(x_{i-1}) - f(x_{i-2})}{2h^3}$	$O(h^2)$
$f'''(x_i) = \dfrac{-f(x_{i+3}) + 8f(x_{i+2}) - 13f(x_{i+1}) + 13f(x_{i-1}) - 8f(x_{i-2}) + f(x_{i-3})}{8h^3}$	$O(h^4)$
Fourth Derivative	
$f''''(x_i) = \dfrac{f(x_{i+2}) - 4f(x_{i+1}) + 6f(x_i) - 4f(x_{i-1}) + f(x_{i-2})}{h^4}$	$O(h^2)$
$f''''(x_i) = \dfrac{-f(x_{i+3}) + 12f(x_{i+2}) + 39f(x_{i+1}) + 56f(x_i) - 39f(x_{i-1}) + 12f(x_{i-2}) + f(x_{i-3})}{6h^4}$	$O(h^4)$



Derivadas de funciones discretas (datos espaciados no uniforme)

- One way to calculated derivatives of unequally spaced data is to determine a polynomial fit and take its derivative at a point.
- As an example, using a second-order Lagrange polynomial to fit three points and taking its derivative yields:

$$f'(x) = f(x_0)\frac{2x - x_1 - x_2}{(x_0 - x_1)(x_0 - x_2)} + f(x_1)\frac{2x - x_0 - x_2}{(x_1 - x_0)(x_1 - x_2)} + f(x_2)\frac{2x - x_0 - x_1}{(x_2 - x_0)(x_2 - x_1)}$$



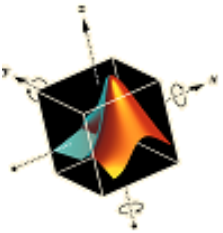
Derivadas de funciones discretas - ejemplo

- The upward velocity of a rocket is given as a function of time in the following table.

Velocity as a function of time

t (s)	**v(t)** (m/s)
0	0
10	227.04
15	362.78
20	517.35
22.5	602.97
30	901.67

Using forward divided difference, find the acceleration of the rocket at $t = 16$ s.



Derivadas de funciones discretas - ejemplo

- Solution

To find the acceleration at $t = 16\text{s}$, we need to choose the two values closest to $t = 16\text{s}$, that also bracket $t = 16\text{s}$ to evaluate it. The two points are $t = 15\text{s}$ and $t = 20\text{s}$.

$$a(t_i) \approx \frac{v(t_{i+1}) - v(t_i)}{\Delta t}$$

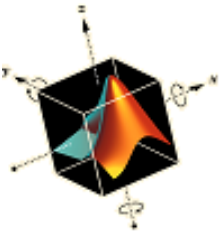
$$t_i = 15$$

$$t_{i+1} = 20$$

$$\Delta t = t_{i+1} - t_i = 20 - 15 = 5$$

$$a(16) \approx \frac{v(20) - v(15)}{5} \approx \frac{517.35 - 362.78}{5}$$

$$\approx 30.914\ \text{m/s}^2$$



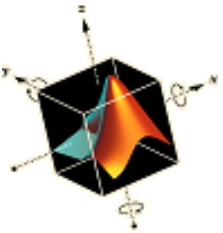
Derivadas de funciones discretas – ejemplo ajuste polinómico

- The upward velocity of a rocket is given as a function of time in the following table.

Velocity as a function of time

t (s)	**v(t)** (m/s)
0	0
10	227.04
15	362.78
20	517.35
22.5	602.97
30	901.67

Using the third order polynomial interpolant for velocity, find the acceleration of the rocket at $t = 16$ s.



Derivadas de funciones discretas – ejemplo ajuste polinómico

- Solution

In this method, given $n+1$ data points

$$(x_0, y_0), (x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$$

one can fit a n^{th} order polynomial given by

$$P_n(x) = a_0 + a_1 x + \ldots\ldots + a_{n-1} x^{n-1} + a_n x^n$$

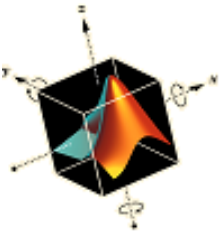
To find the first derivative,

$$P_n'(x) = \frac{dP_n(x)}{dx} = a_1 + 2a_2 x + \ldots\ldots + (n-1) a_{n-1} x^{n-2} + n a_n x^{n-1}$$

Similarly other derivatives can be found.

For the third order polynomial (also called cubic interpolation), we choose the velocity given by

$$v(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3$$



Derivadas de funciones discretas – ejemplo ajuste polinómico

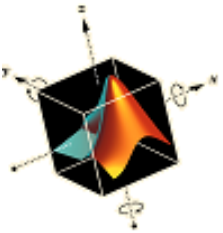
Since we want to find the velocity at $t = 16$ s , and we are using third order polynomial, we need to choose the four points closest to $t = 16$ s and that also bracket $t = 16$ s to evaluate it.

The four points are $t_o = 10, t_1 = 15, t_2 = 20,$ and $t_3 = 22.5.$

$$\left.\begin{array}{l} t_o = 10,\ v(t_o) = 227.04 \\ t_1 = 15,\ v(t_1) = 362.78 \\ t_2 = 20,\ v(t_2) = 517.35 \\ t_3 = 22.5,\ v(t_3) = 602.97 \end{array}\right\} \begin{array}{l} v(10) = 227.04 = a_0 + a_1(10) + a_2(10)^2 + a_3(10)^3 \\ v(15) = 362.78 = a_0 + a_1(15) + a_2(15)^2 + a_3(15)^3 \\ v(20) = 517.35 = a_0 + a_1(20) + a_2(20)^2 + a_3(20)^3 \\ v(22.5) = 602.97 = a_0 + a_1(22.5) + a_2(22.5)^2 + a_3(22.5)^3 \end{array}$$

Writing the four equations in matrix form, we have

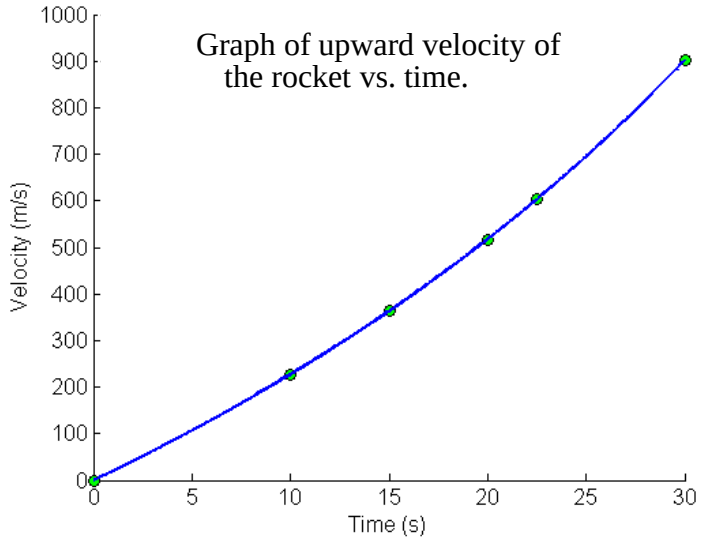
$$\begin{bmatrix} 1 & 10 & 100 & 1000 \\ 1 & 15 & 225 & 3375 \\ 1 & 20 & 400 & 8000 \\ 1 & 22.5 & 506.25 & 11391 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 227.04 \\ 362.78 \\ 517.35 \\ 602.97 \end{bmatrix}$$



Derivadas de funciones discretas – ejemplo ajuste polinómico

Solving the above four equations gives

$$a_0 = -4.3810$$
$$a_1 = 21.289$$
$$a_2 = 0.13065$$
$$a_3 = 0.0054606$$



Hence

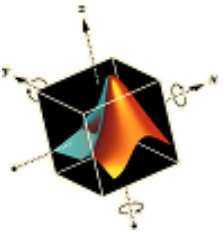
$$v(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3$$
$$= -4.3810 + 21.289t + 0.13065t^2 + 0.0054606t^3, \quad 10 \le t \le 22.5$$

The acceleration at t=16 is given by $a(16) = \frac{d}{dt} v(t)\Big|_{t=16}$

Given that $a(t) = \frac{d}{dt} v(t) = \frac{d}{dt}\left(-4.3810 + 21.289t + 0.13065t^2 + 0.0054606t^3\right)$

$$= 21.289 + 0.26130t + 0.016382t^2, \quad 10 \le t \le 22.5$$

$$a(16) = 21.289 + 0.26130(16) + 0.016382(16)^2 = 29.664 \text{m/s}^2$$



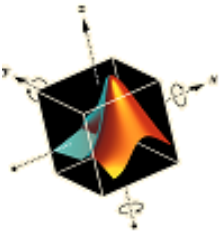
Derivadas de funciones discretas – ejemplo ajuste polinomio Lagrange

- The upward velocity of a rocket is given as a function of time in the following table.

Velocity as a function of time

t (s)	**v(t)** (m/s)
0	0
10	227.04
15	362.78
20	517.35
22.5	602.97
30	901.67

Using the 2nd order Lagrangian polynomial interpolant for velocity, find the acceleration of the rocket at $t = 16$ s.



Derivadas de funciones discretas – ejemplo ajuste polinomio Lagrange

- Solution

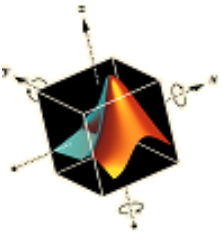
In this method, given $n+1$ data points $(x_0, y_0), (x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ one can fit a $(n-1)^{th}$ order Lagrangian polynomial given by

$$f_n(x) = \sum_{i=0}^{n} L_i(x)\, f(x_i)$$

where n in $f_n(x)$ stands for the n^{th} order polynomial that approximates the function $y = f(x)$ given at data points and

$L_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j}$ a weighting function

Then to find the first derivative, one can differentiate $f_n(x)$ once, and so on for other derivatives.



Derivadas de funciones discretas - ejemplo

The second order Lagrange polynomial passing through $(x_0, y_0), (x_1, y_1), (x_2, y_2)$ is

$$f_2(x) = \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)} f(x_0) + \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)} f(x_1) + \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)} f(x_2)$$

Differentiating this equation gives

$$f_2'(x) = \frac{2x-(x_1+x_2)}{(x_0-x_1)(x_0-x_2)} f(x_0) + \frac{2x-(x_0+x_2)}{(x_1-x_0)(x_1-x_2)} f(x_1) + \frac{2x-(x_0+x_1)}{(x_2-x_0)(x_2-x_1)} f(x_2)$$

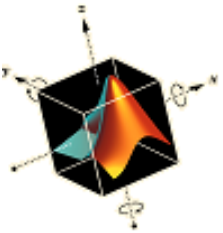
Differentiating again would give the second derivative as

$$f_2''(x) = \frac{2}{(x_0-x_1)(x_0-x_2)} f(x_0) + \frac{2}{(x_1-x_0)(x_1-x_2)} f(x_1) + \frac{2}{(x_2-x_0)(x_2-x_1)} f(x_2)$$

From the data

$$v(t) = \left(\frac{t-t_1}{t_0-t_1}\right)\left(\frac{t-t_2}{t_0-t_2}\right) v(t_0) + \left(\frac{t-t_0}{t_1-t_0}\right)\left(\frac{t-t_2}{t_1-t_2}\right) v(t_1) + \left(\frac{t-t_0}{t_2-t_0}\right)\left(\frac{t-t_1}{t_2-t_1}\right) v(t_2)$$

$$a(t) = \frac{2t-(t_1+t_2)}{(t_0-t_1)(t_0-t_2)} v(t_0) + \frac{2t-(t_0+t_2)}{(t_1-t_0)(t_1-t_2)} v(t_1) + \frac{2t-(t_0+t_1)}{(t_2-t_0)(t_2-t_1)} v(t_2)$$



Derivadas de funciones discretas - ejemplo

Replacing

$$a(16)=\frac{2(16)-(15+20)}{(10-15)(10-20)}(227.04)+\frac{2(16)-(10+20)}{(15-10)(15-20)}(362.78)+\frac{2(16)-(10+15)}{(20-10)(20-15)}(517.35)$$

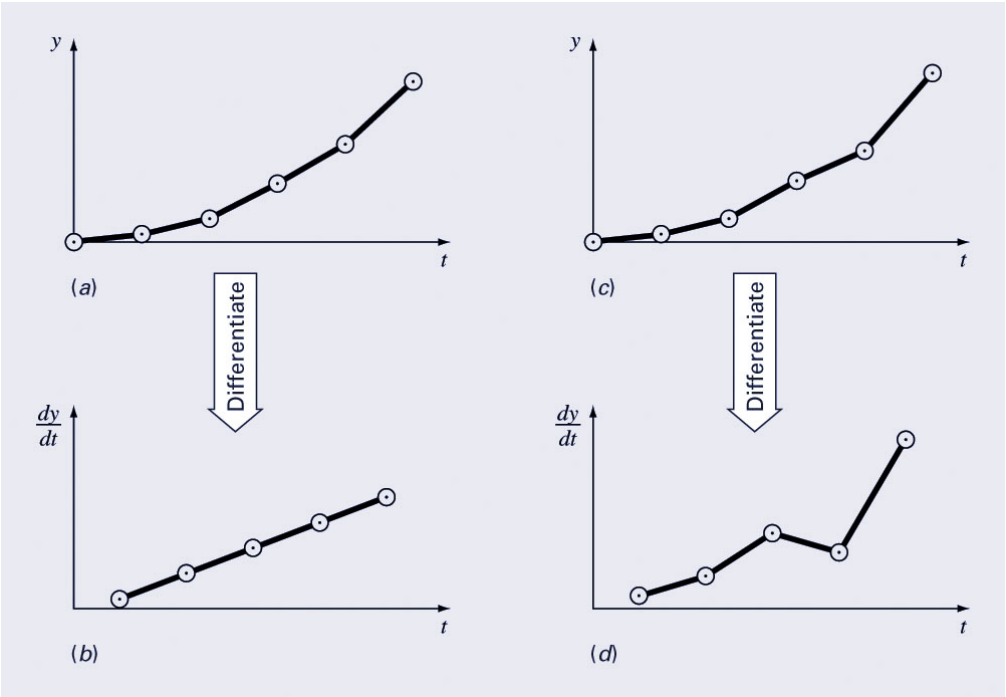
$$=-0.06(227.04)-0.08(362.78)+0.14(517.35)=29.784\text{m/s}^2$$

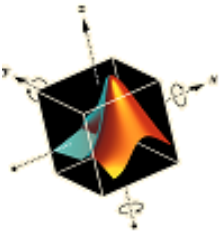
Summary of *a* (16) using different polynomic approximations

Type of polynomial	$a(16)$ (m/s^2)
Forward	30.914
Cubic	29.664
Lagrange	29.784

Derivadas e Integrales para datos con errores

- A shortcoming of numerical differentiation is that it tends to amplify errors in data, whereas integration tends to smooth data errors.
- One approach for taking derivatives of data with errors is to fit a smooth, differentiable function to the data and take the derivative of the function.





Extrapolación de Richardson

- As with integration, the Richardson extrapolation can be used to combine two lower-accuracy estimates of the derivative to produce a higher-accuracy estimate.
- For the cases where there are two $O(h^2)$ estimates and the interval is halved ($h_2=h_1/2$), an improved $O(h^4)$ estimate may be formed using:

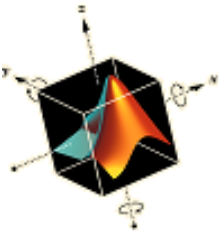
$$D = \frac{4}{3} D(h_2) - \frac{1}{3} D(h_1)$$

- For the cases where there are two $O(h^4)$ estimates and the interval is halved ($h_2=h_1/2$), an improved $O(h^6)$ estimate may be formed using:

$$D = \frac{16}{15} D(h_2) - \frac{1}{15} D(h_1)$$

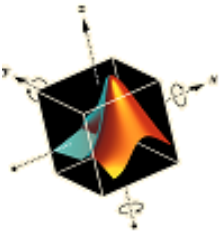
- For the cases where there are two $O(h^6)$ estimates and the interval is halved ($h_2=h_1/2$), an improved $O(h^8)$ estimate may be formed using:

$$D = \frac{64}{63} D(h_2) - \frac{1}{63} D(h_1)$$



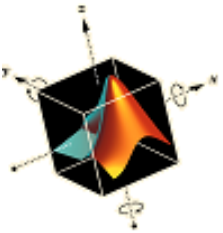
Diferenciación numérica con Matlab

- MATLAB has two built-in functions to help take derivatives, `diff` and `gradient`:
- `diff(x)`
  - Returns the difference between adjacent elements in `x`
- `diff(y)./diff(x)`
  - Returns the difference between adjacent values in `y` divided by the corresponding difference in adjacent values of `x`



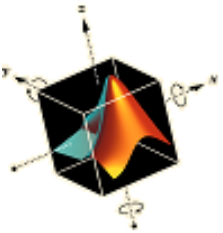
Diferenciación numérica con Matlab

- `fx = gradient(f, h)`
  Determines the derivative of the data in `f` at each of the points. The program uses forward difference for the first point, backward difference for the last point, and centered difference for the interior points. `h` is the spacing between points; if omitted `h=1`.
- The major advantage of `gradient` over `diff` is `gradient`'s result is the same size as the original data.
- Gradient can also be used to find partial derivatives for matrices: `[fx, fy] = gradient(f, h)`

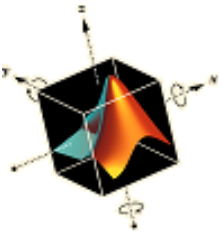


Diferenciación numérica con Matlab - Visualización

- MATLAB can generate contour plots of functions as well as vector fields. Assuming `x` and `y` represent a meshgrid of `x` and `y` values and `z` represents a function of `x` and `y`,
  - `contour(x, y, z)` can be used to generate a contour plot
  - `[fx, fy]=gradient(z,h)` can be used to generate partial derivatives and
  - `quiver(x, y, fx, fy)` can be used to generate vector fields

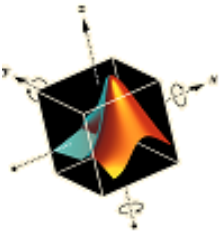


Interpolación



¿Qué es interpolación?

- Given (x_0,y_0), (x_1,y_1), …… (x_n,y_n), find the value of 'y' at a value of 'x' that is not given.
- Hence, the interpolation operation estimates intermediate values between precise data points.
- The function used to interpolate must pass through the actual data points interpolation more restrictive than fitting.
- The most common method for this purpose is *polynomial interpolation* (easy to evaluate, differentiate, and integrate).



Interpolación - Método directo

- Given ‘n+1’ data points (x_0,y_0), (x_1,y_1), …… (x_n,y_n), pass a polynomial of order ‘n’ through the data as

$$y = a_0 + a_1 x + \ldots\ldots\ldots\ldots\ldots + a_n x^n$$

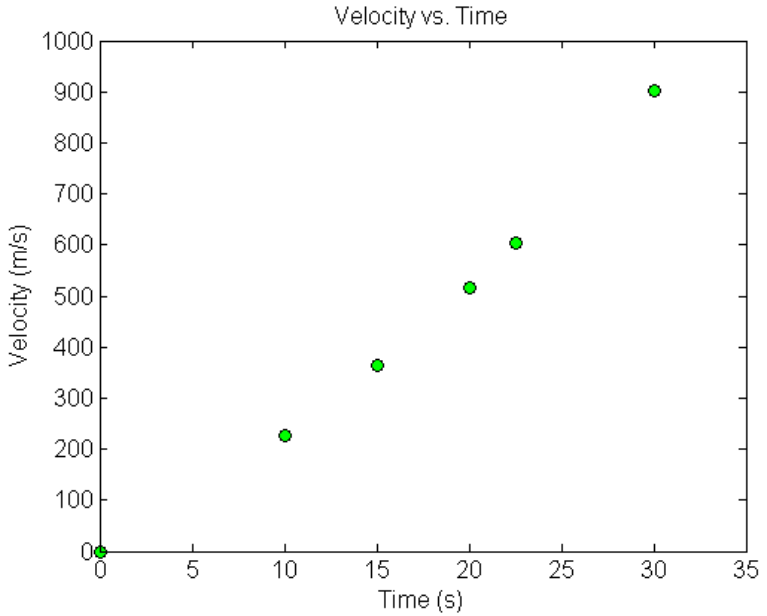
  where a_0, a_1,…… a_n are real constants.
- Set up ‘n+1’ equations to find ‘n+1’ constants. The polynomial coefficients can be found exactly using linear algebra.
- To find the value ‘y’ at a given value of ‘x’, simply substitute the value of ‘x’ in the above polynomial.
- Matlab commands: polyfit and polyval (order is n)

Interpolación - Método directo ejemplo

- The upward velocity of a rocket is given as a function of time in the following table.

Velocity as a function of time

$t, (\mathrm{s})$	$v(t), (\mathrm{m/s})$
0	0
10	227.04
15	362.78
20	517.35
22.5	602.97
30	901.67



Velocity vs. time data

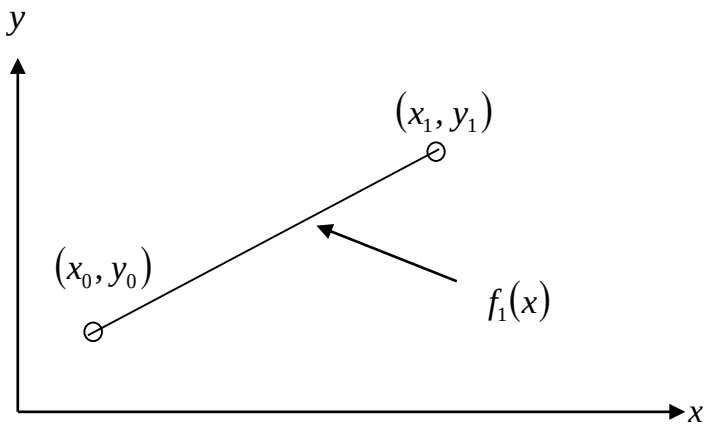
- Find the velocity at t=16 seconds using the direct method for linear, quadratic and cubic interpolation.

Interpolación - Método directo ejemplo

- Solution

Linear interpolation

$$v(t) = a_0 + a_1 t$$
$$v(15) = a_0 + a_1(15) = 362.78$$
$$v(20) = a_0 + a_1(20) = 517.35$$



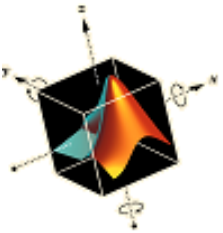
Linear interpolation.

Solving the above two equations gives,

$$a_0 = -100.93 \quad a_1 = 30.914$$

Hence

$$v(t) = -100.93 + 30.914t, \ 15 \le t \le 20.$$
$$v(16) = -100.93 + 30.914(16) = 393.7 \text{ m/s}$$



Interpolación - Método directo ejemplo

Quadratic interpolation $v(t) = a_0 + a_1 t + a_2 t^2$

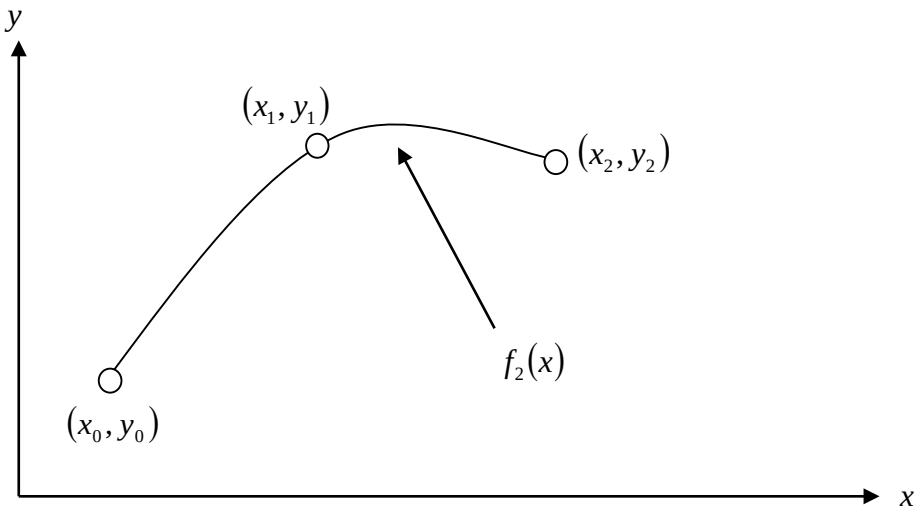
$$v(10) = a_0 + a_1(10) + a_2(10)^2 = 227.04$$

$$v(15) = a_0 + a_1(15) + a_2(15)^2 = 362.78$$

$$v(20) = a_0 + a_1(20) + a_2(20)^2 = 517.35$$

Solving the above equations

$$a_0 = 12.05 \quad a_1 = 17.733 \quad a_2 = 0.3766$$



Quadratic interpolation.

Hence

$$v(t) = 12.05 + 17.733t + 0.3766t^2, \quad 10 \le t \le 20$$

$$v(16) = 12.05 + 17.733(16) + 0.3766(16)^2 = 392.19 \text{ m/s}$$

The absolute relative approximate error $|\epsilon_a|$ obtained between the results from the first and second order polynomial is

$$|\epsilon_a| = \left| \frac{392.19 - 393.70}{392.19} \right| \cdot 100 = 0.38410\%$$

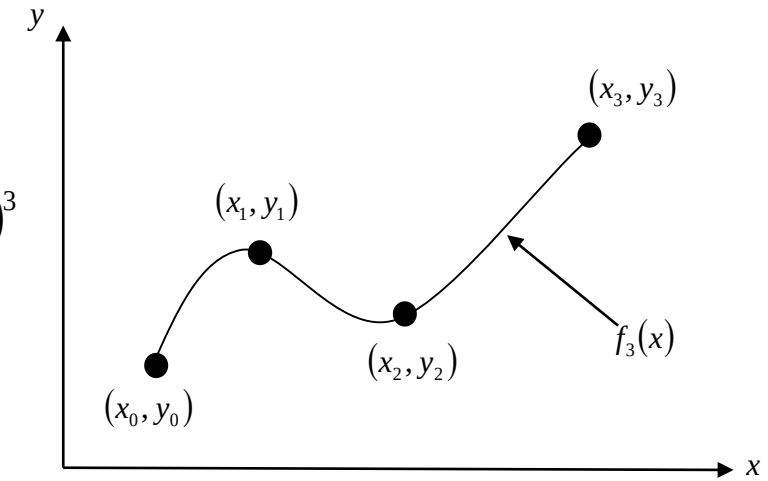
Interpolación - Método directo ejemplo

Cubic interpolation $v(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3$

$$v(10) = 227.04 = a_0 + a_1(10) + a_2(10)^2 + a_3(10)^3$$
$$v(15) = 362.78 = a_0 + a_1(15) + a_2(15)^2 + a_3(15)^3$$
$$v(20) = 517.35 = a_0 + a_1(20) + a_2(20)^2 + a_3(20)^3$$
$$v(22.5) = 602.97 = a_0 + a_1(22.5) + a_2(22.5)^2 + a_3(22.5)^3$$

Solving the above equations

$$a_0 = -4.2540 \quad a_1 = 21.266$$
$$a_2 = 0.13204 \quad a_3 = 0.0054347$$



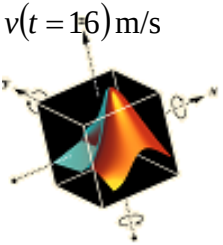
Cubic interpolation.

Hence

$$v(t) = -4.2540 + 21.266t + 0.13204t^2 + 0.0054347t^3, \quad 10 \le t \le 22.5$$
$$v(16) = -4.2540 + 21.266(16) + 0.13204(16)^2 + 0.0054347(16)^3 = 392.06 \text{ m/s}$$

The absolute relative approximate error $|\epsilon_a|$ obtained between the results from the second and third order polynomial is

$$|\epsilon_a| = \left| \frac{392.06 - 392.19}{392.06} \right| \times 100 = 0.033269\%$$



Interpolación - Métodos directos comparación

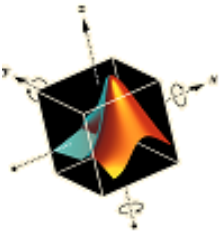
- Comparison table:

Velocity as a function of time

$t,(\mathrm{s})$	$v(t),(\mathrm{m/s})$
0	0
10	227.04
15	362.78
20	517.35
22.5	602.97
30	901.67

Comparison of different orders of the polynomial.

Order of Polynomial	1	2	3
$v(t=16)\,\mathrm{m/s}$	393.7	392.19	392.06
Absolute Relative Approximate Error	----------	0.38410 %	0.033269 %



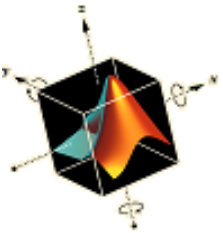
Interpolación - Método directo ejemplo

- The upward velocity of a rocket is given as a function of time in the following table.

- From the cubic interpolation method find the distance covered by the rocket from t=11s to t=16s, and the acceleration of the rocket at t=16s.

Velocity as a function of time

$t, (\mathrm{s})$	$v(t), (\mathrm{m/s})$
0	0
10	227.04
15	362.78
20	517.35
22.5	602.97
30	901.67



Interpolación - Método directo ejemplo

distance covered by the rocket from t=11s to t=16s

$$v(t) = -4.2540 + 21.266t + 0.13204^2 + 0.0054347t^3, 10 \le t \le 22.5$$

$$s(16) - s(11) = \int_{11}^{16} v(t)dt = \int_{11}^{16} \left(-4.2540 + 21.266t + 0.13204t^2 + 0.0054347t^3\right)dt$$

$$= \left[-4.2540t + 21.266\frac{t^2}{2} + 0.13204\frac{t^3}{3} + 0.0054347\frac{t^4}{4}\right]_{11}^{16} = 1605 \text{ m}$$

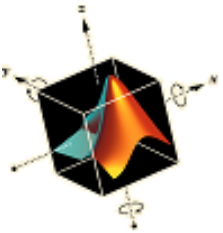
acceleration of the rocket at t=16s given that

$$v(t) = -4.2540 + 21.266t + 0.13204^2 + 0.0054347t^3, 10 \le t \le 22.5$$

$$a(t) = \frac{d}{dt}v(t) = \frac{d}{dt}\left(-4.2540 + 21.266t + 0.13204t^2 + 0.0054347t^3\right)$$

$$= 21.289 + 0.26130t + 0.016382t^2, \quad 10 \le t \le 22.5$$

$$a(16) = 21.266 + 0.26408(16) + 0.016304(16)^2 = 29.665 \text{ m/s}^2$$



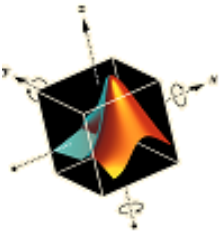
Interpolación polinómica - problemas

- One problem that can occur with solving for the coefficients of a polynomial is that the system to be inverted is in the form:

$$f(x) = a_0 + a_1 x + \ldots\ldots\ldots\ldots\ldots + a_n x^n, \qquad f(x_i) = y_i$$

$$\begin{bmatrix} x_0^n & x_0^{n-1} & \cdots & x_0 & 1 \\ x_1^n & x_1^{n-1} & \cdots & x_1 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_{n-1}^n & x_{n-1}^{n-1} & \cdots & x_{n-1} & 1 \\ x_n^n & x_n^{n-1} & \cdots & x_n & 1 \end{bmatrix} \begin{bmatrix} a_n \\ a_{n-1} \\ \vdots \\ a_1 \\ a_0 \end{bmatrix} = \begin{bmatrix} f(x_0) \\ f(x_1) \\ \vdots \\ f(x_{n-1}) \\ f(x_n) \end{bmatrix}$$

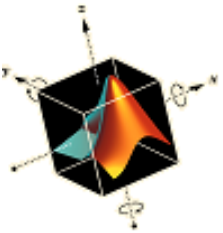
- Matrices such as that on the left are known as *Vandermonde* matrices, and they are very ill-conditioned - meaning their solutions are very sensitive to round-off errors.
- The issue can be minimized by scaling and shifting the data.



Polinomio interpolante de Newton

- Another way to express a polynomial interpolation is to use *Newton's interpolating polynomial*.
- The differences between a simple polynomial and Newton's interpolating polynomial for first and second order interpolations are:

Order	Simple	Newton
1*st*	$f_1(x) = a_0 + a_1 x$	$f_1(x) = b_0 + b_1(x - x_0)$
2*nd*	$f_2(x) = a_0 + a_1 x + a_2 x^2$	$f_2(x) = b_0 + b_1(x - x_0) + b_2(x - x_0)(x - x_1)$



Método de diferencia dividida de Newton

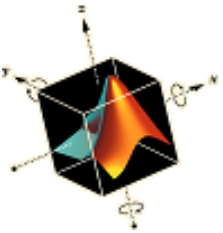
- Given $(x_0, y_0), \ldots (x_n, y_n)$
- Linear interpolation: $f_1(x) = b_0 + b_1(x - x_0)$

  where $b_0 = f(x_0)$ $\quad b_1 = \dfrac{f(x_1) - f(x_0)}{x_1 - x_0}$
- Quadratic interpolation: $f_2(x) = b_0 + b_1(x - x_0) + b_2(x - x_0)(x - x_1)$

  where $b_0 = f(x_0)$

$$b_1 = \frac{f(x_1) - f(x_0)}{x_1 - x_0}$$

$$b_2 = \frac{\dfrac{f(x_2) - f(x_1)}{x_2 - x_1} - \dfrac{f(x_1) - f(x_0)}{x_1 - x_0}}{x_2 - x_0}$$



Método de diferencia dividida de Newton

- Given $(x_0, y_0), \ldots (x_n, y_n)$
- Forma general:

$$f_n(x) = b_0 + b_1(x - x_0) + ... + b_n(x - x_0)(x - x_1)...(x - x_{n-1})$$

where

$$b_0 = f[x_0]$$

$$b_1 = f[x_1, x_0]$$

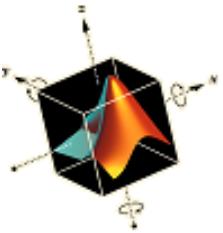
$$b_2 = f[x_2, x_1, x_0]$$

$$\vdots$$

$$b_{n-1} = f[x_{n-1}, x_{n-2}, ..., x_0]$$

$$b_n = f[x_n, x_{n-1}, ..., x_0]$$

and the f[…] represent divided differences.



Diferencias divididas

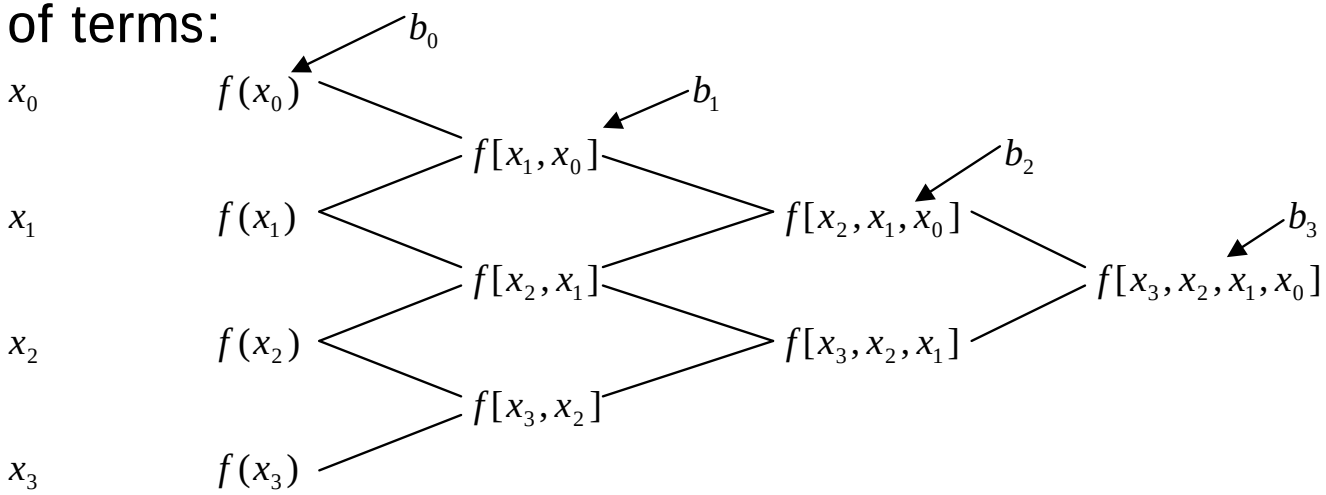
- Divided difference are calculated as follows:

$$f[x_i, x_j] = \frac{f(x_i) - f(x_j)}{x_i - x_j}$$

$$f[x_i, x_j, x_k] = \frac{f[x_i, x_j] - f[x_j, x_k]}{x_i - x_k}$$

$$f[x_n, x_{n-1}, \cdots, x_2, x_1] = \frac{f[x_n, x_{n-1}, \cdots, x_2] - f[x_{n-1}, x_{n-2}, \cdots, x_1]}{x_n - x_1}$$

- Divided differences are calculated using divided difference of a smaller number of terms:

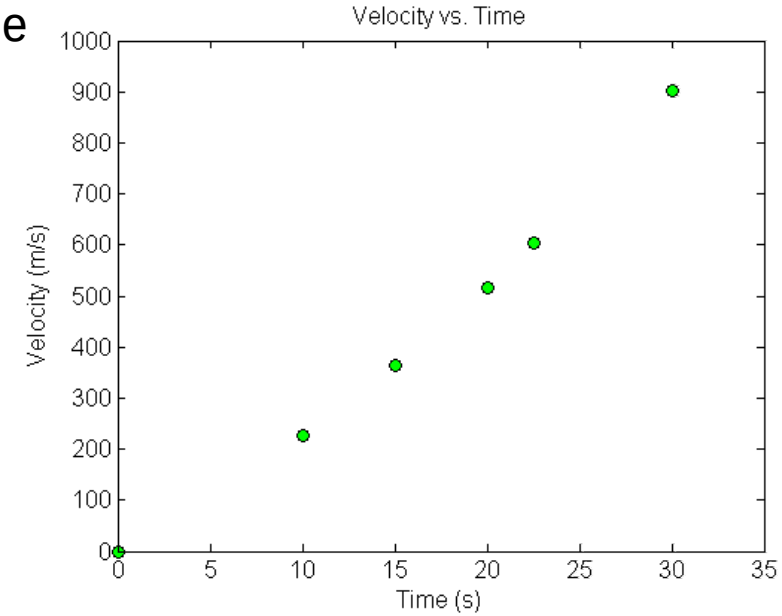


Método diferencia dividida de Newton - ejemplo

- The upward velocity of a rocket is given as a function of time in the following table.

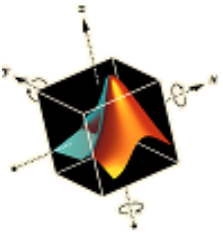
Velocity as a function of time

$t, (\mathrm{s})$	$v(t), (\mathrm{m/s})$
0	0
10	227.04
15	362.78
20	517.35
22.5	602.97
30	901.67



Velocity vs. time data

- Find the velocity at t=16 seconds using the Newton Divided Difference method for cubic interpolation.



Método diferencia dividida de Newton - ejemplo

Solution

The velocity profile is chosen as

$$v(t) = b_0 + b_1(t - t_0) + b_2(t - t_0)(t - t_1) + b_3(t - t_0)(t - t_1)(t - t_2)$$

we need to choose four data points that are closest to $t = 16$

$t_0 = 10, \quad v(t_0) = 227.04$

$t_1 = 15, \quad v(t_1) = 362.78$

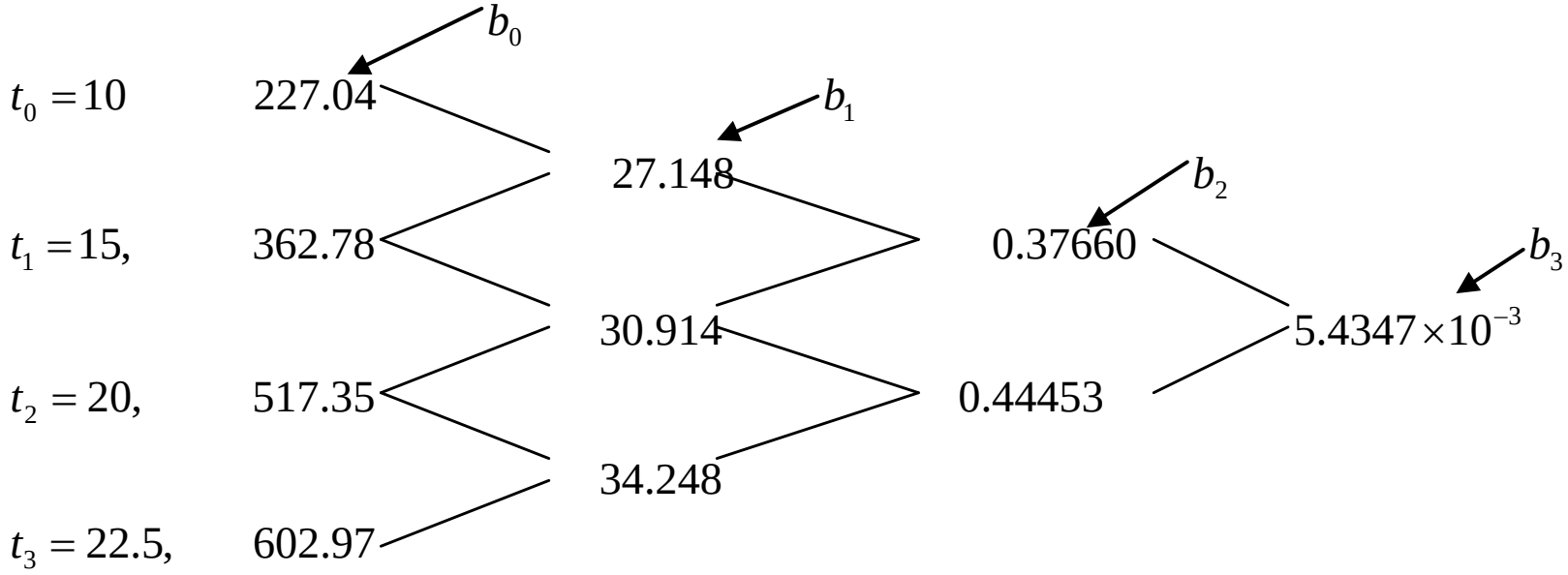
$t_2 = 20, \quad v(t_2) = 517.35$

$t_3 = 22.5, \; v(t_3) = 602.97$

The values of the constants are found as:

$$b_0 = 227.04; \; b_1 = 27.148; \; b_2 = 0.37660; \; b_3 = 5.4347 \times 10^{-3}$$

Método diferencia dividida de Newton - ejemplo



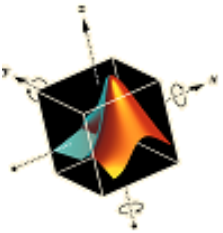
$b_0 = 227.04;\ b_1 = 27.148;\ b_2 = 0.37660;\ b_3 = 5.4347 \times 10^{-3}$

Hence

$$v(t) = b_0 + b_1(t - t_0) + b_2(t - t_0)(t - t_1) + b_3(t - t_0)(t - t_1)(t - t_2)$$

$$= 227.04 + 27.148(t - 10) + 0.3766(t - 10)(t - 15) + 5.4347 \cdot 10^{-3}(t - 10)(t - 15)(t - 20)$$

At $t = 16$ $v(16) = 392.06\ \text{m/s}$



Interpolación de Lagrange

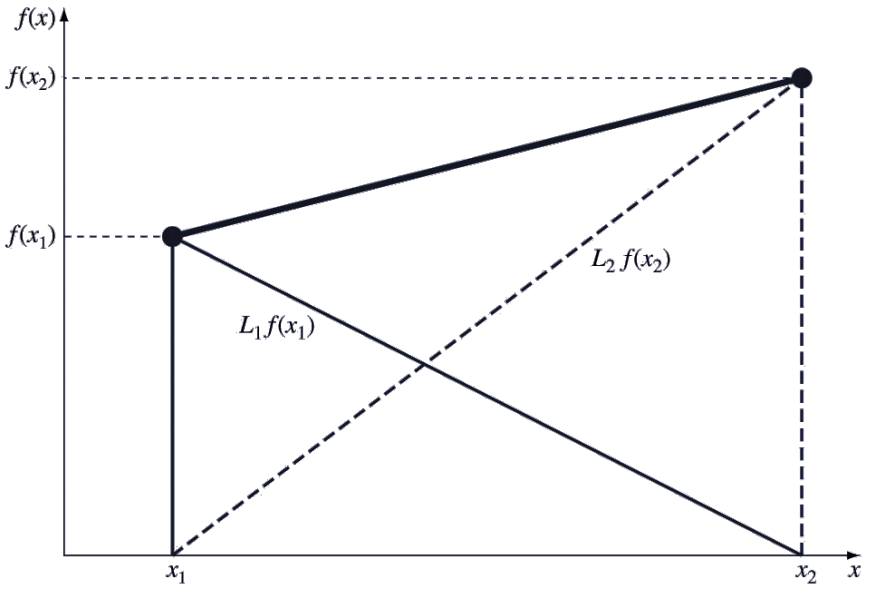
- Another method that uses shifted value to express an interpolating polynomial is the *Lagrange interpolating polynomial*.
- The differences between a simply polynomial and Lagrange interpolating polynomials for first and second order polynomials is:

Order	Simple	Lagrange
1st	$f_1(x) = a_1 + a_2 x$	$f_1(x) = L_1 f(x_1) + L_2 f(x_2)$
2nd	$f_2(x) = a_1 + a_2 x + a_3 x^2$	$f_2(x) = L_1 f(x_1) + L_2 f(x_2) + L_3 f(x_3)$

where the L_i are weighting coefficients that are functions of x.

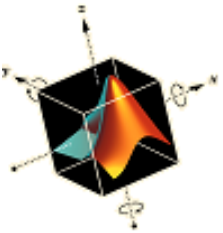
Interpolación de Lagrange

- The first-order Lagrange interpolating polynomial may be obtained from a weighted combination of two linear interpolations, as shown.
- The resulting formula based on known points x_1 and x_2 and the values of the dependent function at those points is:



$$f_1(x) = L_1 f(x_1) + L_2 f(x_2) \quad \text{where} \quad L_1 = \frac{x - x_2}{x_1 - x_2}, L_2 = \frac{x - x_1}{x_2 - x_1}$$

$$f_1(x) = \frac{x - x_2}{x_1 - x_2} f(x_1) + \frac{x - x_1}{x_2 - x_1} f(x_2)$$



Interpolación de Lagrange

- In general, the Lagrange polynomial interpolation for *n+1* points is:

$$f_n(x) = \sum_{i=0}^{n} L_i(x) f(x_i)$$

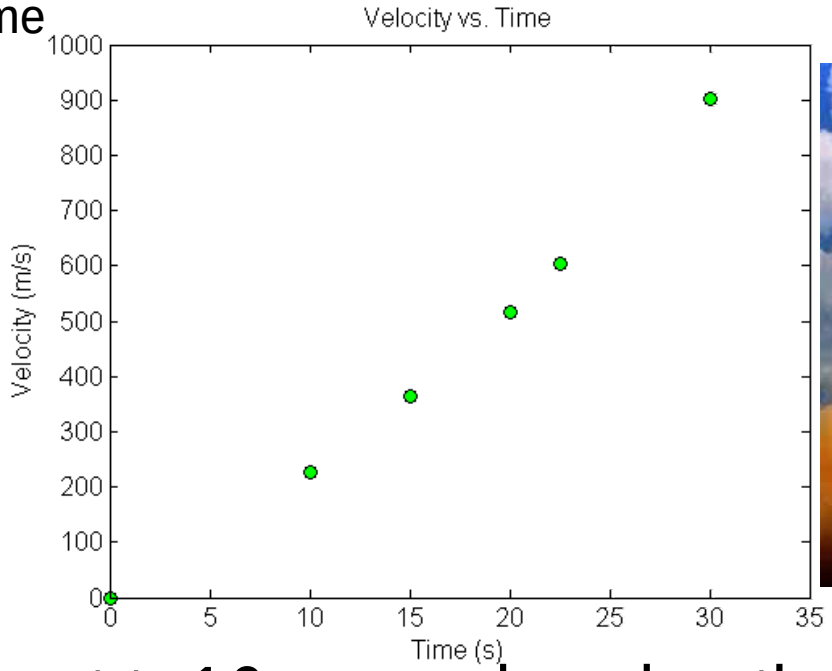
where L_i is given by: $L_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j}$

Interpolación de Lagrange - ejemplo

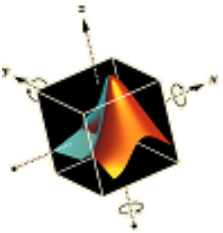
- The upward velocity of a rocket is given as a function of time in the following table.

Velocity as a function of time

$t, (\mathrm{s})$	$v(t), (\mathrm{m/s})$
0	0
10	227.04
15	362.78
20	517.35
22.5	602.97
30	901.67



- Find the velocity at t=16 seconds using the Lagrangian method for cubic interpolation.



Interpolación de Lagrange - ejemplo

Solution

$t_0 = 10, \quad v(t_0) = 227.04$

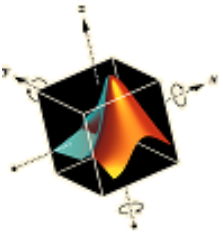
$t_1 = 15, \quad v(t_1) = 362.78$

$t_2 = 20, \quad v(t_2) = 517.35$

$t_3 = 22.5, \quad v(t_3) = 602.97$

$$L_0(t) = \prod_{\substack{j=0 \\ j \neq i}}^{3} \frac{t - t_j}{t_0 - t_j} = \left(\frac{t - t_1}{t_0 - t_1}\right)\left(\frac{t - t_2}{t_0 - t_2}\right)\left(\frac{t - t_3}{t_0 - t_3}\right) \quad L_1(t) = \prod_{\substack{j=0 \\ j \neq i}}^{3} \frac{t - t_j}{t_1 - t_j} = \left(\frac{t - t_0}{t_1 - t_0}\right)\left(\frac{t - t_2}{t_1 - t_2}\right)\left(\frac{t - t_3}{t_1 - t_3}\right)$$

$$L_2(t) = \prod_{\substack{j=0 \\ j \neq i}}^{3} \frac{t - t_j}{t_2 - t_j} = \left(\frac{t - t_0}{t_2 - t_0}\right)\left(\frac{t - t_1}{t_2 - t_1}\right)\left(\frac{t - t_3}{t_2 - t_3}\right) \quad L_3(t) = \prod_{\substack{j=0 \\ j \neq i}}^{3} \frac{t - t_j}{t_3 - t_j} = \left(\frac{t - t_0}{t_3 - t_0}\right)\left(\frac{t - t_1}{t_3 - t_1}\right)\left(\frac{t - t_2}{t_3 - t_2}\right)$$



Interpolación de Lagrange - ejemplo

Solution

$$v(t)=\left(\frac{t-t_1}{t_0-t_1}\right)\left(\frac{t-t_2}{t_0-t_2}\right)\left(\frac{t-t_3}{t_0-t_3}\right)v(t_1)+\left(\frac{t-t_0}{t_1-t_0}\right)\left(\frac{t-t_2}{t_1-t_2}\right)\left(\frac{t-t_3}{t_1-t_3}\right)v(t_2)$$

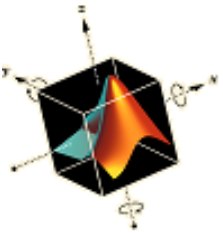
$$+\left(\frac{t-t_0}{t_2-t_0}\right)\left(\frac{t-t_1}{t_2-t_1}\right)\left(\frac{t-t_3}{t_2-t_3}\right)v(t_2)+\left(\frac{t-t_1}{t_3-t_1}\right)\left(\frac{t-t_1}{t_3-t_1}\right)\left(\frac{t-t_2}{t_3-t_2}\right)v(t_3)$$

$$v(16)=\left(\frac{16-15}{10-15}\right)\left(\frac{16-20}{10-20}\right)\left(\frac{16-22.5}{10-22.5}\right)(227.04)+\left(\frac{16-10}{15-10}\right)\left(\frac{16-20}{15-20}\right)\left(\frac{16-22.5}{15-22.5}\right)(362.78)$$

$$+\left(\frac{16-10}{20-10}\right)\left(\frac{16-15}{20-15}\right)\left(\frac{16-22.5}{20-22.5}\right)(517.35)+\left(\frac{16-10}{22.5-10}\right)\left(\frac{16-15}{22.5-15}\right)\left(\frac{16-20}{22.5-20}\right)(602.97)$$

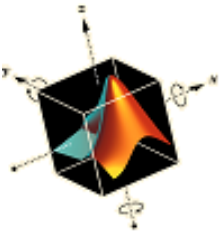
$$=(-0.0416)(227.04)+(0.832)(362.78)+(0.312)(517.35)+(-0.1024)(602.97)$$

$$=392.06\,\text{m/s}$$



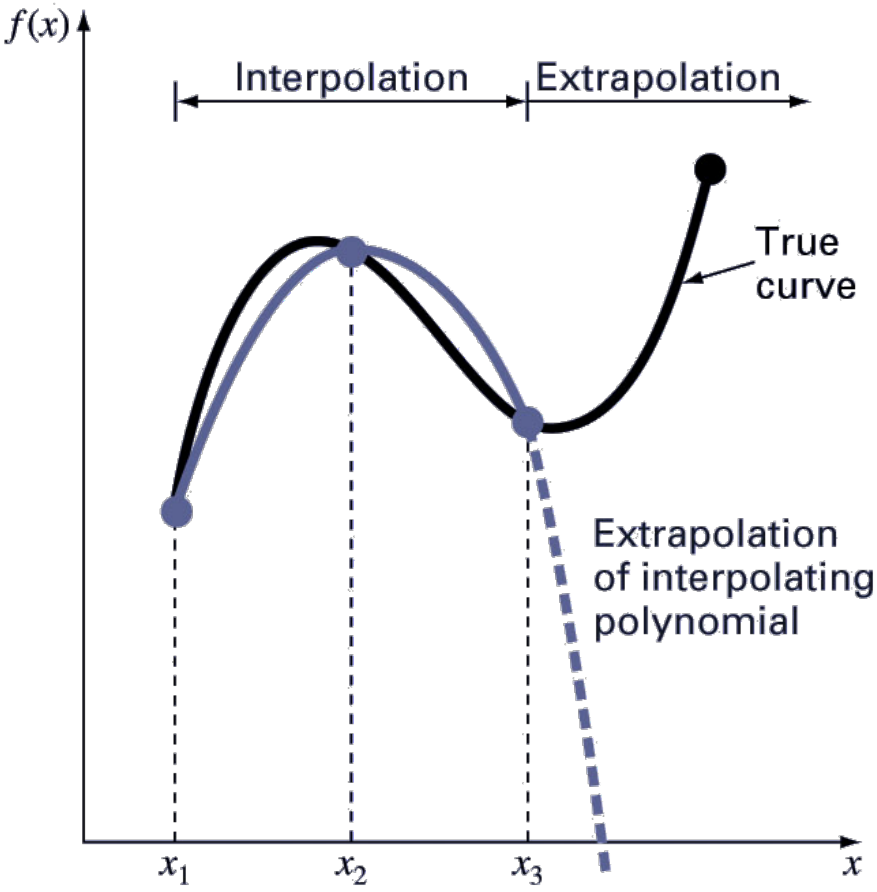
Interpolación inversa

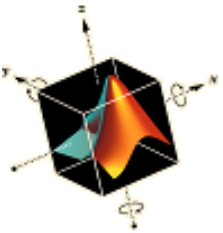
- Interpolation general means finding some value $f(x)$ for some x that is between given independent data points.
- Sometimes, it will be useful to find the x for which $f(x)$ is a certain value - this is *inverse interpolation*.
- Rather than finding an interpolation of x as a function of $f(x)$, it may be useful to find an equation for $f(x)$ as a function of x using interpolation and then solve the corresponding roots problem:
  $f(x)-f_{desired}=0$ for x.



Extrapolación

- *Extrapolation* is the process of estimating a value of $f(x)$ that lies outside the range of the known base points $x_1, x_2, \ldots, x_n$.
- Extrapolation represents a step into the unknown, and extreme care should be exercised when extrapolating!

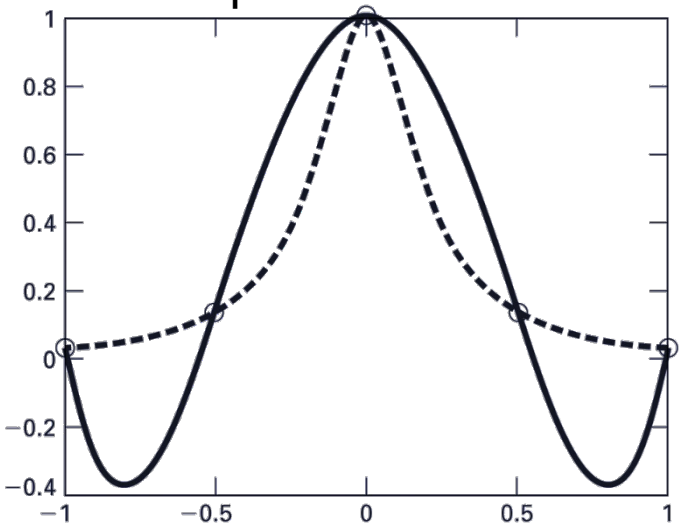


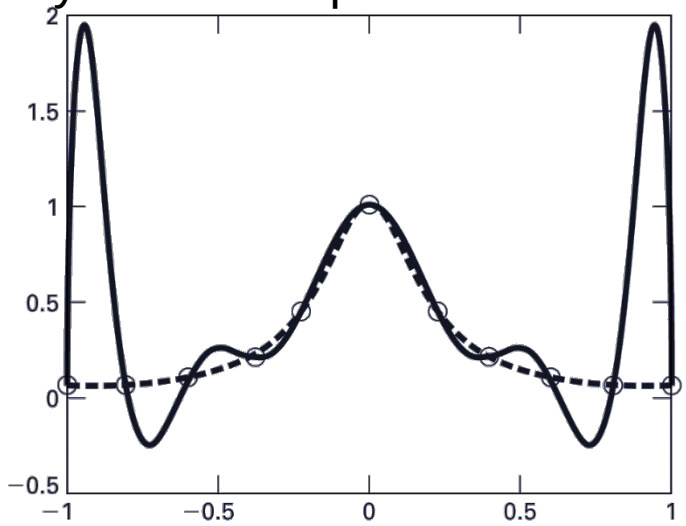


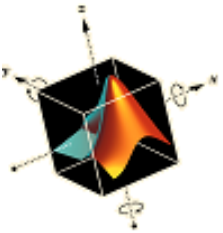
Oscilaciones

- Higher-order polynomials can not only lead to round-off errors due to ill-conditioning, but can also introduce oscillations to an interpolation or fit where they should not be.

The dashed line represents an function, the circles represent samples of the function, and the solid line represents the results of a polynomial interpolation



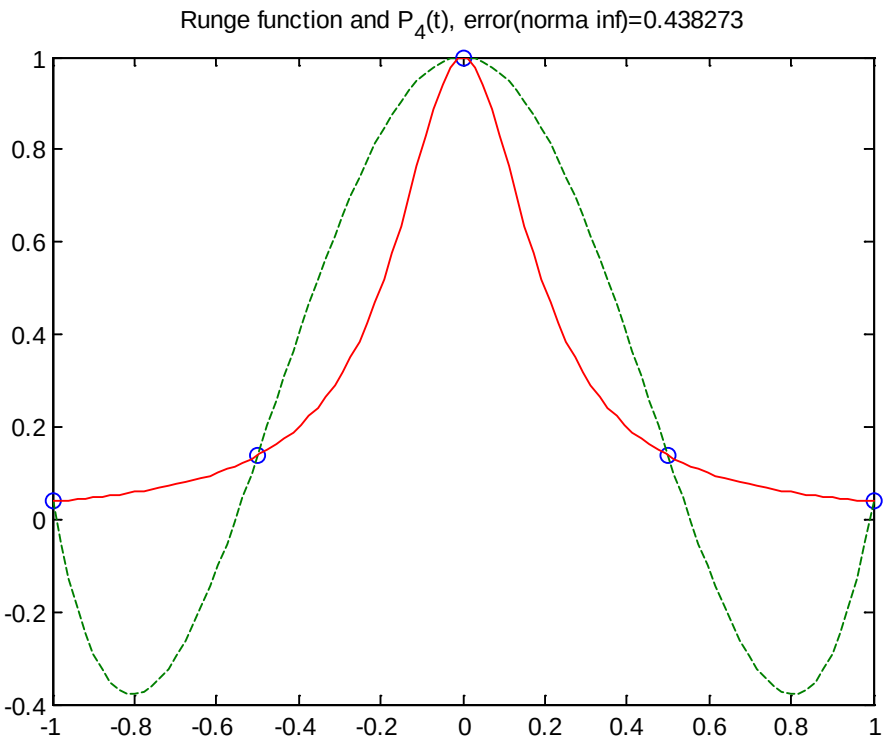


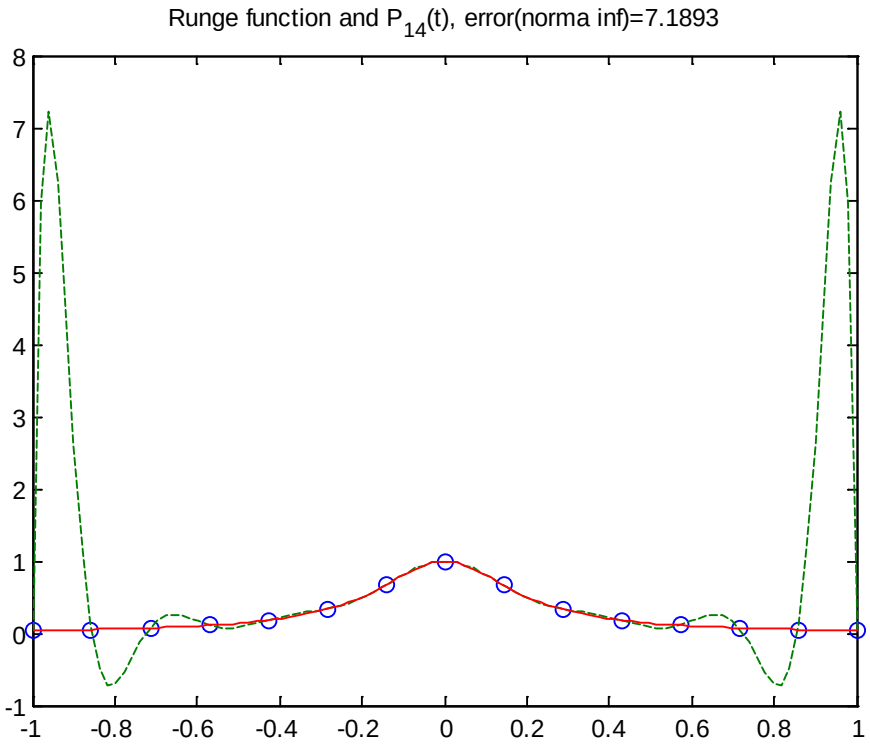


Oscilaciones - ejemplo

- Interpolate for the Runge function

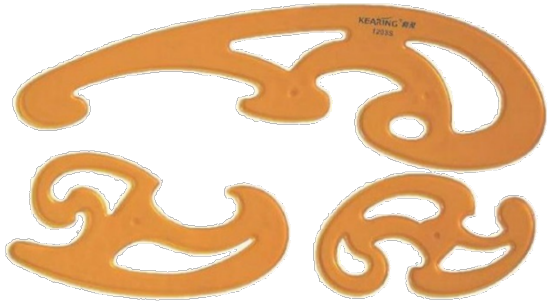
$$f(x) = \frac{1}{1+25x^2} \qquad -1 < x < 1 \qquad \text{error} = \left\| f(t) - P_n(t) \right\|_\infty$$

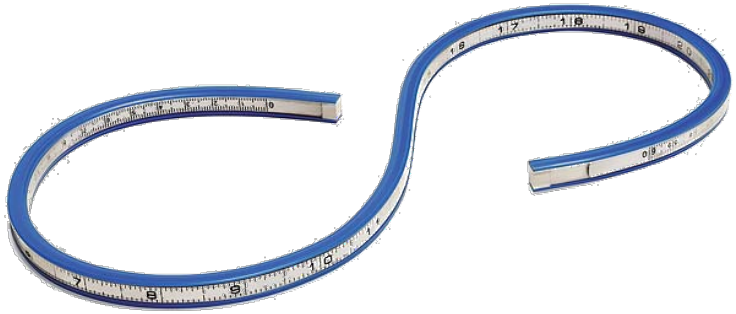


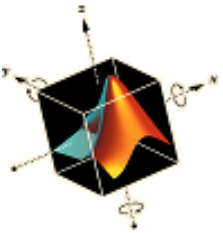


Interpolación polinómica a trozos - Splines

- An alternative approach to using a single (n-1)th order polynomial to interpolate between n points is to apply lower-order polynomials in a piecewise fashion to subsets of data points.
- These connecting polynomials are called *spline functions*.
- Splines minimize oscillations and reduce round-off error due to their lower-order nature.

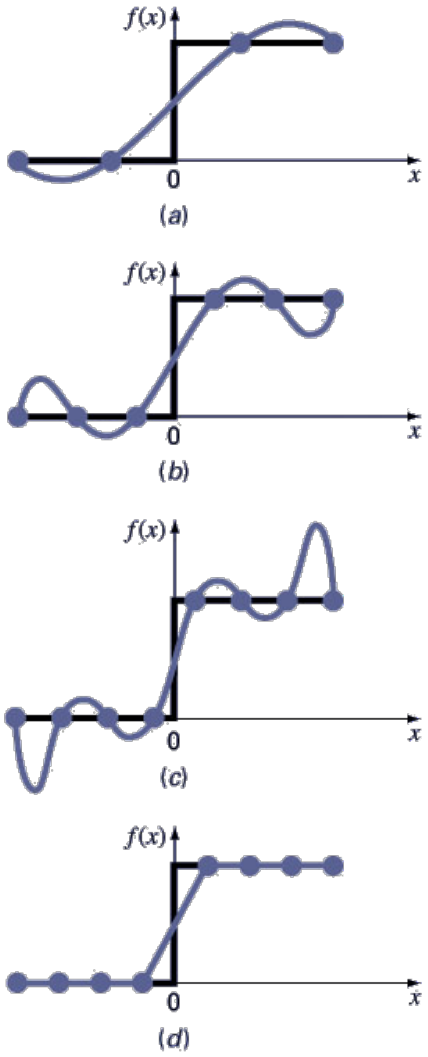


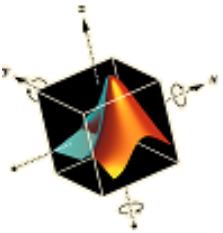




Polinomios de gado alto vs. Splines

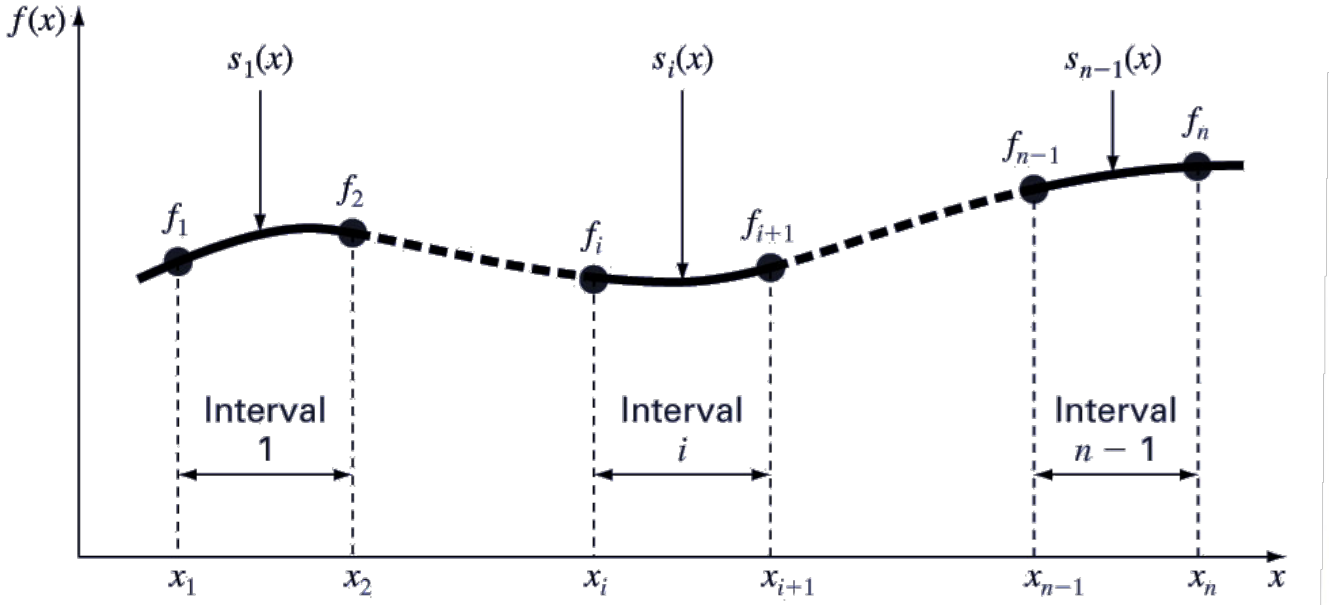
- Splines eliminate oscillations by using small subsets of points for each interval rather than every point. This is especially useful when there are jumps in the data:
  a) 3rd order polynomial
  b) 5th order polynomial
  c) 7th order polynomial
  d) Linear spline
     - seven 1st order polynomials generated by using pairs of points at a time.

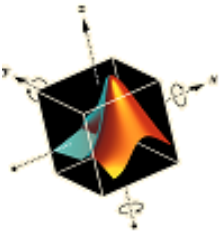




Desarrollo de Splines

- Spline function ($s_i(x)$)coefficients are calculated for each interval of a data set.
- The number of data points (f_i) used for each spline function depends on the order of the spline function.





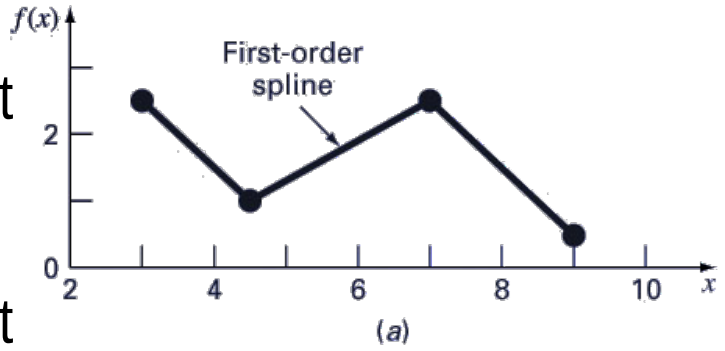
Polinomios de grado alto vs. Splines

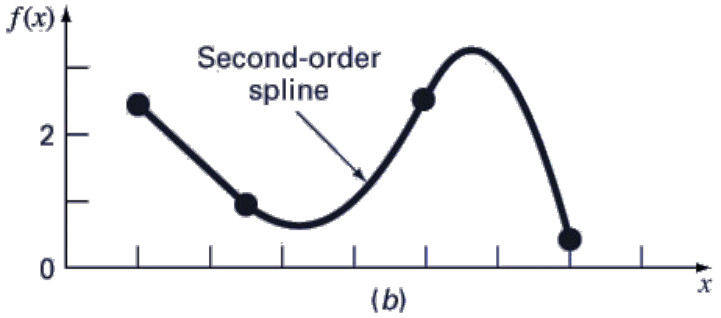
a) First-order splines find straight-line equations between each pair of points that
   - Go through the points

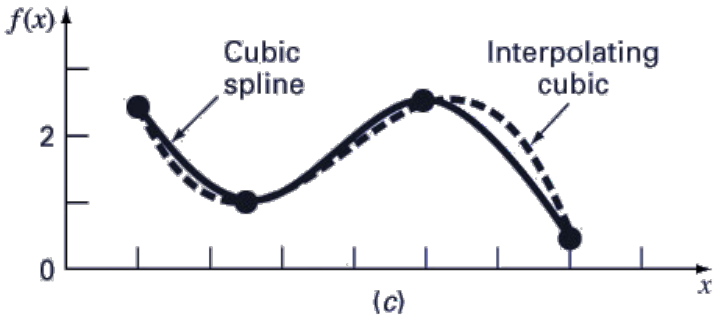
b) Second-order splines find quadratic equations between each pair of points that
   - Go through the points
   - Match first derivatives at the interior points

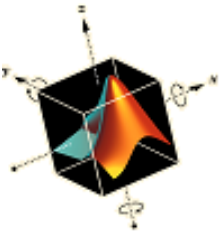
c) Third-order splines find cubic equations between each pair of points that
   - Go through the points
   - Match first and second derivatives at the interior points

Note that the results of cubic spline interpolation are different from the results of an interpolating cubic.



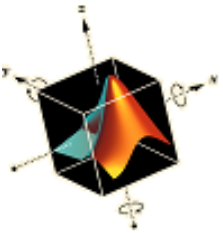






Splines cúbicos

- While data of a particular size presents many options for the order of spline functions, *cubic splines* are preferred because they provide the simplest representation that exhibits the desired appearance of smoothness.
  - Linear splines have discontinuous first derivatives
  - Quadratic splines have discontinuous second derivatives and require setting the second derivative at some point to a pre-determined value
  - Quartic or higher-order splines tend to exhibit the instabilities inherent in higher order polynomials (ill-conditioning or oscillations)

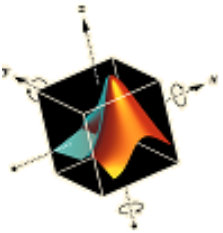


Splines - definición

- Given *n*+1 data points $(x_0, y_0), \ldots, (x_n, y_n)$ where

$$a = x_0 < x_1 < \ldots < x_{n-1} < x_n = b$$

- A spline of degree *m* is a function $S(x)$ which satisfy the following conditions:
- For $x \in [x_i, x_{i+1}]$, $S(x) = S_i(x)$: polynomial of degree $\leq m$
- $S^{(m-1)}$ exists and continuous at the interior points $x_1, \ldots, x_{n-1}$ i.e. $\lim_{x \to x_i^-} S_{i-1}^{(m-1)}(x) = \lim_{x \to x_i^+} S_{i-1}^{(m-1)}(x)$
- Spline is piecewise polynomial of degree *m* that is *m-1* times continuously differentiable

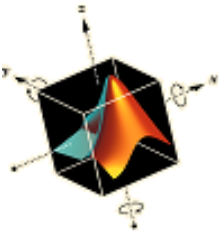


Splines cúbicos

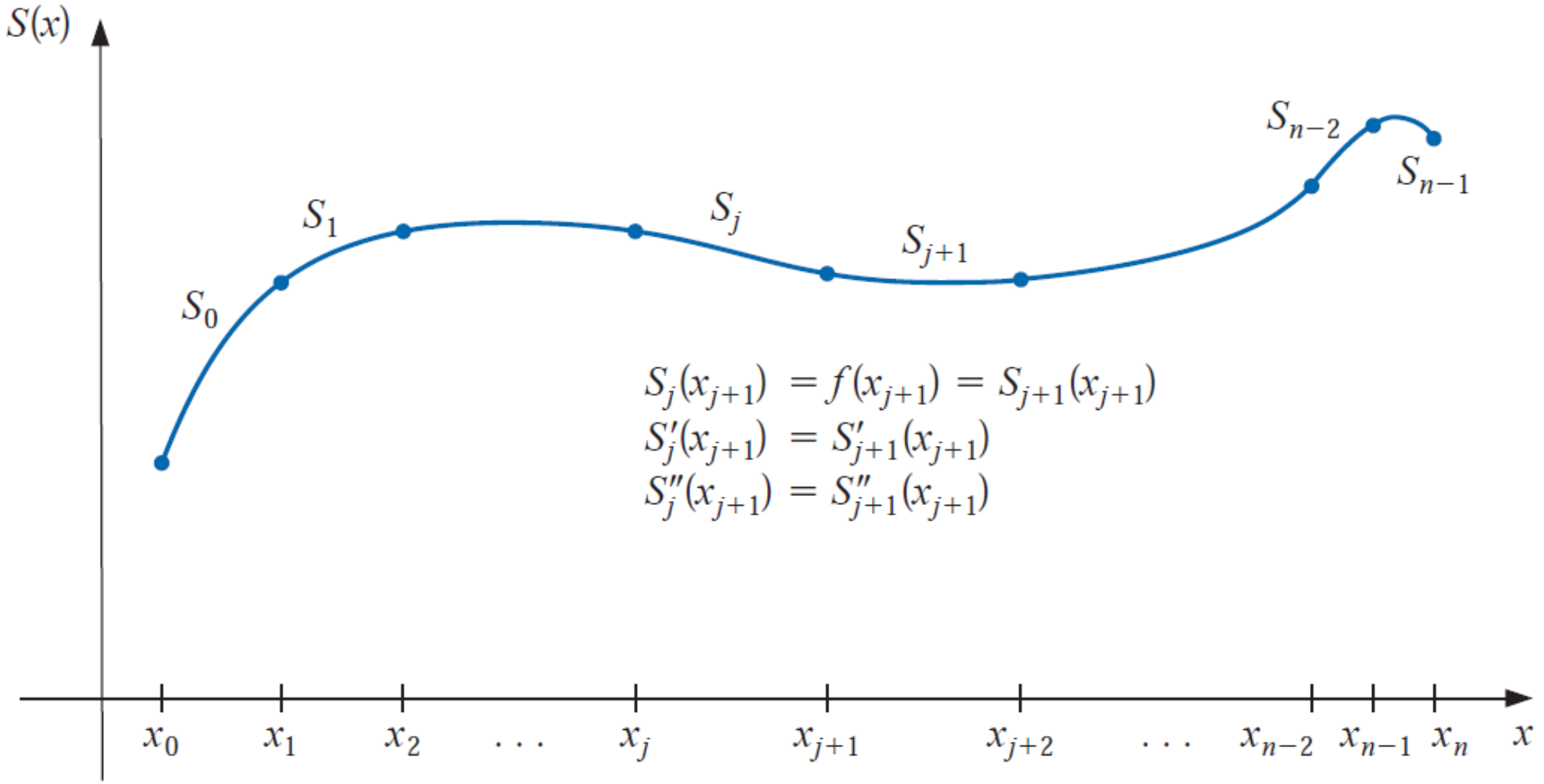
- Given *n*+1 data points $(x_0, y_0), \ldots, (x_n, y_n)$, the *i*th spline function for a cubic spline can be written as:

$$s_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3$$

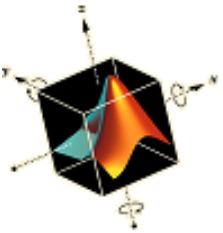
- there are *n* intervals and thus 4*n* unknowns to evaluate to solve all the spline function coefficients.
- 2*n* conditions needed to interpolate $f(x_0), \ldots, f(x_n)$
- 2(*n*-1) conditions needed to ensure S', S'' are continuous at the interior points
- 2 free conditions (some described later)



Splines cúbicos



$$S_j(x_{j+1}) = f(x_{j+1}) = S_{j+1}(x_{j+1})$$
$$S_j'(x_{j+1}) = S_{j+1}'(x_{j+1})$$
$$S_j''(x_{j+1}) = S_{j+1}''(x_{j+1})$$



Cálculo de los coeficientes de Splines cúbicos

- *n*+1 data points $(x_0, y_0), \ldots, (x_n, y_n)$
- 2*n* conditions needed to interpolate the first and last point of the interval

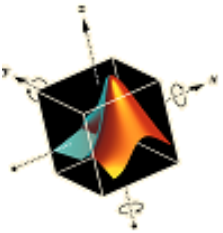
$$s_i(x_i) = f_i \Rightarrow a_i = f_i$$

$$s_i(x_{i+1}) = f_{i+1} \Rightarrow s_i(x_{i+1}) = a_i + b_i(x_{i+1} - x_i) + c_i(x_{i+1} - x_i)^2 + d_i(x_{i+1} - x_i)^3 = f_{i+1}$$

- 2(*n*-1) conditions to ensure the first and second derivative are continuous at each interior points

$$s_i'(x_{i+1}) = s_{i+1}'(x_{i+1}) \Rightarrow b_i + 2c_i(x_{i+1} - x_i) + 3d_i(x_{i+1} - x_i)^2 = b_{i+1}$$

$$s_i''(x_{i+1}) = s_{i+1}''(x_{i+1}) \Rightarrow 2c_i + 6d_i(x_{i+1} - x_i) = 2c_{i+1}$$



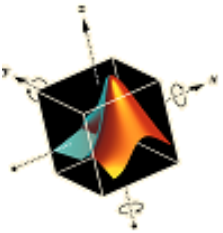
Cálculo de los coeficientes de Splines cúbicos

- Hay varias opciones para las ecuaciones en los extremos:
  - **Natural** (or free) end conditions - assume the second derivative at the end knots are zero: $s_0''(x_0) = s_{n-1}''(x_n) = 0$
  - **Clamped** end conditions - assume the first derivatives at the first and last knots are known:

$$s_0'(x_0) = f'(x_0),\ s_{n-1}'(x_n) = f'(x_n)$$

  - "Not-a-knot" end conditions - force continuity of the third derivative at the second and penultimate points (the first and the last two intervals having the same spline function)

$$s_0'''(x_1) = s_1'''(x_1),\ \ s_{n-2}'''(x_{n-1}) = s_{n-1}'''(x_{n-1})$$



Cálculo de los coeficientes de Splines cúbicos

$$S_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3 \qquad i = 0,1,\ldots,n-1$$

$$S_i(x_i) = a_i = f(x_i)$$

$$a_{i+1} = S_{i+1}(x_{i+1}) = S_i(x_{i+1}) = a_i + b_i(x_{i+1} - x_i) + c_i(x_{i+1} - x_i)^2 + d_i(x_{i+1} - x_i)^3$$

$$i = 0,1,\ldots,n-2$$

if $h_i = (x_{i+1} - x_i)$ for $i = 0,1,\ldots,n-1$ and $a_n = f(x_n)$

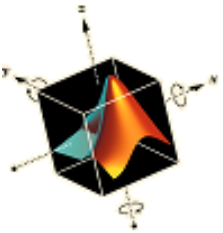
$$a_{i+1} = a_i + b_i h_i + c_i h_i^2 + d_i h_i^3$$

$$b_n = S'(x_n) \qquad S_i'(x) = b_i + 2c_i(x - x_i) + 3d_i(x - x_i)^3 \qquad S_i'(x_i) = b_i$$

$$b_{i+1} = b_i + 2c_i h_i + 3d_i h_i^2$$

$$c_n = S''(x_n)/2$$

$$c_{i+1} = c_i + 3d_i h_i$$



Cálculo de los coeficientes de Splines cúbicos

$$a_{i+1} = a_i + b_i h_i + \frac{h_i^2}{3}(2c_i + c_{i+1})$$

$$b_{i+1} = b_i + h_i(c_i + c_{i+1})$$

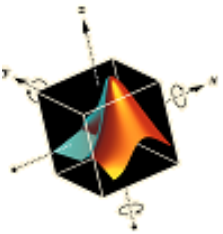
$$b_i = \frac{1}{h_i}(a_{i+1} - a_i) - \frac{h_i}{3}(2c_i + c_{i+1})$$

$$b_{i-1} = \frac{1}{h_{i-1}}(a_i - a_{i-1}) - \frac{h_{i-1}}{3}(2c_{i-1} + c_i)$$

$$h_{i-1}c_{i-1} + 2(h_{i-1} + h_i)c_i + h_i c_{i+1} = \frac{3}{h_i}(a_{i+1} - a_i) - \frac{3}{h_{i-1}}(a_i - a_{i-1})$$

$\{c_i\}_{i=0}^{n}$ unknowns

given values $\{h_i\}_{i=0}^{n-1}$ and $\{a_i\}_{i=0}^{n}$ by nodes $\{x_i\}_{i=0}^{n}$ and values of f



Cálculo de los coeficientes de Splines cúbicos naturales

f defined at $a = x_0 < x_1 < \cdots < x_n = b$ $\quad S''(a) = 0 \quad S''(b) = 0$

$c_0 = c_n = 0$

$A\mathbf{x} = \mathbf{b}$

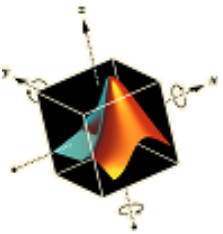
- Tridiagonal
- Symmetric
- Positive definite
- Strictly diagonally dominant

$$A = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ h_0 & 2(h_0+h_1) & h_1 & & \vdots \\ 0 & h_1 & 2(h_1+h_2) & \ddots & 0 \\ \vdots & & \ddots & \ddots & h_{n-1} \\ 0 & \cdots & 0 & 0 & 1 \end{bmatrix}$$

Invertible

Gaussian elimination

$$\mathbf{b} = \begin{bmatrix} 0 \\ \frac{3}{h_1}(a_2 - a_1) - \frac{3}{h_0}(a_1 - a_0) \\ \vdots \\ \frac{3}{h_{n-1}}(a_n - a_{n-1}) - \frac{3}{h_{n-2}}(a_{n-1} - a_{n-2}) \\ 0 \end{bmatrix} \qquad \mathbf{x} = \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{bmatrix}$$



Cálculo de los coeficientes de Splines cúbicos fijados

f defined at $a = x_0 < x_1 < \cdots < x_n = b$ $S''(a) = f'(a)$ $S''(b) = f'(b)$

$$A\mathbf{x} = \mathbf{b}$$

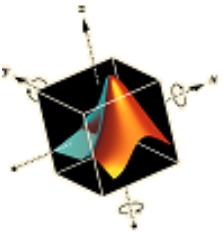
- Tridiagonal
- Symmetric
- Positive definite
- Strictly diagonally dominant

$$A = \begin{bmatrix} 2h_0 & h_0 & 0 & \cdots & 0 \\ h_0 & 2(h_0+h_1) & h_1 & & \vdots \\ 0 & h_1 & 2(h_1+h_2) & \ddots & 0 \\ \vdots & & \ddots & h_{n-2} & 2(h_{n-2}+h_{n-1}) & h_{n-1} \\ 0 & \cdots & 0 & h_{n-1} & 2h_{n-1} \end{bmatrix}$$

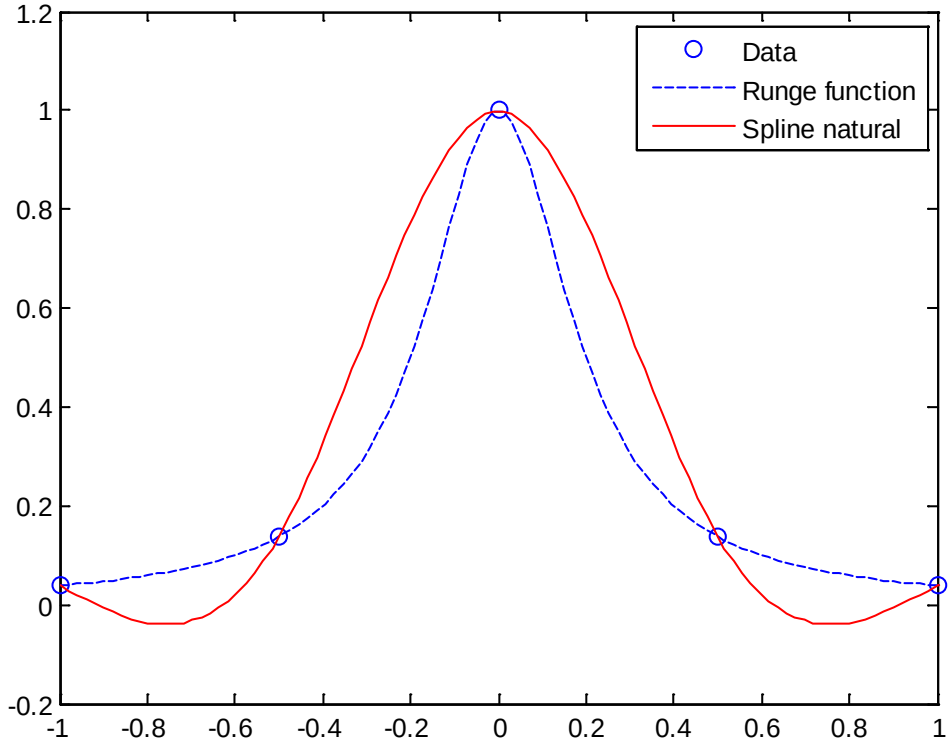
Invertible

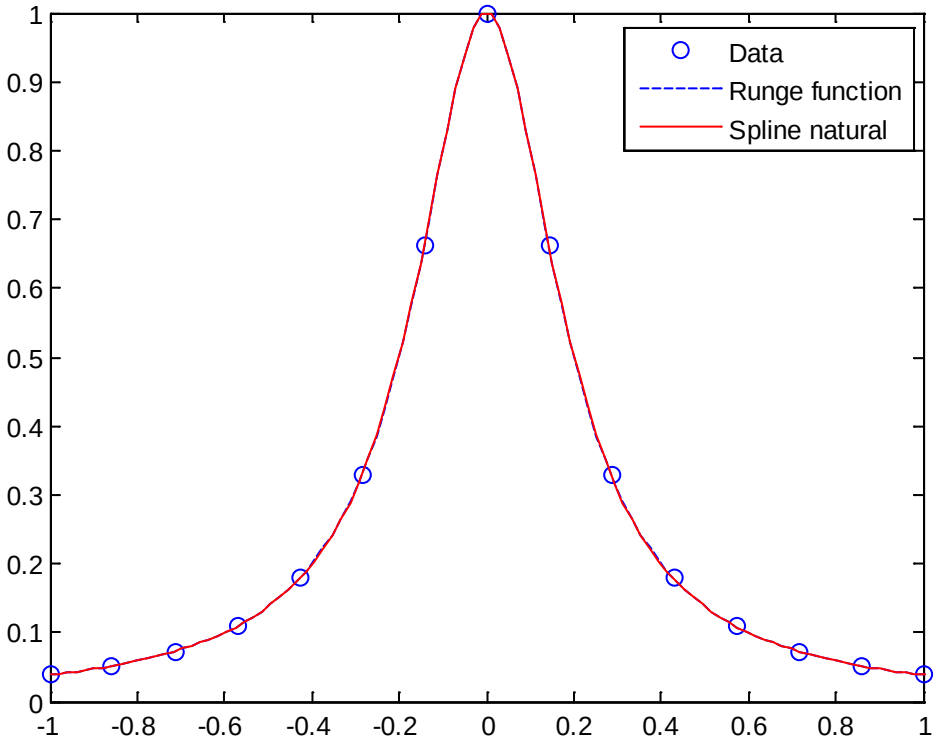
Gaussian elimination

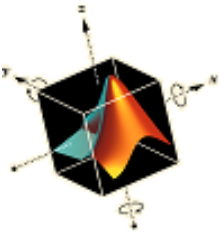
$$\mathbf{b} = \begin{bmatrix} \frac{3}{h_0}(a_1 - a_0) - 3f'(a) \\ \frac{3}{h_1}(a_2 - a_1) - \frac{3}{h_0}(a_1 - a_0) \\ \vdots \\ \frac{3}{h_{n-1}}(a_n - a_{n-1}) - \frac{3}{h_{n-2}}(a_{n-1} - a_{n-2}) \\ 3f'(b) - \frac{3}{h_{n-1}}(a_n - a_{n-1}) \end{bmatrix} \qquad \mathbf{x} = \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{bmatrix}$$



Interpolación polinómica a trozos – ejemplo Spline natural







Interpolación polinómica a trozos en MATLAB

- MATLAB has several built-in functions to implement piecewise interpolation. The first is spline:

```
yy=spline(x, y, xx)
```

This performs cubic spline interpolation, generally using not-a-knot conditions. If `y` contains two more values than `x` has entries, then the first and last value in `y` are used as the derivatives at the end points (i.e. clamped)

Interpolación polinómica a trozos en MATLAB – ejemplo Not a Knot

- Generate data:

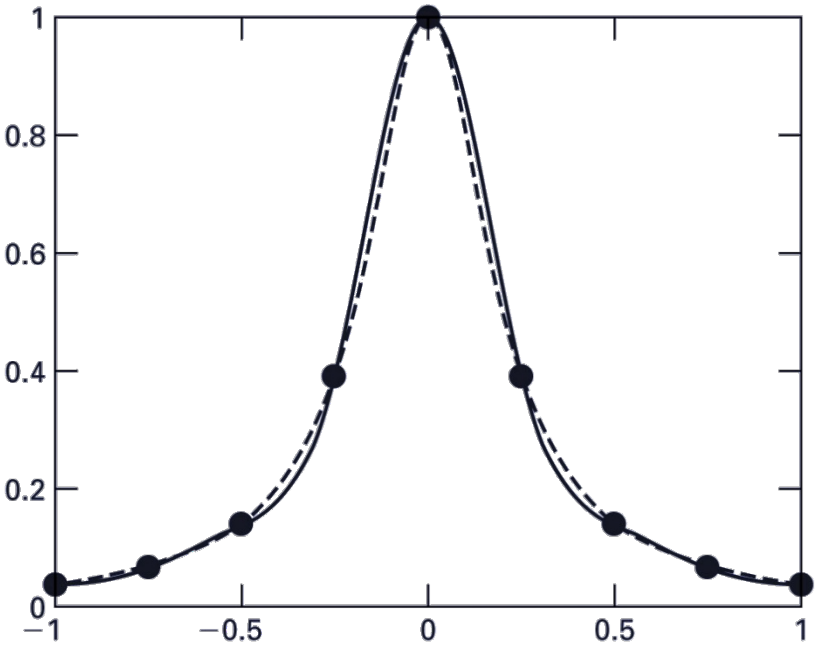
```
x = linspace(-1, 1, 9);
y = 1./(1+25*x.^2);
```

- Calculate 100 model points and determine not-a-knot interpolation

```
xx = linspace(-1, 1);
yy = spline(x, y, xx);
```

- Calculate actual function values at model points and data points, the 9-point not-a-knot interpolation (solid), and the actual function (dashed),

```
yr = 1./(1+25*xx.^2)
plot(x, y, 'o', xx, yy, '-', xx, yr, '--')
```



Interpolación polinómica a trozos en MATLAB – ejemplo clamped

- Generate data w/ first derivative information:

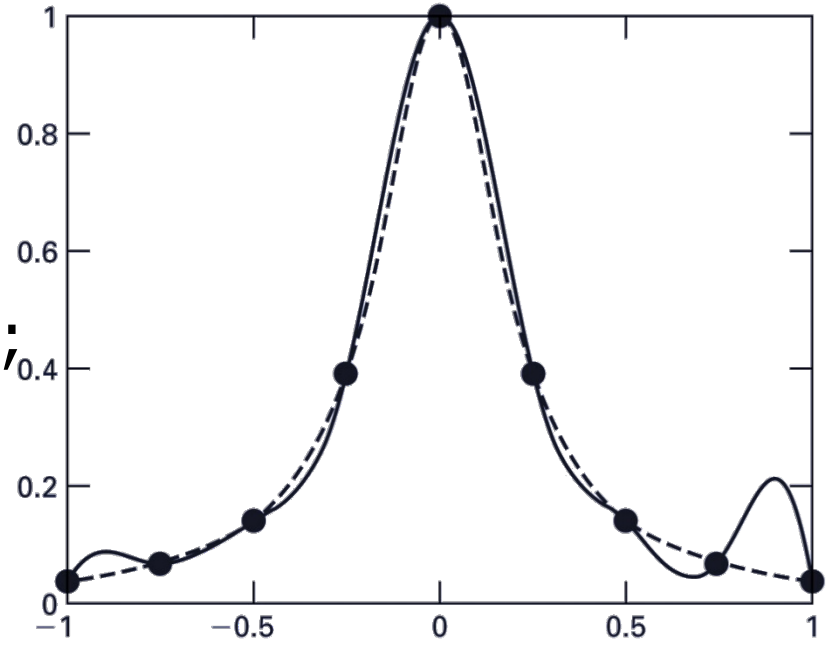
```
x = linspace(-1, 1, 9);
y = 1./(1+25*x.^2);
yc = [1 y -4]
```

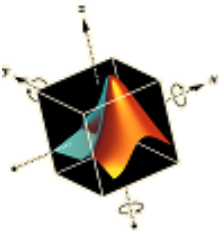
- Calculate 100 model points and determine not-a-knot interpolation

```
xx = linspace(-1, 1);
yyc = spline(x, yc, xx);
```

- Calculate actual function values at model points and data points, the 9-point clamped interpolation (solid), and the actual function (dashed),

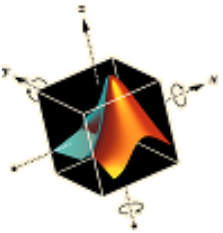
```
yr = 1./(1+25*xx.^2)
plot(x, y, 'o', xx, yyc, '-', xx, yr, '--')
```



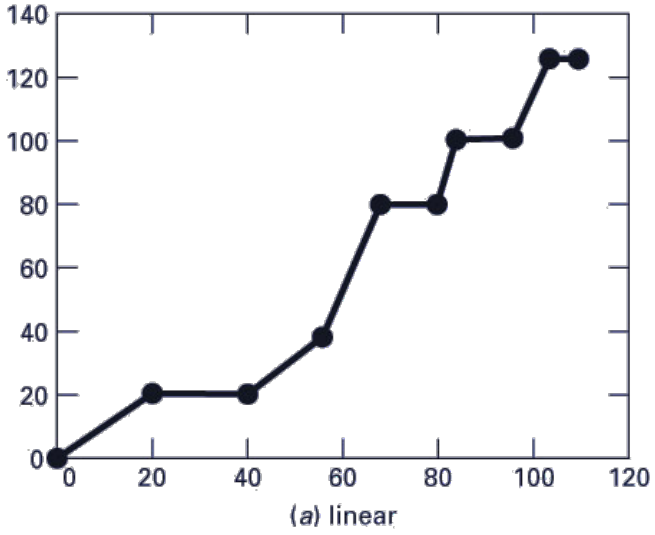


Función `interp1` de MATLAB

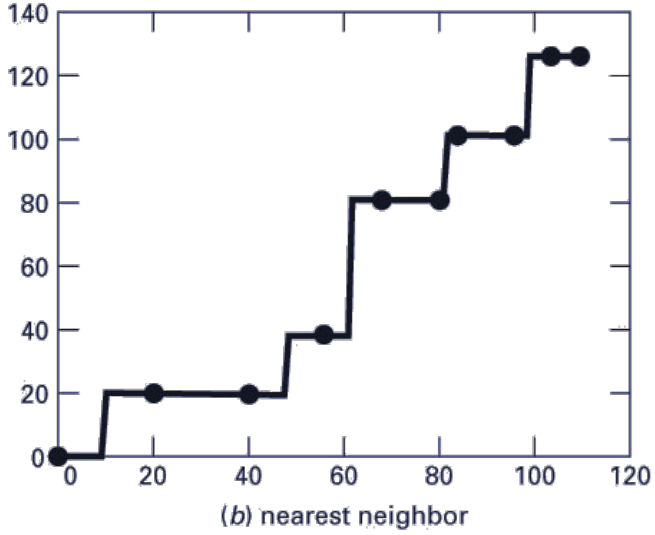
- While `spline` can only perform cubic splines, MATLAB's `interp1` function can perform several different kinds of interpolation:
  `yi = interp1(x, y, xi, 'method')`
  - `x` & `y` contain the original data
  - `xi` contains the points at which to interpolate
  - `'method'` is a string containing the desired method:
    - `'nearest'` - nearest neighbor interpolation
    - `'linear'` - connects the points with straight lines
    - `'spline'` - not-a-knot cubic spline interpolation
    - `'pchip'` or `'cubic'` - piecewise cubic Hermite interpolation



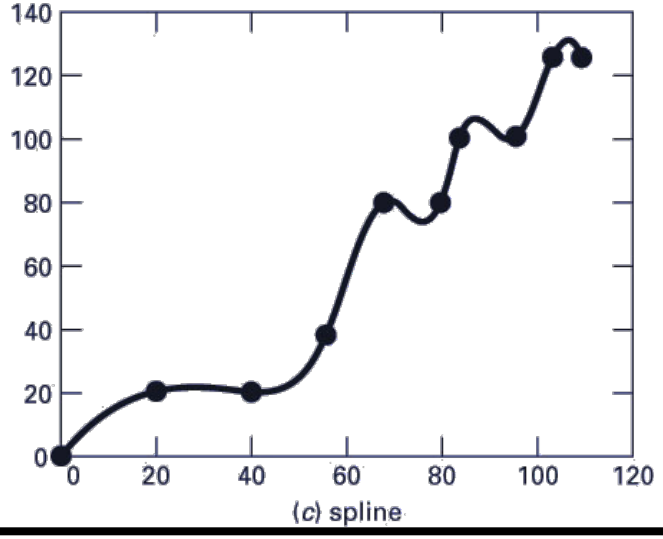
Interpolación polinómica a trozos en MATLAB - comparación



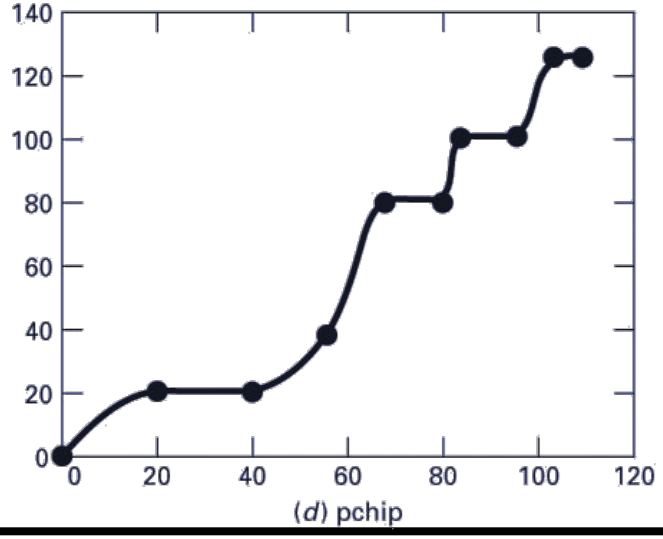
(*a*) linear



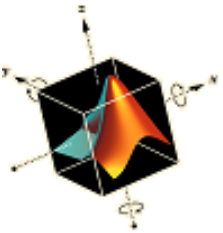
(*b*) nearest neighbor



(*c*) spline



(*d*) pchip



Interpolación multidimensional

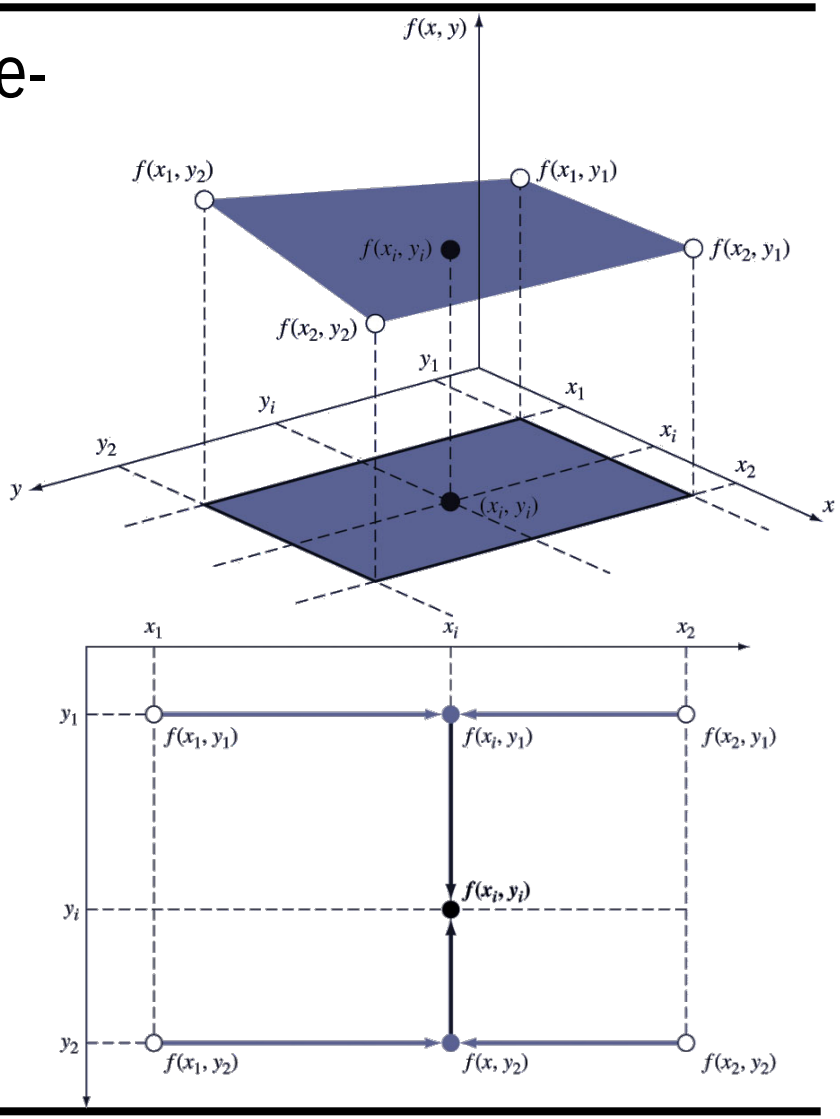
- The interpolation methods for one-dimensional problems can be extended to multidimensional interpolation.
- Example - bilinear interpolation using Lagrange-form equations:

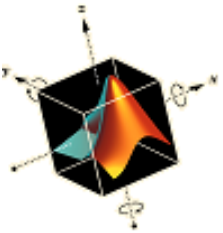
$$f(x_i, y_i) = \frac{x_i - x_2}{x_1 - x_2}\frac{y_i - y_2}{y_1 - y_2} f(x_1, y_1) + \cdots$$

$$\frac{x_i - x_1}{x_2 - x_1}\frac{y_i - y_2}{y_1 - y_2} f(x_2, y_1) + \cdots$$

$$\frac{x_i - x_2}{x_1 - x_2}\frac{y_i - y_1}{y_2 - y_1} f(x_1, y_2) + \cdots$$

$$\frac{x_i - x_1}{x_2 - x_1}\frac{y_i - y_1}{y_2 - y_1} f(x_2, y_2)$$



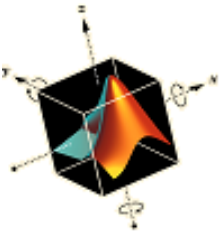


Interpolación multidimensional en MATLAB

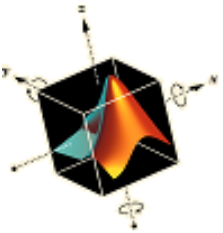
- MATLAB has built-in functions for two- and three-dimensional piecewise interpolation:

```
zi = interp2(x, y, z, xi, yi, 'method')
vi = interp3(x, y, z, v, xi, yi, zi, 'method')
```

- `'method'` is again a string containing the desired method: `'nearest'`, `'linear'`, `'spline'`, `'pchip'`, or `'cubic'`
- For 2-D interpolation, the inputs must either be vectors or same-size matrices.
- For 3-D interpolation, the inputs must either be vectors or same-size 3-D arrays.



Regresión

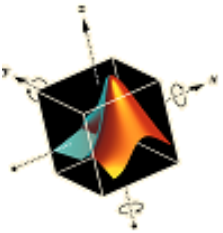


Revisión de estadística – Medida de posición

- *Arithmetic mean*: the sum of the individual data points (y_i) divided by the number of points n:

$$\bar{y} = \frac{\sum y_i}{n}$$

- *Median*: the midpoint of a group of data.
- *Mode*: the value that occurs most frequently in a group of data.



Revisión de estadística – Medida de dispersión

- *Standard deviation*: $$s_y = \sqrt{\frac{S_t}{n-1}}$$

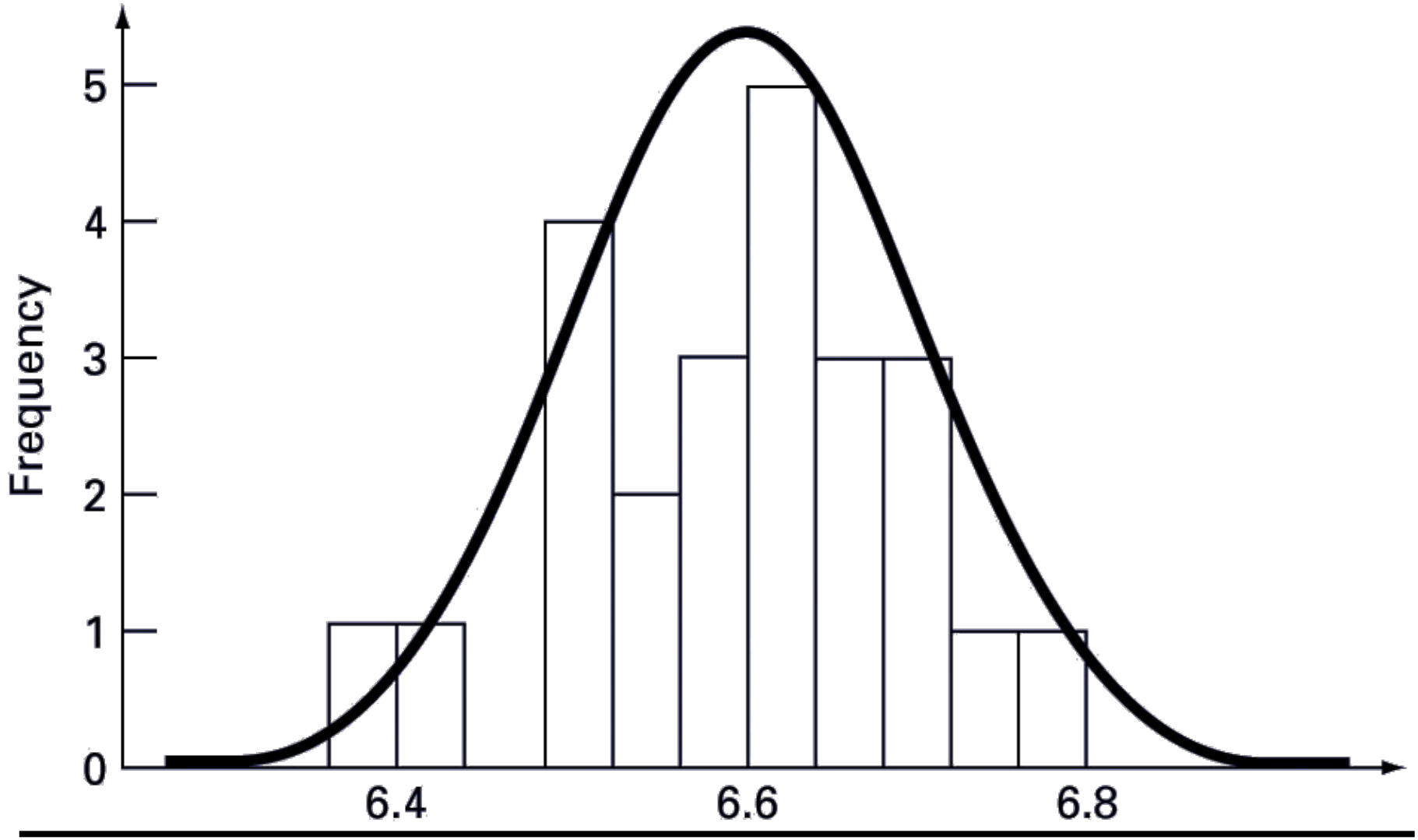
  where S_t is the sum of the squares of the data residuals: $$S_t = \sum (y_i - \bar{y})^2$$

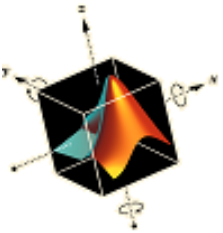
  and n-1 is referred to as the *degrees of freedom*.

- *Variance*: $$s_y^2 = \frac{\sum (y_i - \bar{y})^2}{n-1} = \frac{\sum y_i^2 - \left(\sum y_i\right)^2 / n}{n-1}$$

- *Coefficient of variation*: $$\text{c.v.} = \frac{s_y}{\bar{y}} \times 100\%$$

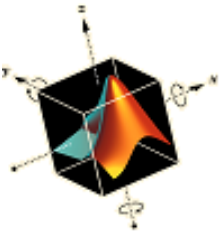
Revisión de estadística – distribución normal





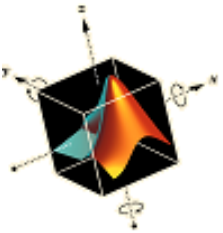
Estadística descriptiva en Matlab

- MATLAB has several built-in commands to compute and display descriptive statistics. Assuming some column vector *s*:
  - `mean(s), median(s), mode(s)`
    - Calculate the mean, median, and mode of *s*. `mode` is a part of the statistics toolbox.
  - `min(s), max(s)`
    - Calculate the minimum and maximum value in *s*.
  - `var(s), std(s)`
    - Calculate the variance and standard deviation of *s*
- Note - if a matrix is given, the statistics will be returned for each column.

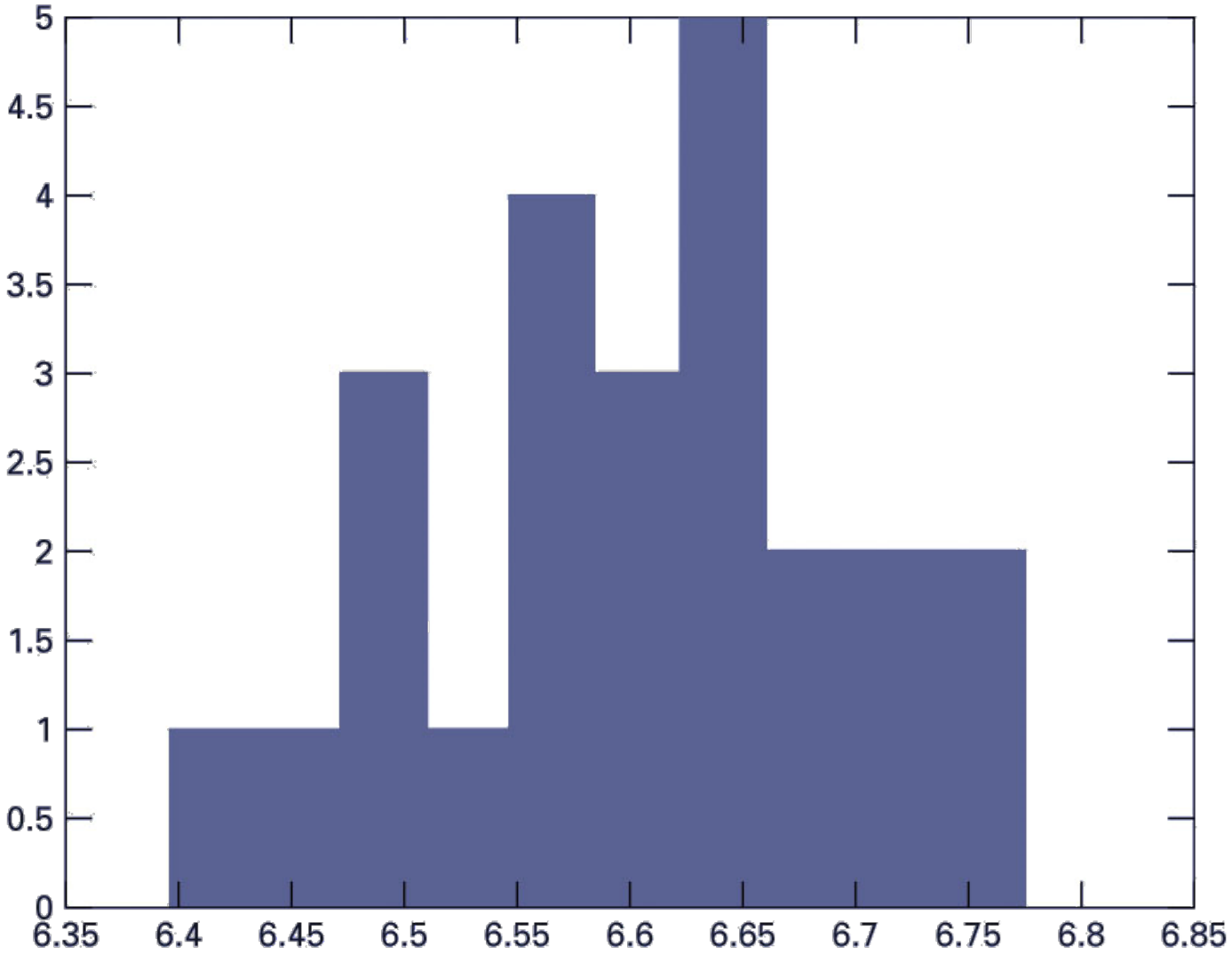


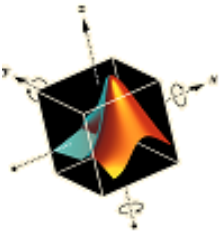
Histogramas en Matlab

- `[n, x] = hist(s, x)`
  - Determine the number of elements in each bin of data in *s*. *x* is a vector containing the center values of the bins.
- `[n, x] = hist(s, m)`
  - Determine the number of elements in each bin of data in *s* using *m* bins. *x* will contain the centers of the bins. The default case is *m*=10
- `hist(s, x)` or `hist(s, m)` or `hist(s)`
  - With no output arguments, `hist` will actually produce a histogram.



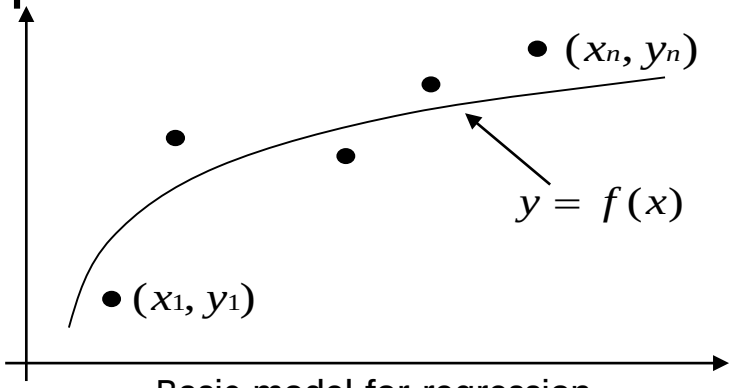
Histograma





¿Qué es regresión?

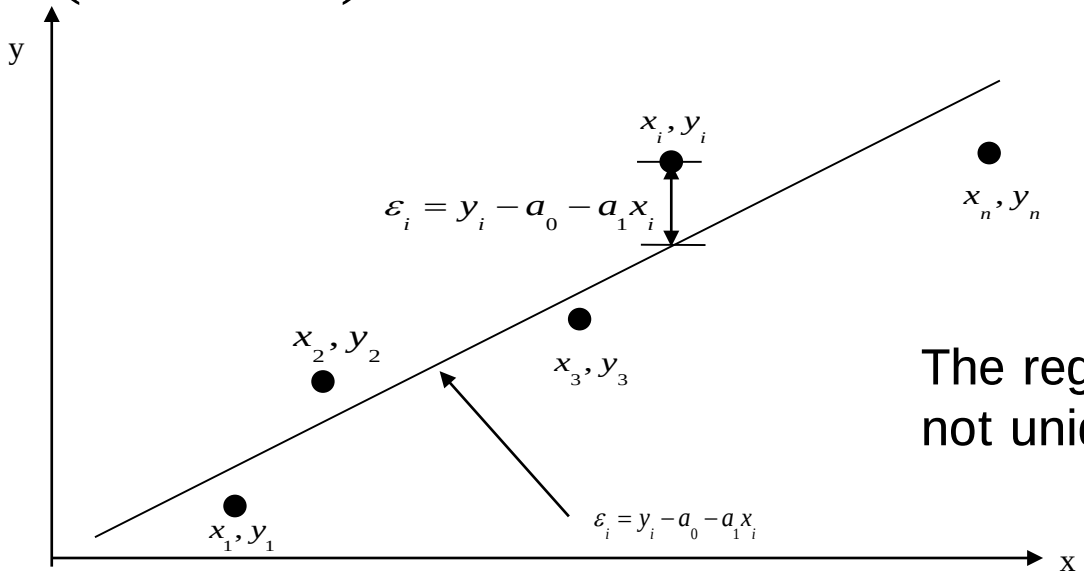
- Given n data points $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, best fit $y = f(x)$ to the data.
- The best fit is generally based on minimizing the sum of the square of the residuals, S_r
- Residual at a point is $\varepsilon_i = y_i - f(x_i)$
- Sum of the square of the residuals $S_r = \sum_{i=1}^{n} (y_i - f(x_i))^2$



Basic model for regression

Regresión lineal – criterio 1

- Given n data points $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, best fit $y = a_0 + a_1 x$ to the data.
- Does minimizing $\sum_{i=1}^{n} \varepsilon_i$ work as a criterion, where $\varepsilon_i = y_i - (a_0 + a_1 x_i)$

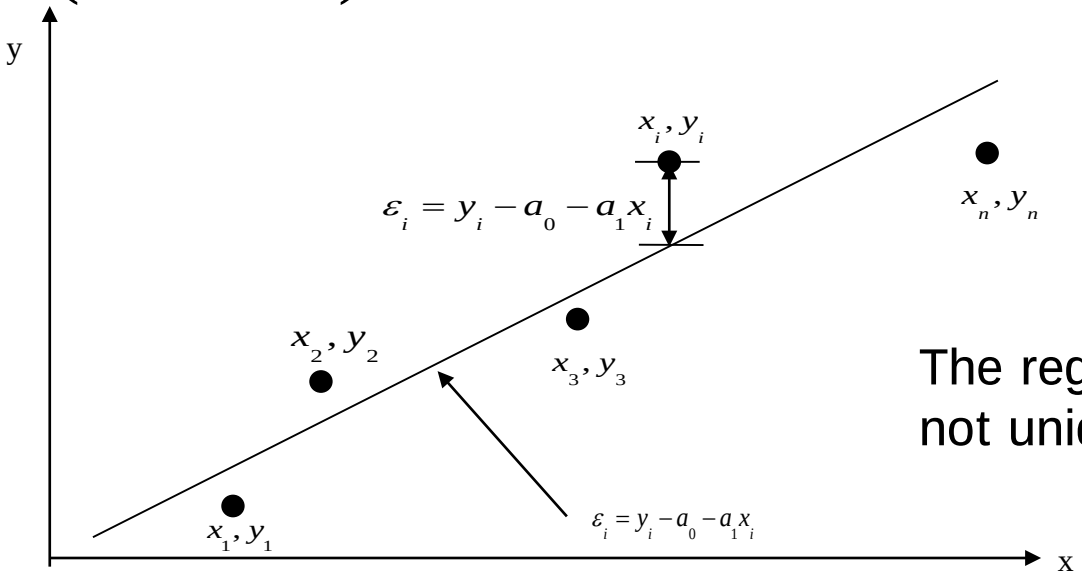


The regression model is not unique. Bad criterion.

Linear regression of y vs. x data showing residuals at a typical point, x_i.

Regresión lineal – criterio 2

- Given n data points $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, best fit $y = a_0 + a_1 x$ to the data.
- Does minimizing $\sum_{i=1}^{n} |\varepsilon_i|$ work as a criterion, where $\varepsilon_i = y_i - (a_0 + a_1 x_i)$

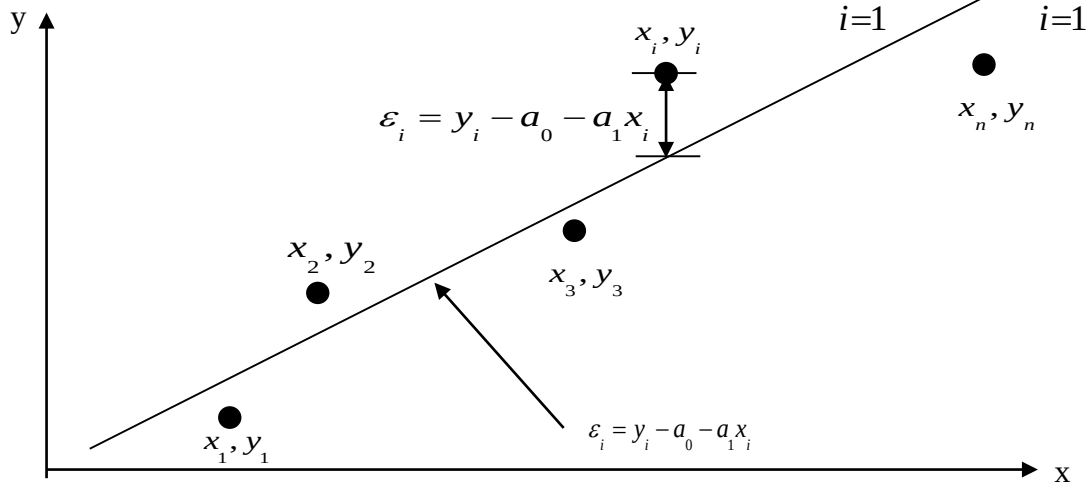


The regression model is not unique. Bad criterion.

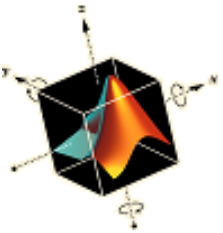
Linear regression of y vs. x data showing residuals at a typical point, x_i.

Regresión lineal – criterio mínimos cuadrados

- Given n data points $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, best fit $y = a_0 + a_1 x$ to the data.
- The least squares criterion minimizes the sum of the square of the residuals in the model, and also produces a unique line. $S_r = \sum_{i=1}^{n} \varepsilon_i^{\,2} = \sum_{i=1}^{n} (y_i - a_0 - a_1 x_i)^2$



Linear regression of y vs. x data showing residuals at a typical point, x_i.



Cálculo de coeficientes del modelo lineal

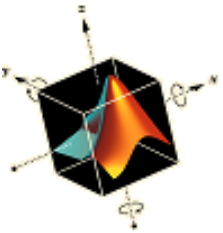
- Minimize the sum of the square of the residuals:

$$S_r = \sum_{i=1}^{n} \varepsilon_i^{\,2} = \sum_{i=1}^{n} \left(y_i - a_0 - a_1 x_i\right)^2$$

- To find a_0 and a_1 we minimize S_r with respect to a_1 and a_0

$$\frac{\partial S_r}{\partial a_0} = -2\sum_{i=1}^{n} \left(y_i - a_0 - a_1 x_i\right)(-1) = 0 \qquad \sum_{i=1}^{n} a_0 + \sum_{i=1}^{n} a_1 x_i = \sum_{i=1}^{n} y_i$$

$$\frac{\partial S_r}{\partial a_1} = -2\sum_{i=1}^{n} \left(y_i - a_0 - a_1 x_i\right)(-x_i) = 0 \qquad \sum_{i=1}^{n} a_0 x_i + \sum_{i=1}^{n} a_1 x_i^{\,2} = \sum_{i=1}^{n} y_i x_i$$



Cálculo de coeficientes del modelo lineal

- Solving for a_0 and a_1 directly yields,

$$a_1 = \frac{n\sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{n\sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2}$$

$$a_0 = \bar{y} - a_1 \bar{x}$$

$$a_0 = \frac{\sum_{i=1}^{n} x_i^2 \sum_{i=1}^{n} y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} x_i y_i}{n\sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2}$$

Regresión lineal - ejemplo

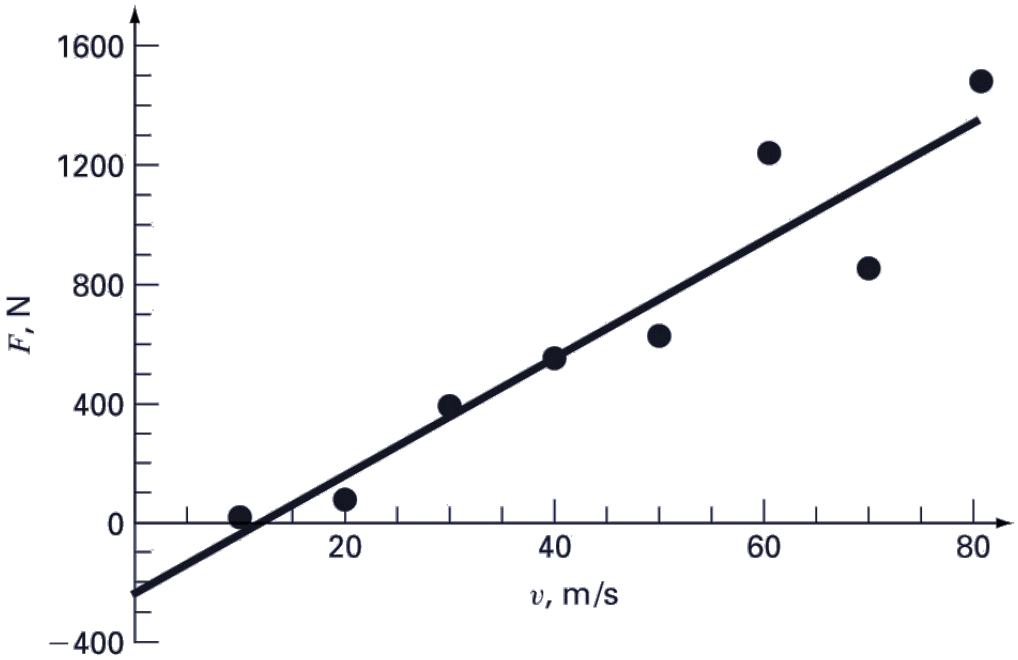
Experimental data for force (N) and velocity (m/s) from a wind tunnel experiment.

	V (m/s)	F (N)		
i	x_i	y_i	$(x_i)^2$	$x_i y_i$
1	10	25	100	250
2	20	70	400	1400
3	30	380	900	11400
4	40	550	1600	22000
5	50	610	2500	30500
6	60	1220	3600	73200
7	70	830	4900	58100
8	80	1450	6400	116000
Σ	360	5135	20400	312850

$$a_1 = \frac{n\sum x_i y_i - \sum x_i \sum y_i}{n\sum x_i^2 - \left(\sum x_i\right)^2} = \frac{8(312850) - (360)(5135)}{8(20400) - (360)^2} = 19.47024$$

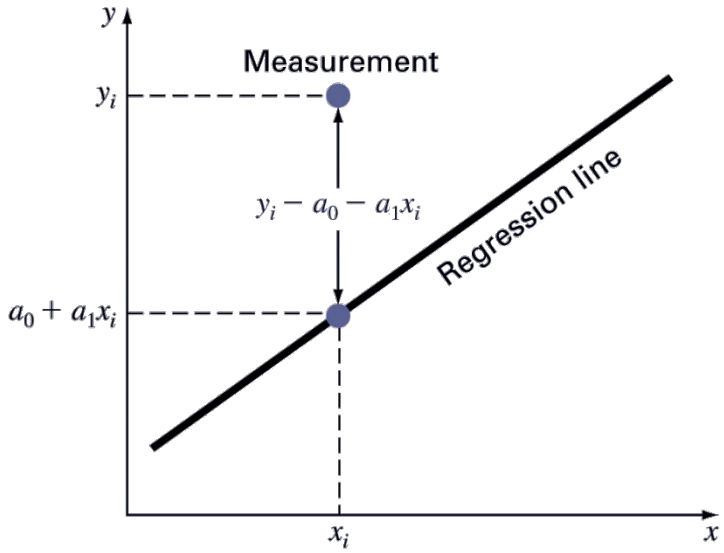
$$a_0 = \bar{y} - a_1 \bar{x} = 641.875 - 19.47024(45) = -234.2857$$

$$F_{est} = -234.2857 + 19.47024v$$



Cuantificación del error

- Recall for a straight line, the sum of the squares of the estimate residuals:



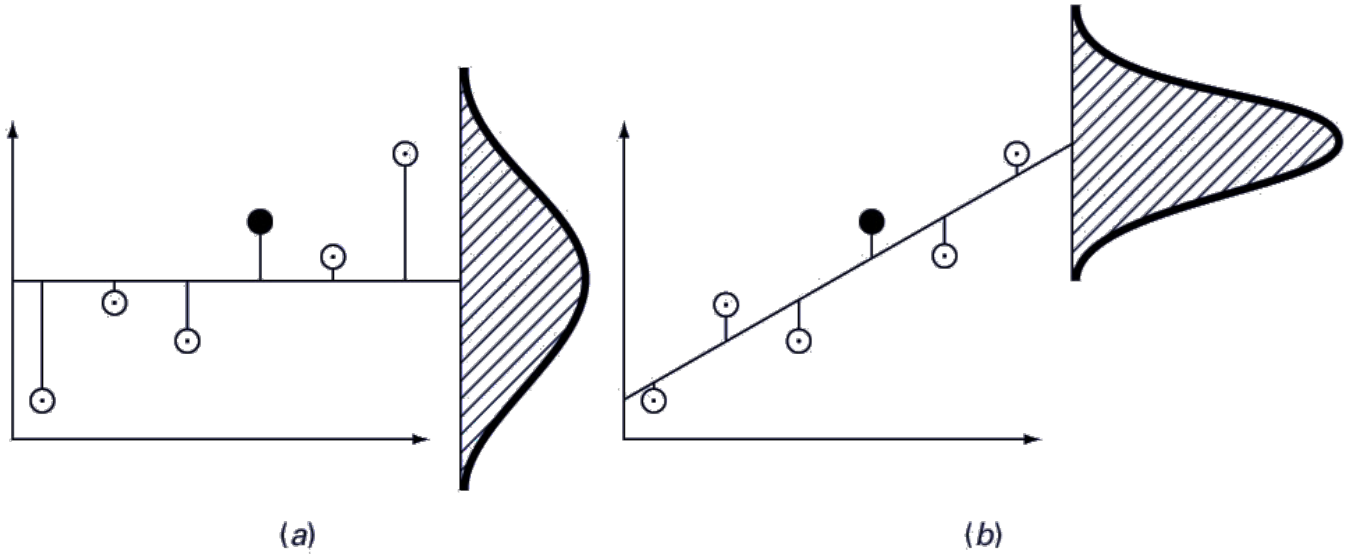
$$S_r = \sum_{i=1}^{n} e_i^2 = \sum_{i=1}^{n} (y_i - a_0 - a_1 x_i)^2$$

- Standard error of the estimate:

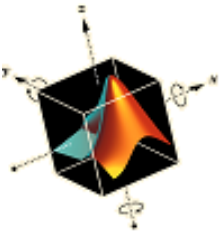
$$s_{y/x} = \sqrt{\frac{S_r}{n-2}}$$

Error estándar de la estimación

- Regression data showing (a) the spread of data around the mean of the dependent data and (b) the spread of the data around the best fit line:



- The reduction in spread represents the improvement due to linear regression.

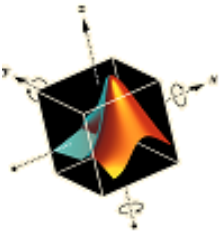


Coeficiente de determinación

- The *coefficient of determination* r^2 is the difference between the sum of the squares of the data residuals and the sum of the squares of the estimate residuals, normalized by the sum of the squares of the data residuals:

$$r^2 = \frac{S_t - S_r}{S_t}$$

- r^2 represents the percentage of the original uncertainty explained by the model.
- For a perfect fit, S_r=0 and r^2=1.
- If r^2=0, there is no improvement over simply picking the mean.
- If r^2<0, the model is *worse* than simply picking the mean!



Coeficiente de determinación - ejemplo

	V (m/s)	F (N)			
i	x_i	y_i	$a_0+a_1x_i$	$(y_i- \bar{y})^2$	$(y_i-a_0-a_1x_i)^2$
1	10	25	-39.58	380535	4171
2	20	70	155.12	327041	7245
3	30	380	349.82	68579	911
4	40	550	544.52	8441	30
5	50	610	739.23	1016	16699
6	60	1220	933.93	334229	81837
7	70	830	1128.63	35391	89180
8	80	1450	1323.33	653066	16044
Σ	360	5135		1808297	216118

$$F_{est} = -234.2857 + 19.47024v$$

$$S_t = \sum (y_i - \bar{y})^2 = 1808297$$

$$S_r = \sum (y_i - a_0 - a_1 x_i)^2 = 216118$$

$$s_y = \sqrt{\frac{1808297}{8-1}} = 508.26$$

$$s_{y/x} = \sqrt{\frac{216118}{8-2}} = 189.79$$

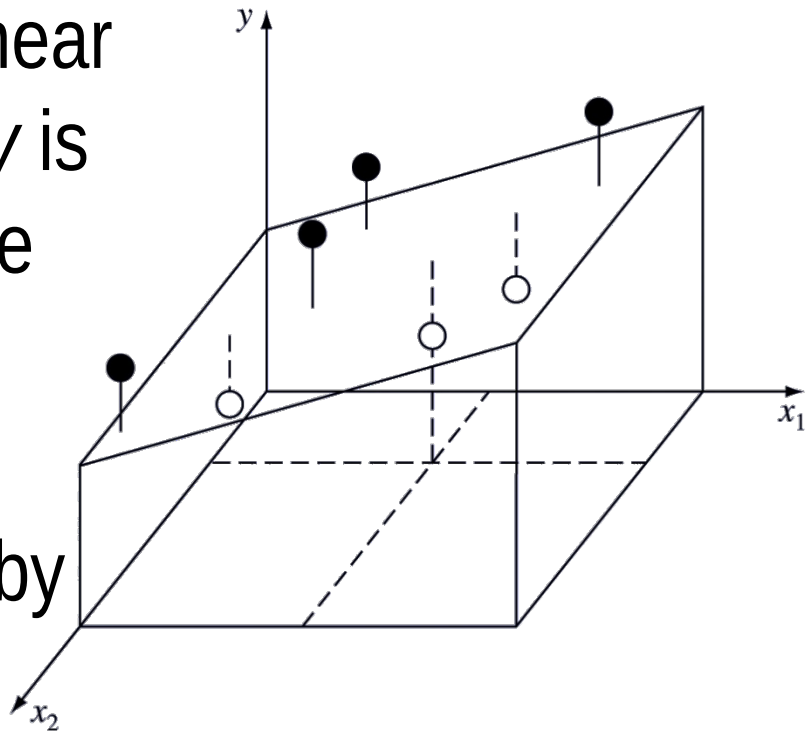
$$r^2 = \frac{1808297 - 216118}{1808297} = 0.8805$$

88.05% of the original uncertainty has been explained by the linear model

Regresión lineal múltiple

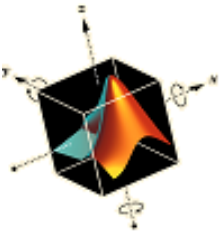
- Another useful extension of linear regression is the case where *y* is a linear function of two or more independent variables:

$$y = a_0 + a_1 x_1 + a_2 x_2 + \cdots a_m x_m$$



- Again, the best fit is obtained by minimizing the sum of the squares of the estimate residuals:

$$S_r = \sum_{i=1}^{n} e_i^2 = \sum_{i=1}^{n} \left(y_i - a_0 - a_1 x_{1,i} - a_2 x_{2,i} - \cdots a_m x_{m,i} \right)^2$$



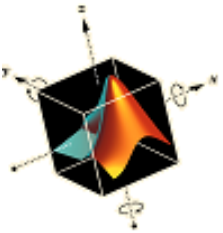
Generalización de regresión lineal por mínimos cuadrados

- Linear, polynomial, and multiple linear regression all belong to the general linear least-squares model:

$$y = a_0 z_0 + a_1 z_1 + a_2 z_2 + \cdots a_m z_m + e$$

where z_0, z_1, …, z_m are a set of m+1 *basis functions* and e is the error of the fit.

- The basis functions can be any function data but *cannot* contain any of the coefficients a_0, a_1, etc.

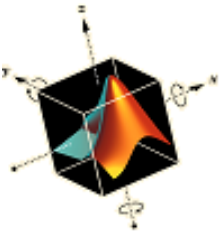


Generalización de regresión lineal por mínimos cuadrados

- The equation: $y = a_0 z_0 + a_1 z_1 + a_2 z_2 + \cdots a_m z_m + e$ can be re-written for each data point as a matrix equation: $\{y\} = [Z]\{a\} + \{e\}$ where {*y*} contains the dependent data, {*a*} contains the coefficients of the equation, {*e*} contains the error at each point, and [*Z*] is:

$$[Z] = \begin{bmatrix} z_{01} & z_{11} & \cdots & z_{m1} \\ z_{02} & z_{12} & \cdots & z_{m2} \\ \vdots & \vdots & \ddots & \vdots \\ z_{0n} & z_{1n} & \cdots & z_{mn} \end{bmatrix}$$

with z_{ji} representing the value of the j_{th} basis function calculated at the i_{th} point.



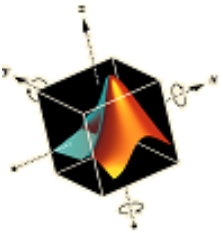
Generalización de regresión lineal por mínimos cuadrados

- Generally, [*Z*] is not a square matrix, so simple inversion cannot be used to solve for {*a*}. Instead the sum of the squares of the estimate residuals is minimized:

$$S_r = \sum_{i=1}^{n} e_i^2 = \sum_{i=1}^{n} \left(y_i - \sum_{j=0}^{m} a_j z_{ji} \right)^2$$

- The outcome of this minimization yields:

$$\left[[Z]^T [Z] \right] \{a\} = \left([Z]^T \{y\} \right)$$



Generalización de regresión lineal – ejemplo Matlab

- Given *x* and *y* data in columns, solve for the coefficients of the best fit line for $y=a_0+a_1x+a_2x^2$

```
Z = [ones(size(x)) x x.^2]
a = (Z'*Z)\(Z'*y)
```

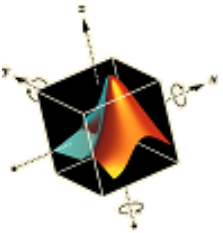
  - Note also that MATLAB's left-divide will automatically include the $[Z]^T$ terms if the matrix is not square, so

```
a = Z\y
```

    would work as well

- To calculate measures of fit:

```
St = sum((y-mean(y)).^2)
Sr = sum((y-Z*a).^2)
r2 = 1-Sr/St
syx = sqrt(Sr/(length(x)-length(a)))
```



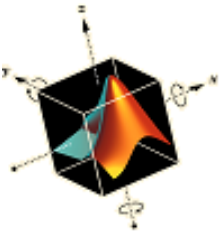
Relaciones no lineales

- Linear regression is predicated on the fact that the relationship between the dependent and independent variables is linear - this is not always the case.
- Three common examples are:

$$\text{exponential}: \quad y = \alpha_1 e^{\beta_1 x}$$

$$\text{power}: \quad y = \alpha_2 x^{\beta_2}$$

$$\text{saturation-growth-rate}: \quad y = \alpha_3 \frac{x}{\beta_3 + x}$$

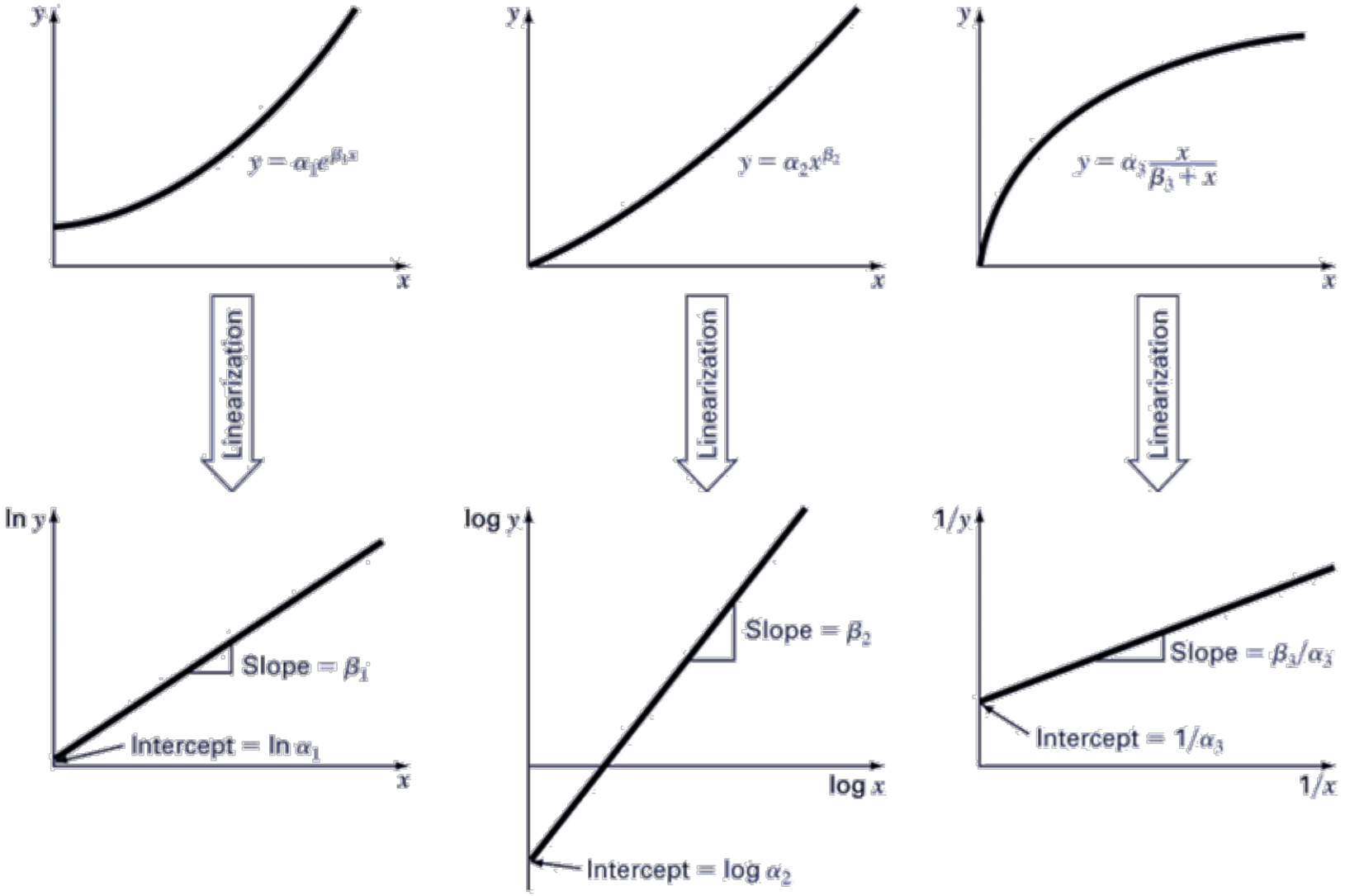


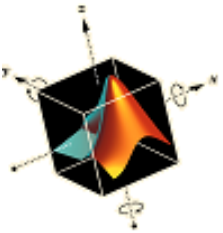
Linealización de relaciones no lineales

- One option for finding the coefficients for a nonlinear fit is to linearize it. For the three common models, this may involve taking logarithms or inversion:

Model	Nonlinear	Linearized
exponential :	$y = \alpha_1 e^{\beta_1 x}$	$\ln y = \ln \alpha_1 + \beta_1 x$
power :	$y = \alpha_2 x^{\beta_2}$	$\log y = \log \alpha_2 + \beta_2 \log x$
saturation - growth - rate :	$y = \alpha_3 \frac{x}{\beta_3 + x}$	$\frac{1}{y} = \frac{1}{\alpha_3} + \frac{\beta_3}{\alpha_3}\frac{1}{x}$

Ejemplos de transformación

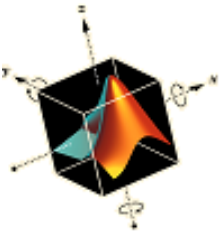




Funciones Matlab

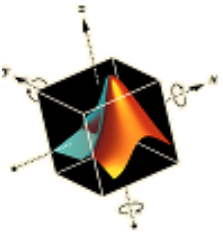
- MATLAB has a built-in function `polyfit` that fits a least-squares nth order polynomial to data:
  - `p = polyfit(x, y, n)`
    - *x*: independent data
    - *y*: dependent data
    - *n*: order of polynomial to fit
    - *p*: coefficients of polynomial

      $f(x)=p_1x^n+p_2x^{n-1}+\ldots+p_nx+p_{n+1}$
- MATLAB's `polyval` command can be used to compute a value using the coefficients.
  - `y = polyval(p, x)`



Regresión no lineal

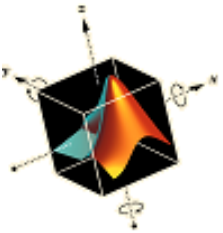
- Not all fits are linear equations of coefficients and basis functions.
- One method to handle this is to transform the variables and solve for the best fit of the transformed variables. There are two problems with this method:
  - Not all equations can be transformed easily or at all
  - The best fit line represents the best fit for the transformed variables, not the original variables.
- Another method is to perform nonlinear regression to directly determine the least-squares fit.



Regresión no lineal

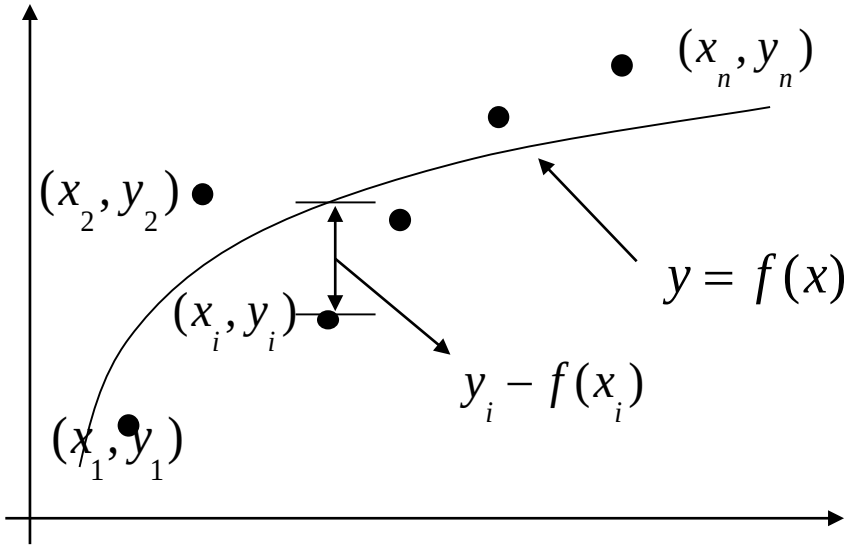
- Some popular nonlinear regression models:

1. Exponential model: $(y = ae^{bx})$
2. Power model: $(y = ax^{b})$
3. Saturation growth model: $\left(y = \frac{ax}{b + x} \right)$
4. Polynomial model: $(y = a_0 + a_1 x + \ldots + a_m x^{m})$

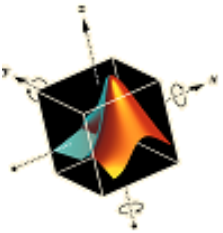


Regresión no lineal

- Given n data points $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ best fit $y = f(x)$ to the data, where $f(x)$ is a nonlinear function of x.

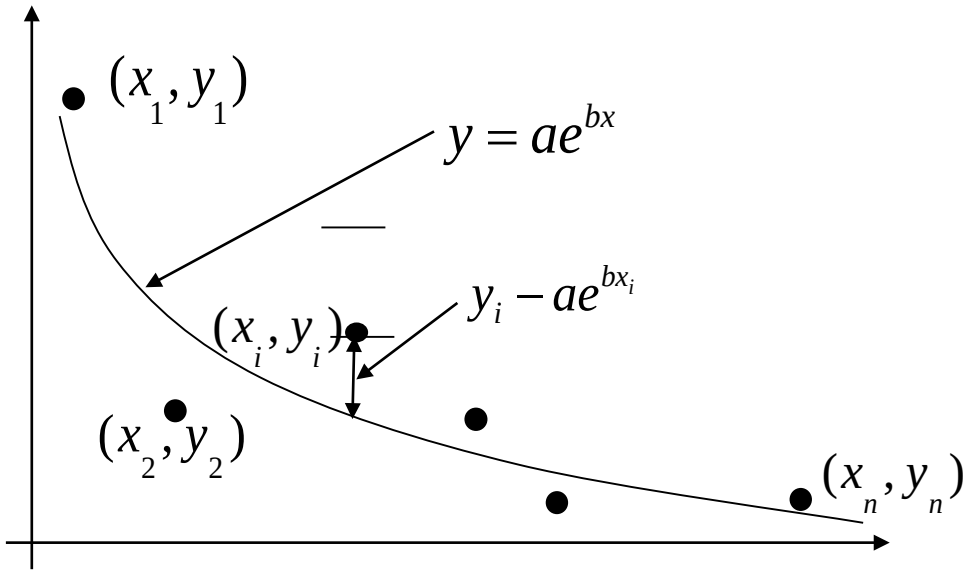


Nonlinear regression model for discrete y vs. x data

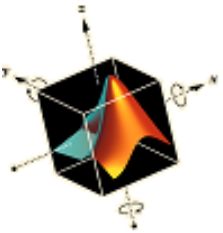


Regresión no lineal – modelo exponencial

- Given n data points $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ best fit $y = ae^{bx}$ to the data.



Exponential model of nonlinear regression for y vs. x data



Cálculo de constantes del modelo exponencial

The sum of the square of the residuals is defined as

$$S_r = \sum_{i=1}^{n} \left(y_i - a e^{bx_i} \right)^2$$

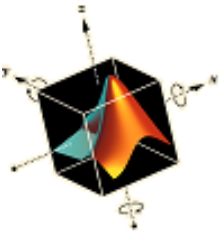
Differentiate with respect to a and b

$$\frac{\partial S_r}{\partial a} = \sum_{i=1}^{n} 2\left(y_i - a e^{bx_i} \right)\left(- e^{bx_i} \right) = 0$$

$$\frac{\partial S_r}{\partial b} = \sum_{i=1}^{n} 2\left(y_i - a e^{bx_i} \right)\left(- a x_i e^{bx_i} \right) = 0$$

Rewriting the equations, we obtain

$$- \sum_{i=1}^{n} y_i e^{bx_i} + a \sum_{i=1}^{n} e^{2bx_i} = 0 \qquad \sum_{i=1}^{n} y_i x_i e^{bx_i} - a \sum_{i=1}^{n} x_i e^{2bx_i} = 0$$



Cálculo de constantes del modelo exponencial

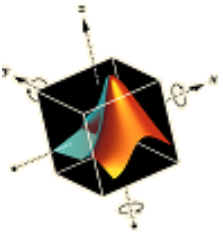
Solving the first equation for a yields

$$a = \frac{\sum_{i=1}^{n} y_i e^{bx_i}}{\sum_{i=1}^{n} e^{2bx_i}}$$

Substituting a back into the previous equation

$$\sum_{i=1}^{n} y_i x_i e^{bx_i} - \frac{\sum_{i=1}^{n} y_i e^{bx_i}}{\sum_{i=1}^{n} e^{2bx_i}} \sum_{i=1}^{n} x_i e^{2bx_i} = 0$$

The constant b can be found through numerical methods such as bisection method.



Modelo exponencial – ejemplo

Many patients get concerned when a test involves injection of a radioactive material. For example for scanning a gallbladder, a few drops of Technetium-99m isotope is used. Half of the techritium-99m would be gone in about 6 hours. It, however, takes about 24 hours for the radiation levels to reach what we are exposed to in day-to-day activities. Below is given the relative intensity of radiation as a function of time.

Relative intensity of radiation as a function of time

t(hrs)	0	1	3	5	7	9
γ	1.000	0.891	0.708	0.562	0.447	0.355

The relative intensity is related to time by the equation $\gamma = Ae^{\lambda t}$. Find:

a) The value of the regression constants A and λ.

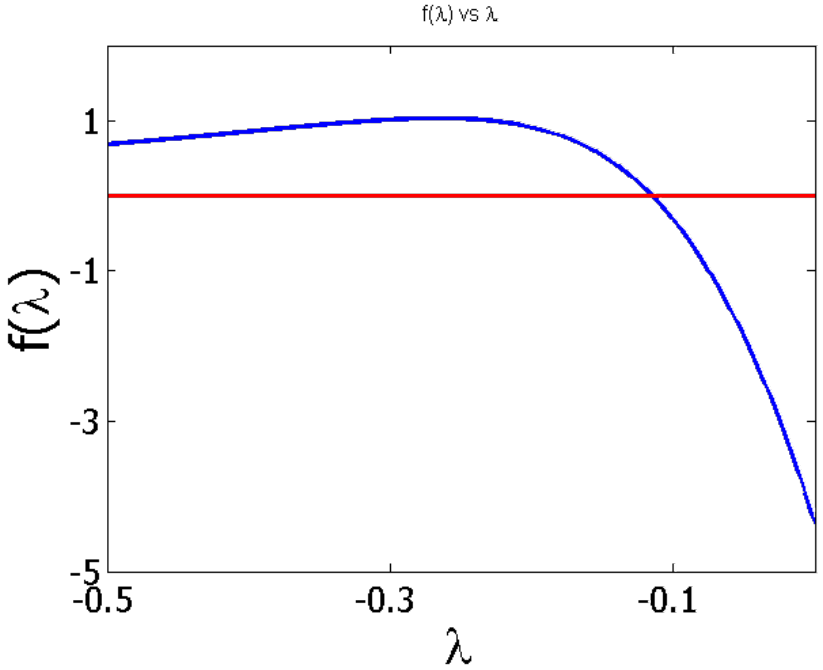
b) Radiation intensity after 24 hours

c) The half-life of Technium-99m

Modelo exponencial – ejemplo constantes del modelo

$$\gamma = Ae^{\lambda t}$$

The value of λ is found by solving the nonlinear equation by bisection method with initial values of λ = -0.12 and λ = -0.11.



$$f(\lambda) = \sum_{i=1}^{n} \gamma_i t_i e^{\lambda t_i} - \frac{\sum_{i=1}^{n} \gamma_i e^{\lambda t_i}}{\sum_{i=1}^{n} e^{2\lambda t_i}} \sum_{i=1}^{n} t_i e^{2\lambda t_i} = 0$$

$$A = \frac{\sum_{i=1}^{n} \gamma_i e^{\lambda t_i}}{\sum_{i=1}^{n} e^{2\lambda t_i}}$$

After 20 iterations (ε = 0.000008%)

$$\lambda = -0.11508$$

$$A = 0.99983$$

The exponential regression model is: $\gamma = 0.99983\, e^{-0.11508t}$

Modelo exponencial – ejemplo intensidad rel. a 24h y tiempo media vida

$$\gamma = 0.99983\, e^{-0.11508t}$$

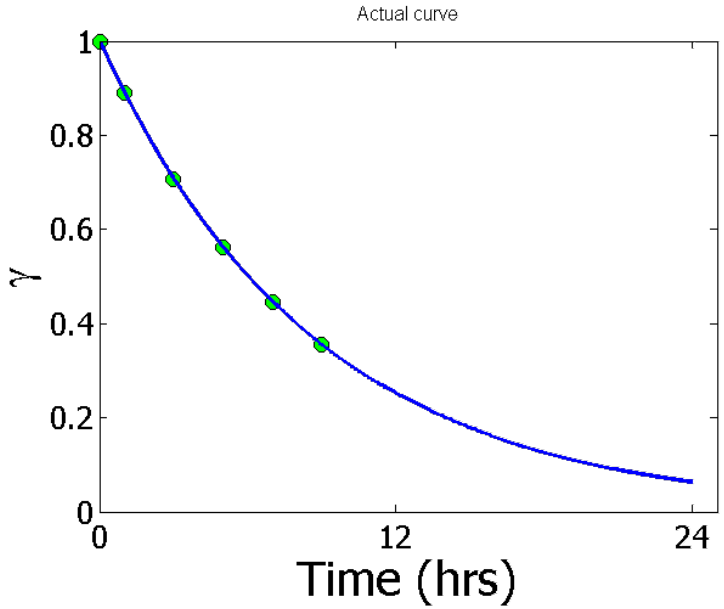
b) The relative intensity of radiation after 24 hours

$$\gamma = 0.99983\, e^{-0.11508(24)}$$
$$= 6.3160 \times 10^{-2}$$

This result implies that only

$$\frac{6.316 \times 10^{-2}}{0.9998} \times 100 = 6.317\%$$

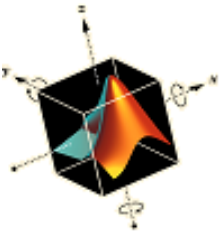
radioactive intensity is left after 24 hours.



c) half-life of Technium-99m is when $\gamma = \frac{1}{2}\gamma\Big|_{t=0}$

$$0.99983\, e^{-0.11508t} = \frac{1}{2} 0.99983\, e^{-0.11508(0)}$$

$$e^{-0.11508t} = 0.5 \qquad -0.11508t = \ln(0.5) \qquad t = 6.0232 \text{ hours}$$



Modelo exponencial – ejemplo linealizando el modelo

$$\gamma = Ae^{\lambda t}$$

Taking ln

$$\ln(\gamma) = \ln(A) + \lambda t$$

Assuming $z = \ln \gamma$

$$a_o = \ln(A)$$

$$a_1 = \lambda$$

We obtain $z = a_0 + a_1 t$

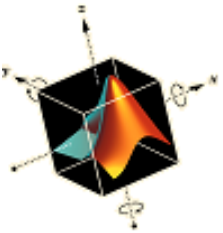
We can calculate

$$a_1 = \frac{n\sum_{i=1}^{n} t_i z_i - \sum_{i=1}^{n} t_i \sum_{i=1}^{n} z_i}{n\sum_{i=1}^{n} t_1^2 - \left(\sum_{i=1}^{n} t_i\right)^2}$$

$$a_0 = \bar{z} - a_1 \bar{t}$$

$$\lambda = a_1$$

$$A = e^{a_0}$$



Modelo exponencial – ejemplo linealizando el modelo

Summations for data linearization are

Table. Summation data for linearization of data model

i	t_i	γ_i	$z_i = \ln \gamma_i$	$t_i z_i$	t_i^2
1	0	1	0.00000	0.0000	0.0000
2	1	0.891	−0.11541	−0.11541	1.0000
3	3	0.708	−0.34531	−1.0359	9.0000
4	5	0.562	−0.57625	−2.8813	25.000
5	7	0.447	−0.80520	−5.6364	49.000
6	9	0.355	−1.0356	−9.3207	81.000
$\sum$	25.		−2.8778	−18.990	165.00

With $n = 6$

$$\sum_{i=1}^{6} t_i = 25.000$$

$$\sum_{i=1}^{6} z_i = -2.8778$$

$$\sum_{i=1}^{6} t_i z_i = -18.990$$

$$\sum_{i=1}^{6} t_i^2 = 165.00$$

$$a_1 = \frac{6(-18.990) - (25)(-2.8778)}{6(165.00) - (25)^2} = -0.11505$$

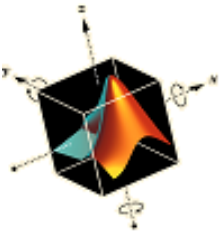
$$a_0 = \frac{-2.8778}{6} - (-0.11505)\frac{25}{6} = -2.6150 \times 10^{-4}$$

$$a_0 = \ln(A) \qquad A = e^{a_0}$$

$$= e^{-2.6150 \times 10^{-4}} = 0.99974$$

$$\lambda = a_1 = -0.11505$$

$$\gamma = 0.99974 \times e^{-0.11505t}$$



Modelo exponencial – ejemplo comparación

Comparison of exponential model with and without data linearization:

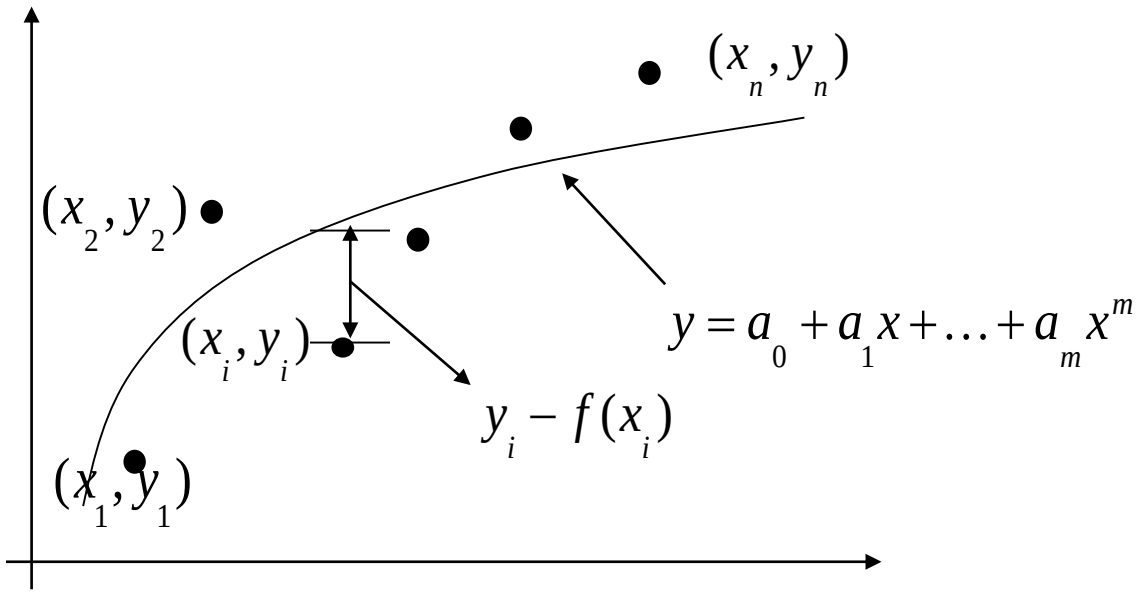
Comparison for exponential model with and without data linearization

	With data linearization	Without data linearization
A	0.99974	0.99983
λ	−0.11505	−0.11508
Half-Life (hrs)	6.0248	6.0232
Relative intensity after 24 hrs.	6.3200×10^{-2}	6.3160×10^{-2}

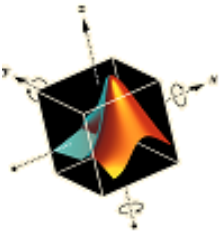
The values are very similar so data linearization was suitable to find the constants of the nonlinear exponential model in this case.

Regresión polinómica

- Given n data points $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ best fit $y = a_0 + a_1 x + \ldots + a_m x^m \quad (m < n)$ to the data.



Polynomial model for nonlinear regression of y vs. x data



Cálculo de constantes del modelo polinómico

The residual at each data point is given by

$$E_i = y_i - a_0 - a_1 x_i - \ldots - a_m x_i^m$$

The sum of the square of the residuals then is

$$S_r = \sum_{i=1}^{n} E_i^2 = \sum_{i=1}^{n} \left(y_i - a_0 - a_1 x_i - \ldots - a_m x_i^m \right)^2$$

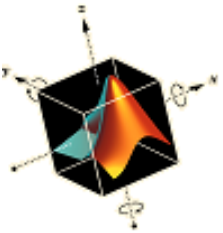
To find the constants of the polynomial model, we set the derivatives with respect to a_i where $i = 1, \ldots m$, equal to zero.

Standard error:

$$s_{y/x} = \sqrt{\frac{S_r}{n-(m+1)}}$$

Coefficient of determination

$$r^2 = \frac{S_t - S_r}{S_t}$$



Cálculo de constantes del modelo polinómico

$$\frac{\partial S_r}{\partial a_0} = \sum_{i=1}^{n} 2.\left(y_i - a_0 - a_1 x_i - \ldots - a_m x_i^m\right)(-1) = 0$$

$$\frac{\partial S_r}{\partial a_1} = \sum_{i=1}^{n} 2.\left(y_i - a_0 - a_1 x_i - \ldots - a_m x_i^m\right)(-x_i) = 0$$

$$\vdots \qquad \vdots \qquad \vdots \qquad \vdots$$

$$\frac{\partial S_r}{\partial a_m} = \sum_{i=1}^{n} 2.\left(y_i - a_0 - a_1 x_i - \ldots - a_m x_i^m\right)(-x_i^m) = 0$$

equations in matrix form

$$\begin{bmatrix} n & \left(\sum_{i=1}^{n} x_i\right) & \ldots & \left(\sum_{i=1}^{n} x_i^m\right) \\ \left(\sum_{i=1}^{n} x_i\right) & \left(\sum_{i=1}^{n} x_i^2\right) & \ldots & \left(\sum_{i=1}^{n} x_i^{m+1}\right) \\ \ldots & \ldots & \ldots & \ldots \\ \left(\sum_{i=1}^{n} x_i^m\right) & \left(\sum_{i=1}^{n} x_i^{m+1}\right) & \ldots & \left(\sum_{i=1}^{n} x_i^{2m}\right) \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \ldots \\ a_m \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n} y_i \\ \sum_{i=1}^{n} x_i y_i \\ \ldots \\ \sum_{i=1}^{n} x_i^m y_i \end{bmatrix}$$

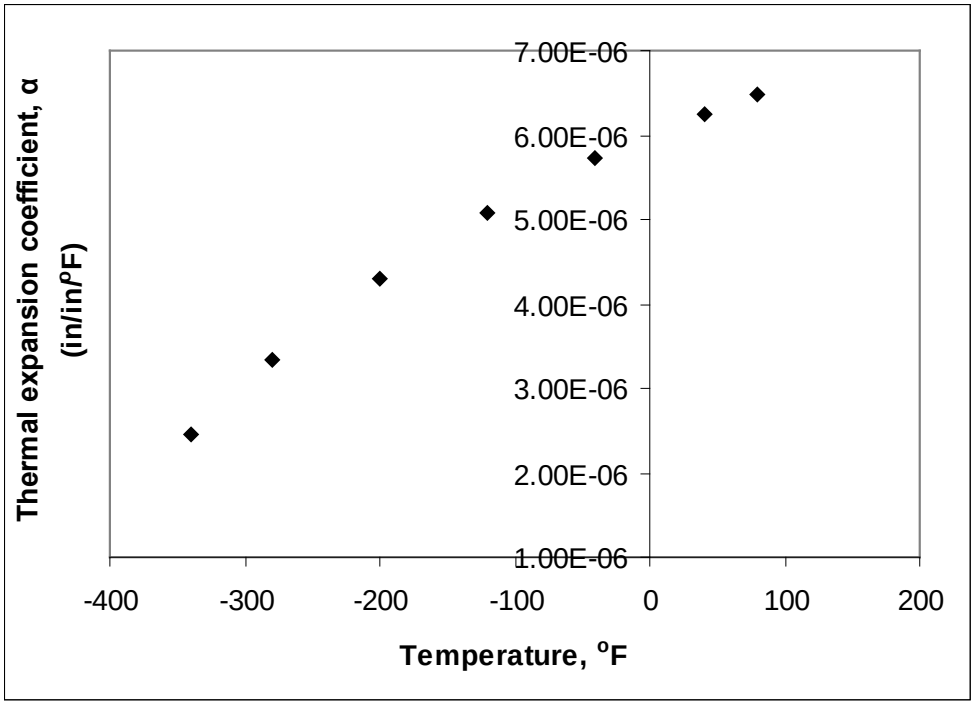
The above equations are then solved for $a_0, a_1, \ldots, a_m$

Modelo polinómico – ejemplo

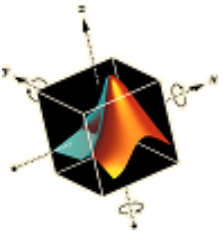
Regress the thermal expansion coefficient vs. temperature data to a second order polynomial of a steel cylinder

Data points for temperature vs α

Temperature, T (^oF)	Coefficient of thermal expansion, α (in/in/^oF)
80	6.47×10^{-6}
40	6.24×10^{-6}
−40	5.72×10^{-6}
−120	5.09×10^{-6}
−200	4.30×10^{-6}
−280	3.33×10^{-6}
−340	2.45×10^{-6}



Data points for thermal expansion coefficient vs temperature.



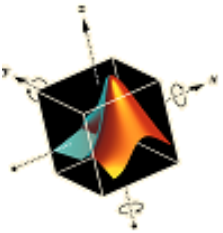
Modelo polinómico – ejemplo

We are to fit the data to the polynomial regression model

$$\alpha = a_0 + a_1 T + a_2 T^2$$

The coefficients a_0, a_1, a_2 are found by differentiating the sum of the square of the residuals with respect to each variable and setting the values equal to zero to obtain

$$\begin{bmatrix} n & \left(\sum_{i=1}^{n} T_i\right) & \left(\sum_{i=1}^{n} T_i^2\right) \\ \left(\sum_{i=1}^{n} T_i\right) & \left(\sum_{i=1}^{n} T_i^2\right) & \left(\sum_{i=1}^{n} T_i^3\right) \\ \left(\sum_{i=1}^{n} T_i^2\right) & \left(\sum_{i=1}^{n} T_i^3\right) & \left(\sum_{i=1}^{n} T_i^4\right) \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n} \alpha_i \\ \sum_{i=1}^{n} T_i \alpha_i \\ \sum_{i=1}^{n} T_i^2 \alpha_i \end{bmatrix}$$



Modelo polinómico – ejemplo

Data points for temperature vs α

Temperature, T (°F)	**Coefficient of thermal expansion, α** (in/in/°F)
80	6.47×10^{-6}
40	6.24×10^{-6}
−40	5.72×10^{-6}
−120	5.09×10^{-6}
−200	4.30×10^{-6}
−280	3.33×10^{-6}
−340	2.45×10^{-6}

The necessary summations are as follows

$$\sum_{i=1}^{7} T_i^2 = 2.5580\times10^{5}$$

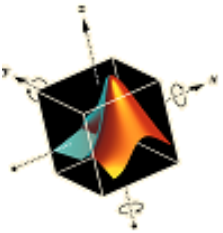
$$\sum_{i=1}^{7} T_i^3 = -7.0472\times10^{7}$$

$$\sum_{i=1}^{7} T_i^4 = 2.1363\times10^{10}$$

$$\sum_{i=1}^{7} \alpha_i = 3.3600\times10^{-5}$$

$$\sum_{i=1}^{7} T_i\alpha_i = -2.6978\times10^{-3}$$

$$\sum_{i=1}^{7} T_i^2\alpha_i = 8.5013\times10^{-1}$$



Modelo polinómico – ejemplo

Using these summations, we can now calculate a_0, a_1, a_2

$$\begin{bmatrix} 7.0000 & -8.6000\times10^{2} & 2.5800\times10^{5} \\ -8.600\times10^{2} & 2.5800\times10^{5} & -7.0472\times10^{7} \\ 2.5800\times10^{5} & -7.0472\times10^{7} & 2.1363\times10^{10} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} 3.3600\times10^{-5} \\ -2.6978\times10^{-3} \\ 8.5013\times10^{-1} \end{bmatrix}$$

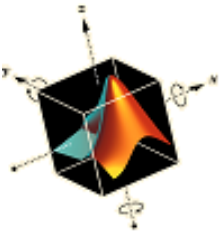
Solving the above system of simultaneous linear equations we have

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} 6.0217\times10^{-6} \\ 6.2782\times10^{-9} \\ -1.2218\times10^{-11} \end{bmatrix}$$

The polynomial regression model is then

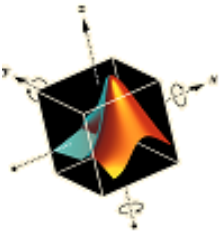
$$\alpha = a_0 + a_1 T + a_2 T^2$$

$$= 6.0217\times10^{-6} + 6.2782\times10^{-9}\mathrm{T} - 1.2218\times10^{-11}\mathrm{T}^2$$



Regresión no lineal en Matlab

- To perform nonlinear regression in MATLAB, write a function that returns the sum of the squares of the estimate residuals for a fit and then use MATLAB's `fminsearch` function to find the values of the coefficients where a minimum occurs.
- The arguments to the function to compute S_r should be the coefficients, the independent variables, and the dependent variables.



Regresión no lineal en Matlab - ejemplo

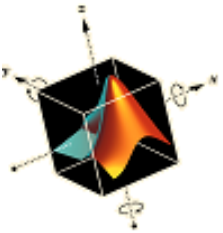
- Given dependent force data F for independent velocity data v, determine the coefficients for the fit:
- First - write a function called `fSSR.m` containing the following:

```
function f = fSSR(a, xm, ym)
yp = a(1)*xm.^a(2);
f = sum((ym-yp).^2);
```

- Then, use fminsearch in the command window to obtain the values of a that minimize `fSSR`:

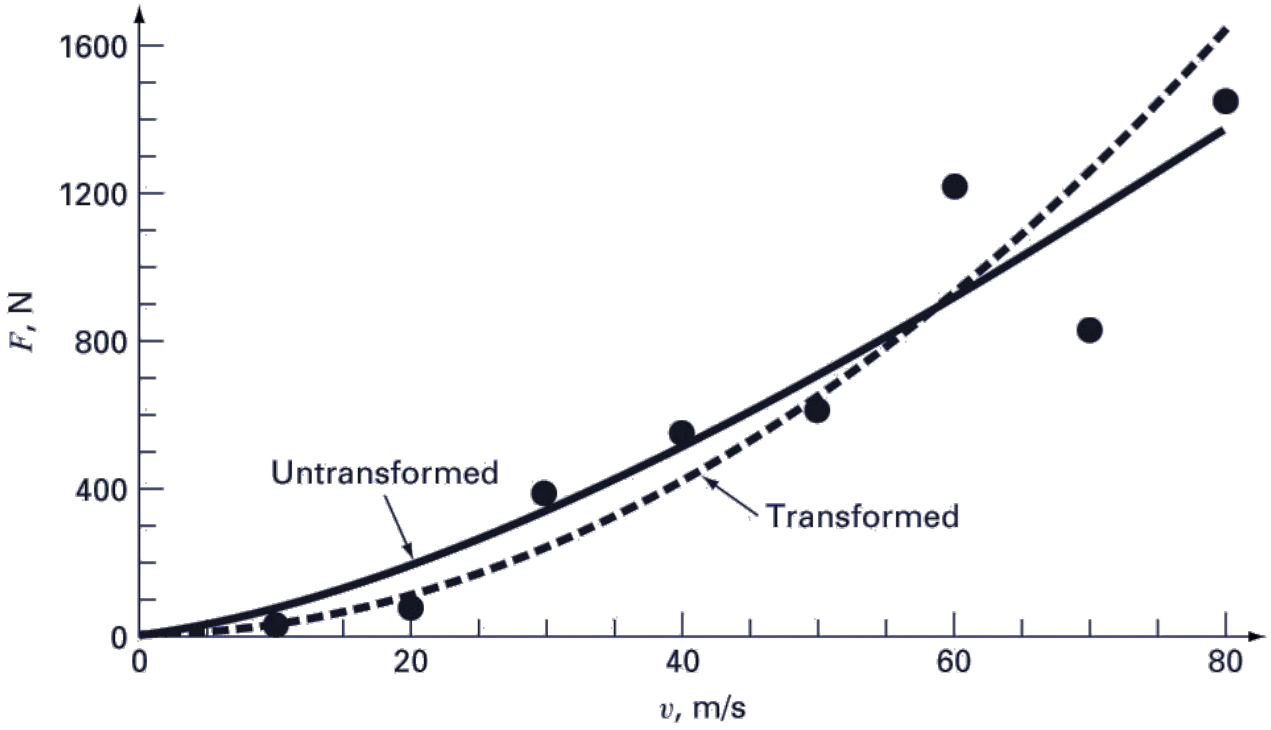
```
a = fminsearch(@fSSR,[1,1],[],v,F)
```

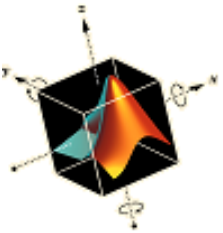
where [1, 1] is an initial guess for the [a0, a1] vector, [] is a placeholder for the options



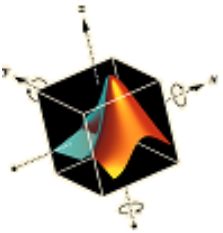
Regresión no lineal en Matlab - ejemplo

- The resulting coefficients will produce the largest r^2 for the data and may be different from the coefficients produced by a transformation:





Raíces de ecuaciones

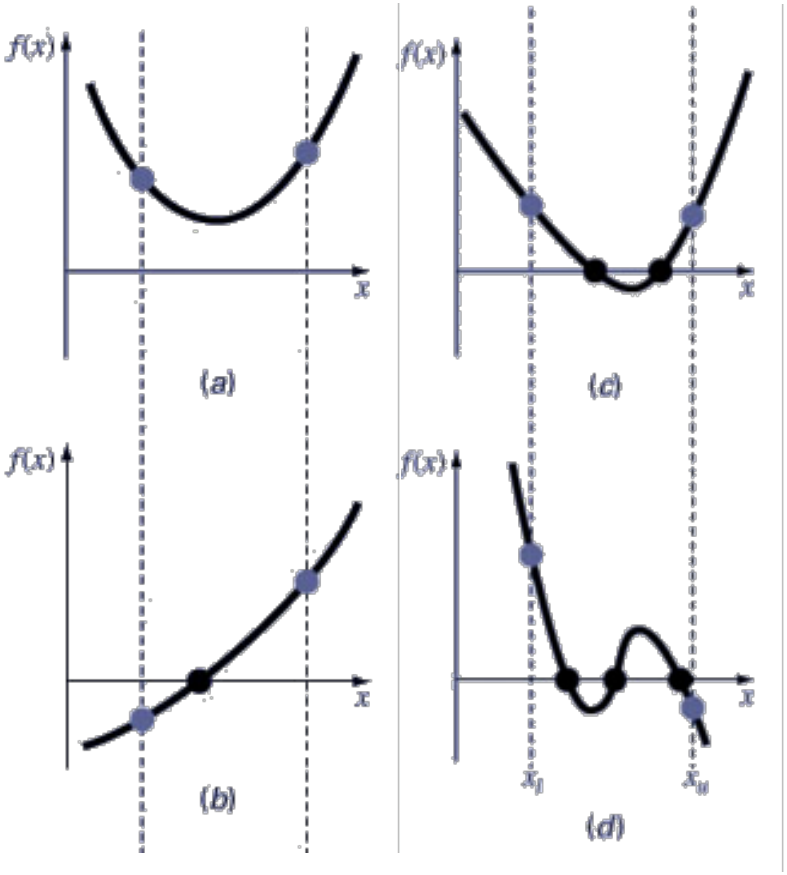


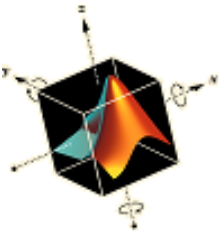
Raíces

- “Roots” problems occur when some function *f* can be written in terms of one or more dependent variables *x*, where the solutions to *f(x)=0* yields the solution to the problem.
- These problems often occur when a design problem presents an implicit equation for a required parameter.

Raíces

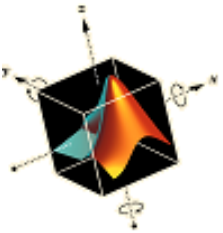
- A simple method for obtaining the estimate of the root of the equation *f(x)*=0 is to make a plot of the function and observe where it crosses the *x*-axis.
- Graphing the function can also indicate where roots may be and where some root-finding methods may fail:
  a) Same sign, no roots
  b) Different sign, one root
  c) Same sign, two roots
  d) Different sign, three roots





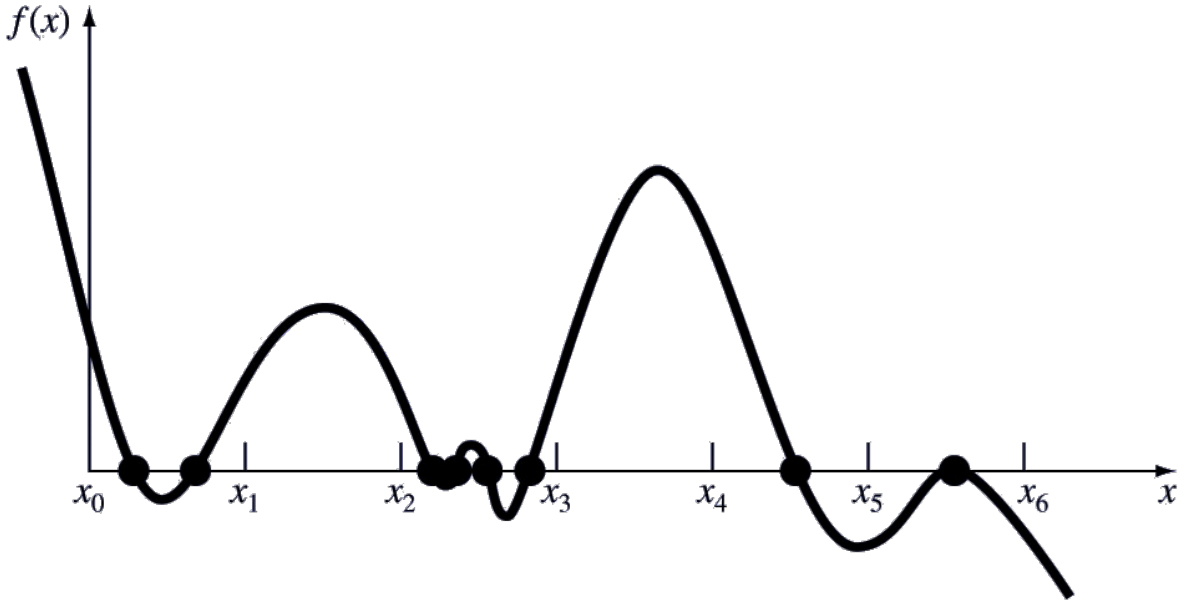
Métodos cerrados

- *Bracketing methods* are based on making two initial guesses that “bracket” the root - that is, are on either side of the root.
- Brackets are formed by finding two guesses x_l and x_u where the sign of the function changes; that is, where $f(x_l)\, f(x_u) < 0$
- The *incremental search* method tests the value of the function at evenly spaced intervals and finds brackets by identifying function sign changes between neighboring points.



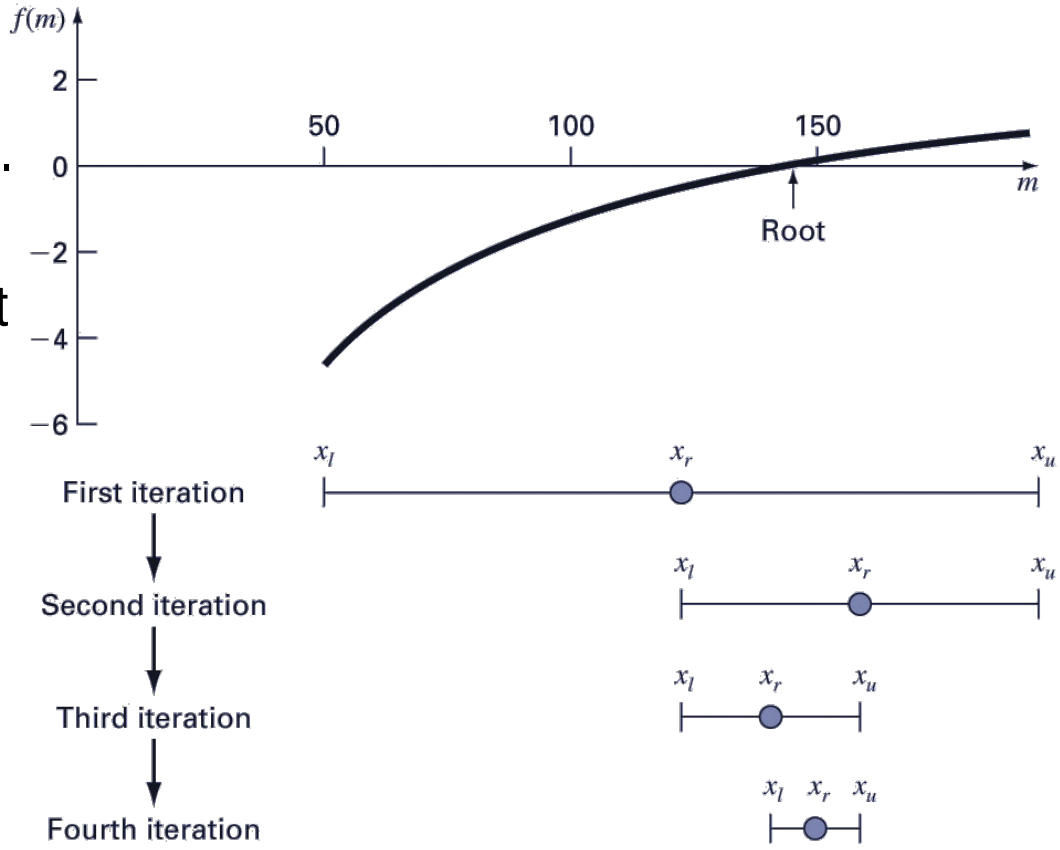
Desventajas de la búsqueda incremental

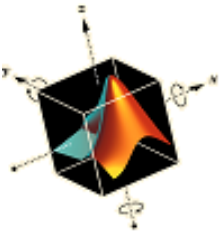
- If the spacing between the points of an incremental search are too far apart, brackets may be missed due to capturing an even number of roots within two points.
- Incremental searches cannot find brackets containing even-multiplicity roots regardless of spacing.



Método de bisección

- The *bisection method* is a variation of the incremental search method in which the interval is always divided in half.
- If a function changes sign over an interval, the function value at the midpoint is evaluated.
- The location of the root is then determined as lying within the subinterval where the sign change occurs.
- The absolute error is reduced by a factor of 2 for each iteration.



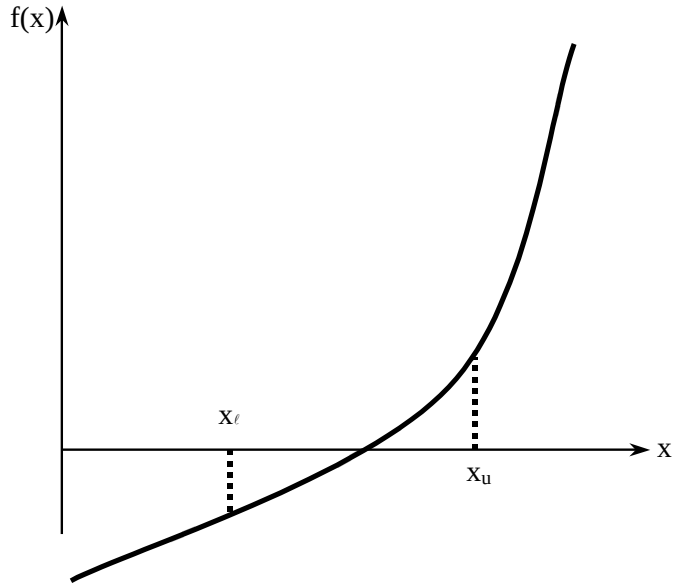


Error del método de bisección

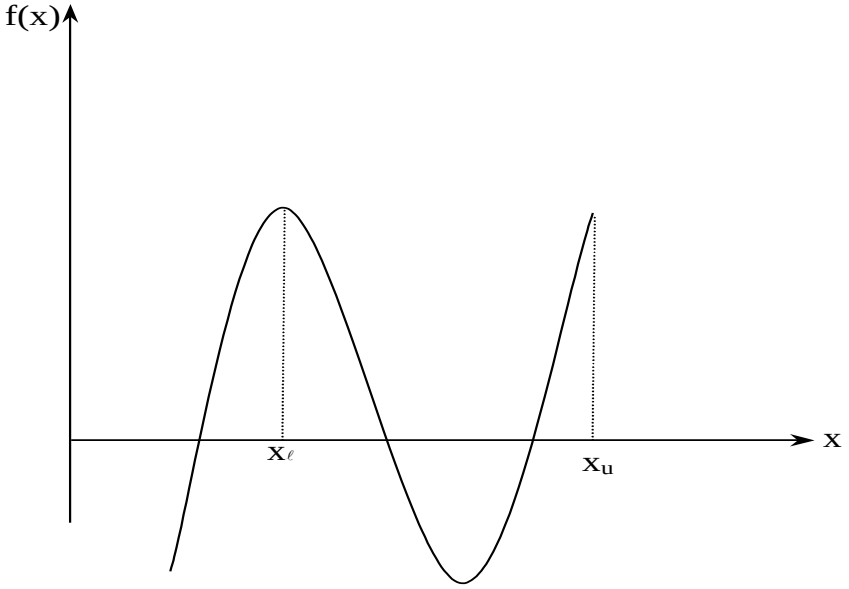
- The absolute error of the bisection method is solely dependent on the absolute error at the start of the process (the space between the two guesses) and the number of iterations: $$E_a^n = \frac{\Delta x^0}{2^n}$$
- The required number of iterations to obtain a particular absolute error can be calculated based on the initial guesses: $$n = \log_2\left(\frac{\Delta x^0}{E_{a,d}}\right)$$

Bases del método de bisección

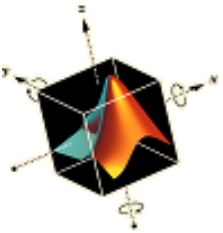
Theorem: An equation f(x)=0, where f(x) is a real continuous function, has at least one root between x_l and x_u if $f(x_l)\, f(x_u) < 0$.



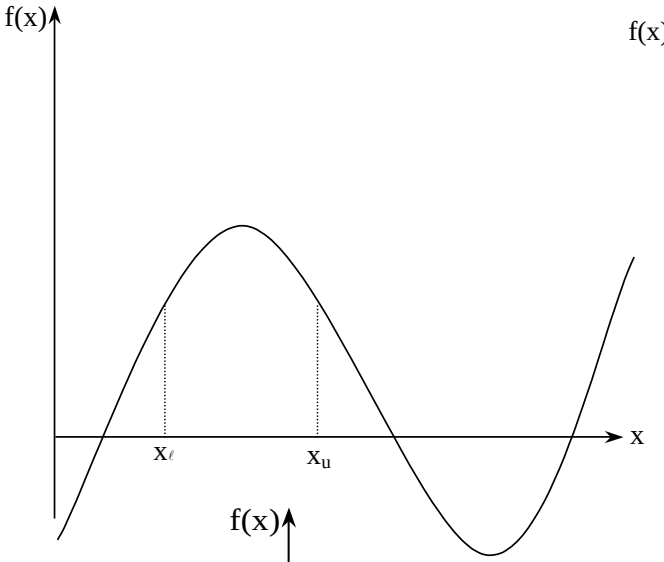
At least one root exists between the two points if the function is real, continuous, and changes sign.

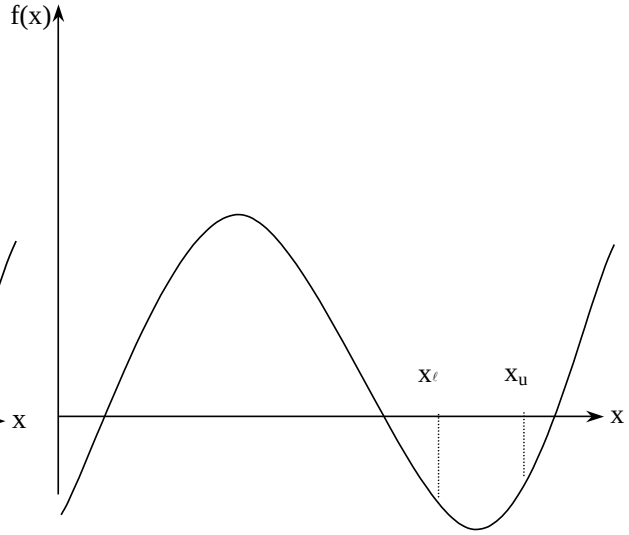


If function does not change sign between two points, roots of the equation may still exist between the two points.

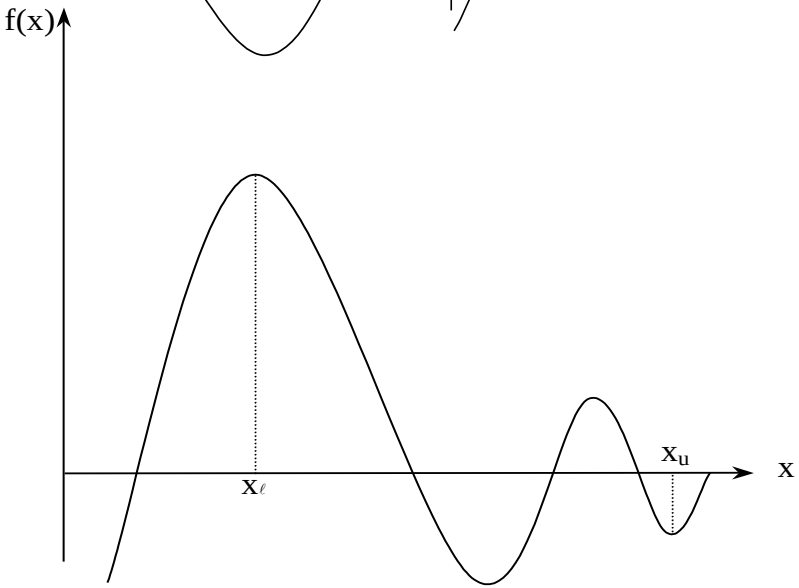


Bases del método de bisección

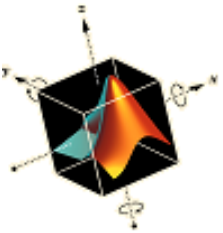




If function does not change sign between two points, there may not be any roots for the equation between the two points.



If the function changes sign between two points, more than one root for the equation may exist between the two points.



Algoritmo del método de bisección

- Step 1. Choose xl and xu as two guesses for the root such that f(xl) f(xu) < 0, or in other words, f(x) changes sign between xl and xu.
- Step 2. Estimate the root, xm of the equation f (x) = 0 as the mid point between xl and xu as $x_m = \frac{x_\ell + x_u}{2}$
- Step 3. Now check the following

a) If $f(x_l) f(x_m) < 0$, then the root lies between x_ℓ and x_m; then $x_\ell = x_\ell$; $x_u = x_m$.

b) If $f(x_l) f(x_m) > 0$, then the root lies between x_m and x_u; then $x_\ell = x_m$; $x_u = x_u$.

c) If $f(x_l) f(x_m) = 0$, then the root is x_m. Stop the algorithm if this is true.

- Step 4. Find the new estimate of the root $x_m = \frac{x_\ell + x_u}{2}$

Find the absolute relative approximate error

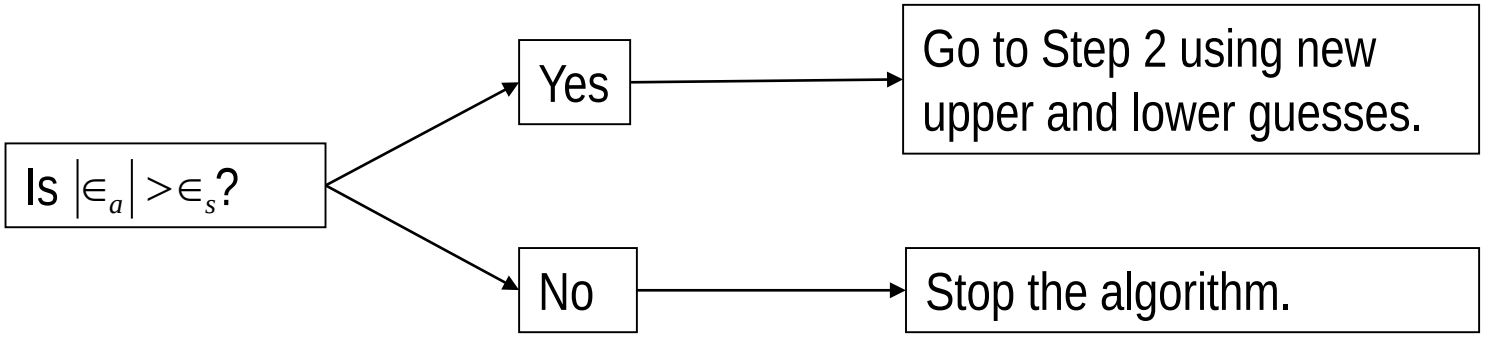
$$\left|\epsilon_a\right| = \left|\frac{x_m^{new} - x_m^{old}}{x_m^{new}}\right| \times 100$$

x_m^{old} = previous estimate of root

x_m^{new} = current estimate of root

Algoritmo del método de bisección

- **Step 5.** Compare the absolute relative approximate error $|\epsilon_a|$ with the pre-specified error tolerance ϵ_s.



- Note one should also check whether the number of iterations is more than the maximum number of iterations allowed. If so, one needs to terminate the algorithm and notify the user about it.

Algoritmo del método de bisección - ejemplo

- The floating ball has a specific gravity of 0.6 and has a radius of 5.5 cm. You are asked to find the depth to which the ball is submerged when floating in water.

  The equation that gives the depth *x* to which the ball is submerged under water is given by

$$x^3 - 0.165x^2 + 3.993\times10^{-4} = 0$$

a) Use the bisection method of finding roots of equations to find the depth *x* to which the ball is submerged under water. Conduct three iterations to estimate the root of the above equation.

b) Find the absolute relative approximate error at the end of each iteration, and the number of significant digits at least correct at the end of each iteration.

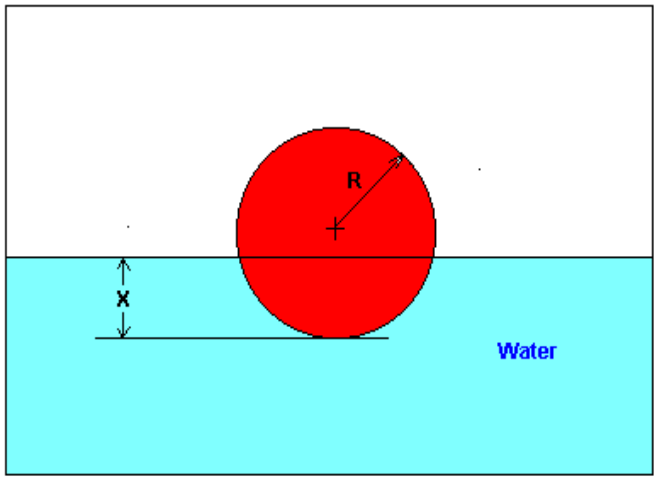


Diagram of the floating ball

Algoritmo del método de bisección - ejemplo

Solution

From the physics of the problem, the ball would be submerged between $x = 0$ and $x = 2R$, where R = radius of the ball, that is

$$0 \le x \le 2R$$

$$0 \le x \le 2(0.055)$$

$$0 \le x \le 0.11$$

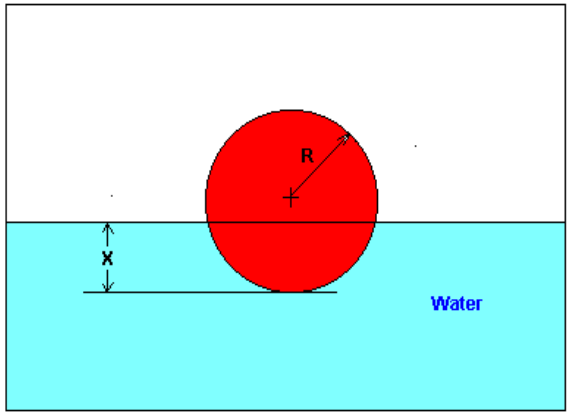
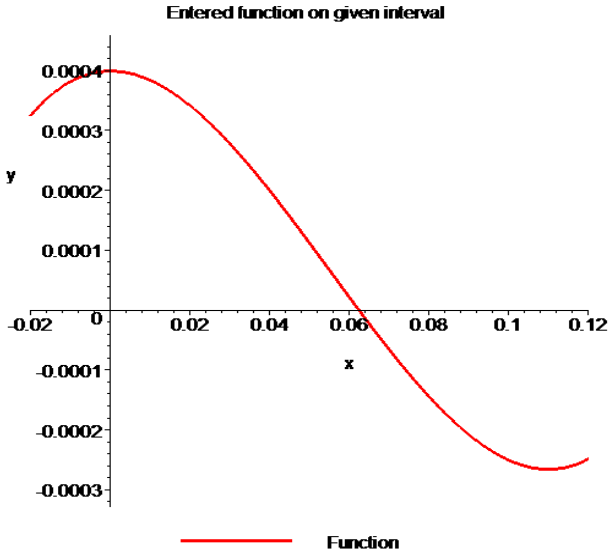


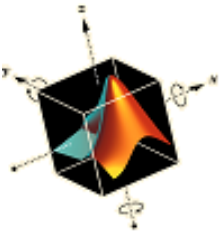
Diagram of the floating ball

To aid in the understanding of how this method works to find the root of an equation, the graph of f(x) is shown to the right, where

$$f(x) = x^3 - 0.165x^2 + 3.993\times10^{-4}$$



Graph of the function f(x)



Algoritmo del método de bisección - ejemplo

Let us assume $x_\ell = 0.00,\ x_u = 0.11$

Check if the function changes sign between x_ℓ and x_u.

$$f(x_l) = f(0) = (0)^3 - 0.165(0)^2 + 3.993\times10^{-4} = 3.993\times10^{-4}$$

$$f(x_u) = f(0.11) = (0.11)^3 - 0.165(0.11)^2 + 3.993\times10^{-4} = -2.662\times10^{-4}$$

Hence $f(x_l)f(x_u) = f(0)f(0.11) = (3.993\times10^{-4})(-2.662\times10^{-4}) < 0$

So there is at least on root between x_ℓ and x_u, that is between 0 and 0.11

Iteration 1

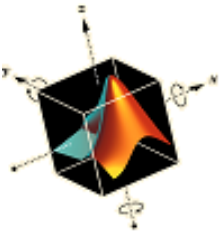
The estimate of the root is $x_m = \dfrac{x_\ell + x_u}{2} = \dfrac{0+0.11}{2} = 0.055$

$$f(x_m) = f(0.055) = (0.055)^3 - 0.165(0.055)^2 + 3.993\times10^{-4} = 6.655\times10^{-5}$$

$$f(x_l)f(x_m) = f(0)f(0.055) = (3.993\times10^{-4})(6.655\times10^{-5}) > 0$$

Hence the root is bracketed between x_m and x_u, that is, between 0.055 and 0.11. So, the lower and upper limits of the new bracket are $x_l = 0.055,\ x_u = 0.11$

At this point, the absolute relative approximate error $|\epsilon_a|$ cannot be calculated as we do not have a previous approximation.



Algoritmo del método de bisección - ejemplo

Iteration 2

The estimate of the root is $x_m = \frac{x_\ell + x_u}{2} = \frac{0.055 + 0.11}{2} = 0.0825$

$$f(x_m) = f(0.0825) = (0.0825)^3 - 0.165(0.0825)^2 + 3.993\times10^{-4} = -1.622\times10^{-4}$$

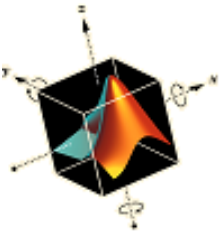
$$f(x_l) f(x_m) = f(0.055) f(0.0825) = (-1.622\times10^{-4})(6.655\times10^{-5}) < 0$$

Hence the root is bracketed between x_ℓ and x_m, that is, between 0.055 and 0.0825. So, the lower and upper limits of the new bracket are $x_l = 0.055$, $x_u = 0.0825$

The absolute relative approximate error $|\epsilon_a|$ at the end of Iteration 2 is

$$|\epsilon_a| = \left|\frac{x_m^{new} - x_m^{old}}{x_m^{new}}\right| \times 100 = \left|\frac{0.0825 - 0.055}{0.0825}\right| \times 100 = 33.333\%$$

None of the significant digits are at least correct in the estimate root of x_m = 0.0825 because the absolute relative approximate error is greater than 5%.



Algoritmo del método de bisección - ejemplo

Iteration 3

The estimate of the root is $x_m = \frac{x_\ell + x_u}{2} = \frac{0.055 + 0.0825}{2} = 0.06875$

$$f(x_m) = f(0.06875) = (0.06875)^3 - 0.165(0.06875)^2 + 3.993\times10^{-4} = -5.563\times10^{-5}$$

$$f(x_l) f(x_m) = f(0.055) f(0.06875) = (6.655\times10^{-5})(-5.563\times10^{-5}) < 0$$

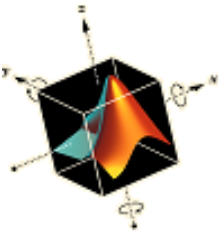
Hence the root is bracketed between x_ℓ and x_m, that is, between 0.055 and 0.06875. So, the lower and upper limits of the new bracket are $x_l = 0.055,\ x_u = 0.06875$

The absolute relative approximate error $|\in_a|$ at the end of Iteration 2 is

$$|\in_a| = \left|\frac{x_m^{new} - x_m^{old}}{x_m^{new}}\right| \times 100 = \left|\frac{0.06875 - 0.0825}{0.06875}\right| \times 100 = 20\%$$

Still none of the significant digits are at least correct in the estimated root of the equation as the absolute relative approximate error is greater than 5%.

Seven more iterations were conducted and these iterations are shown in following Table.



Algoritmo del método de bisección - ejemplo

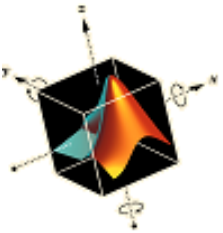
Root of f(x)=0 as function of number of iterations for bisection method.

Iteration	x_ℓ	x_u	x_m	$\lvert\epsilon_a\rvert\,\%$	$f(x_m)$
1	0.00000	0.11	0.055	----------	6.655×10^{-5}
2	0.055	0.11	0.0825	33.33	-1.622×10^{-4}
3	0.055	0.0825	0.06875	20.00	-5.563×10^{-5}
4	0.055	0.06875	0.06188	11.11	4.484×10^{-6}
5	0.06188	0.06875	0.06531	5.263	-2.593×10^{-5}
6	0.06188	0.06531	0.06359	2.702	-1.0804×10^{-5}
7	0.06188	0.06359	0.06273	1.370	-3.176×10^{-6}
8	0.06188	0.06273	0.0623	0.6897	6.497×10^{-7}
9	0.0623	0.06273	0.06252	0.3436	-1.265×10^{-6}
10	0.0623	0.06252	0.06241	0.1721	-3.0768×10^{-7}

Hence the number of significant digits at least correct is given by the largest value or m for which $\lvert\epsilon_a\rvert \le 0.5\times10^{2-m}$, $0.1721 \le 0.5\times10^{2-m}$, $0.3442 \le 10^{2-m}$,

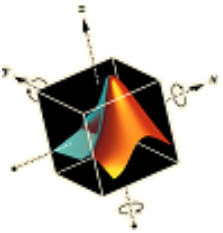
$$\log(0.3442) \le 2-m,\ \ m \le 2-\log(0.3442) = 2.463$$

So $m = 2$. The number of significant digits at least correct at the end of the 10th iter. is 2.



Características del método

- Advantages:
  - Always convergent.
  - The root bracket gets halved with each iteration - guaranteed.
- Drawbacks
  - Slow convergence.
  - If one of the initial guesses is close to the root, the convergence is slower.
  - If a function f(x) is such that it just touches the x-axis it will be unable to find the lower and upper guesses. $f(x) = x^2$
  - Function changes sign but root does not exist. $f(x) = \frac{1}{x}$

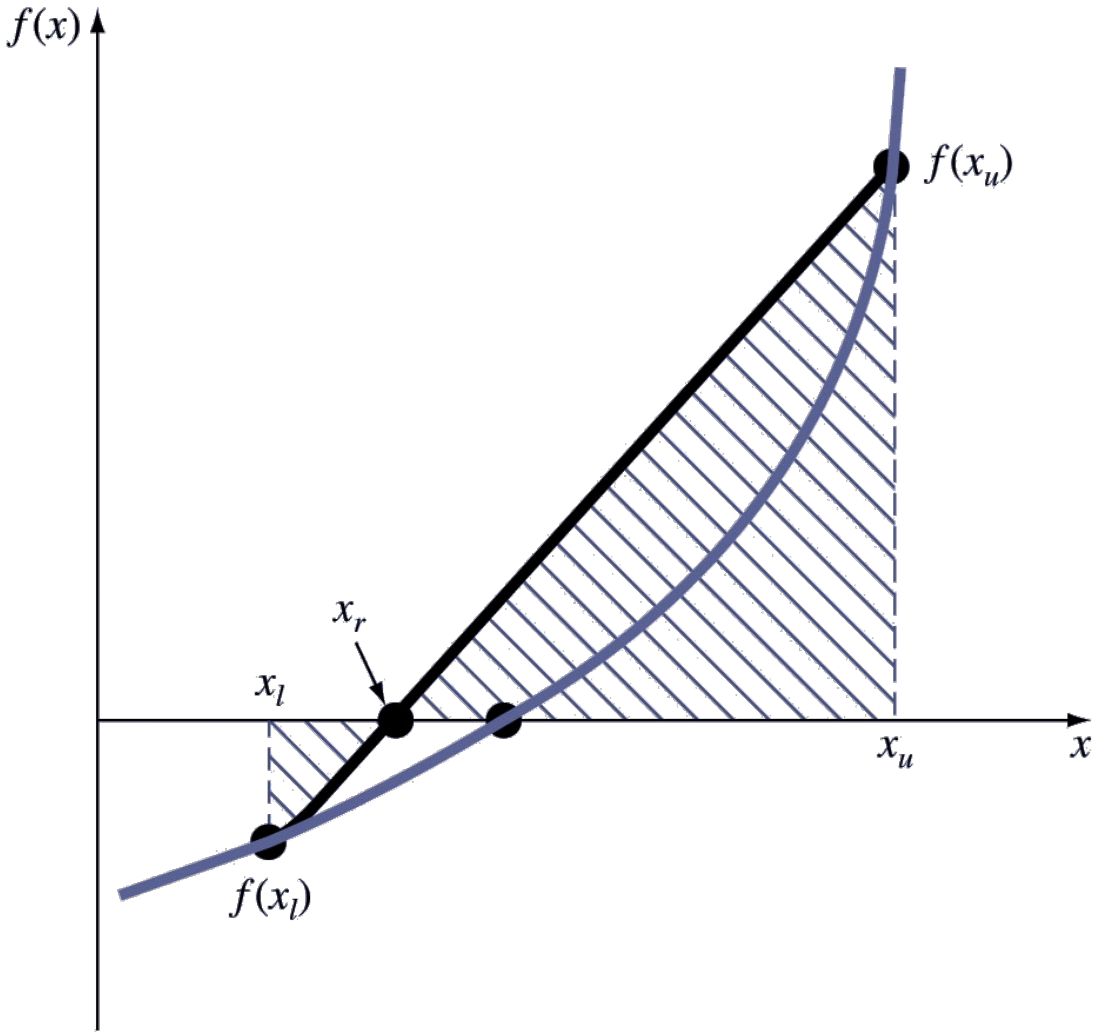


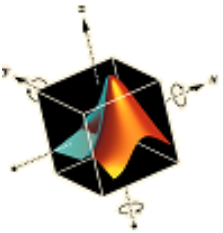
Método de falsa posición

- The *false position* method is another bracketing method.
- It determines the next guess not by splitting the bracket in half but by connecting the endpoints with a straight line and determining the location of the intercept of the straight line (x_r).
- The value of x_r then replaces whichever of the two initial guesses yields a function value with the same sign as $f(x_r)$.

Método de falsa posición

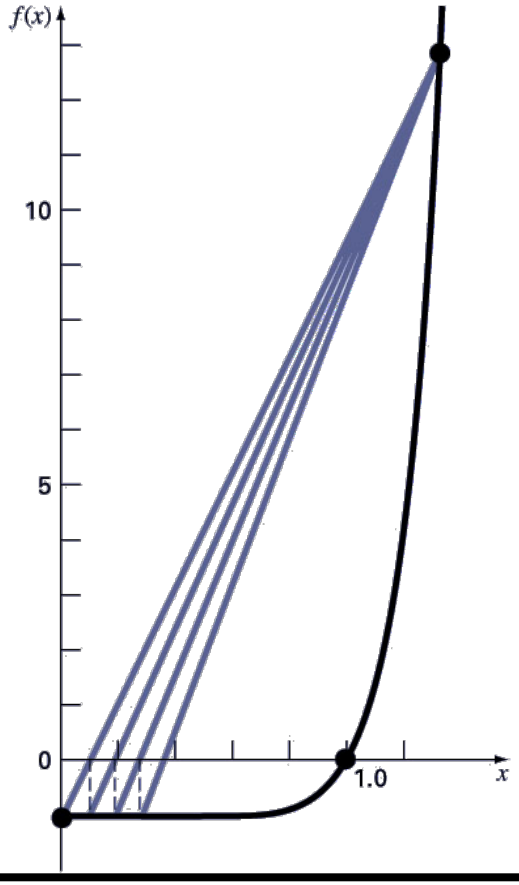
$$x_r = x_u - \frac{f(x_u)(x_l - x_u)}{f(x_l) - f(x_u)}$$

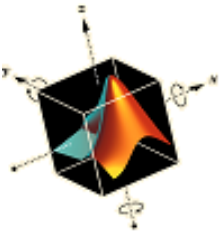




Bisección vs. falsa posición

- Bisection does not take into account the shape of the function; this can be good or bad depending on the function!
- Bad: $f(x) = x^{10} - 1$

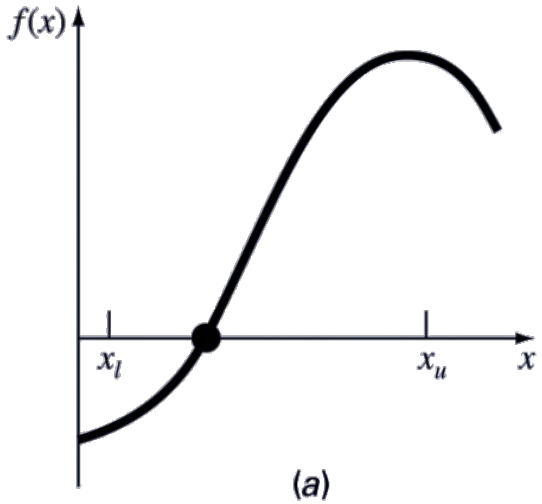




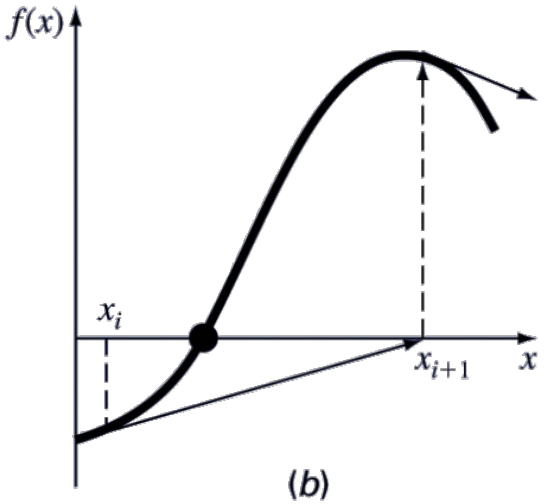
Métodos abiertos

- *Open methods* differ from bracketing methods, in that open methods require only a single starting value or two starting values that do not necessarily bracket a root.
- Open methods may diverge as the computation progresses, but when they do converge, they usually do so much faster than bracketing methods.

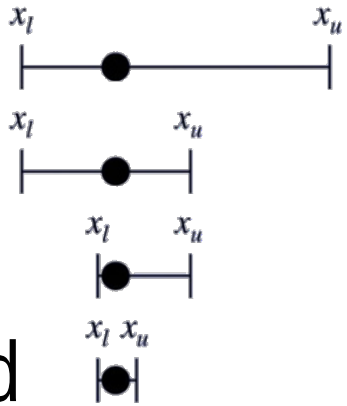
Comparación gráfica de métodos



(a)



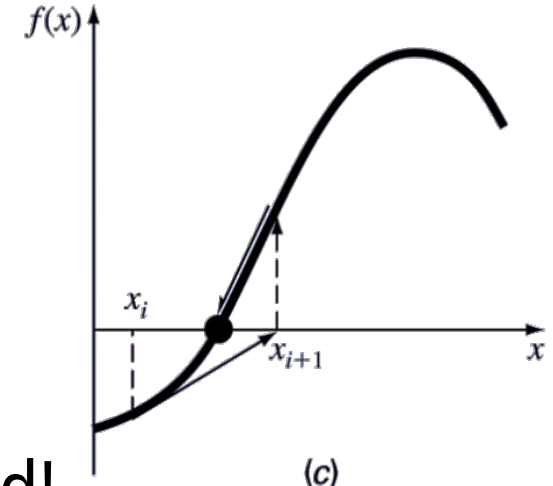
(b)



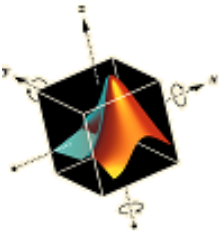
a) Bracketing method

b) Diverging open method

c) Converging open method - note speed!



(c)



Iteración de punto fijo

- Rearrange the function $f(x)=0$ so that x is on the left-hand side of the equation: $x=g(x)$
- Use the new function g to predict a new value of x - that is, $x_{i+1}=g(x_i)$
- The approximate error is given by:

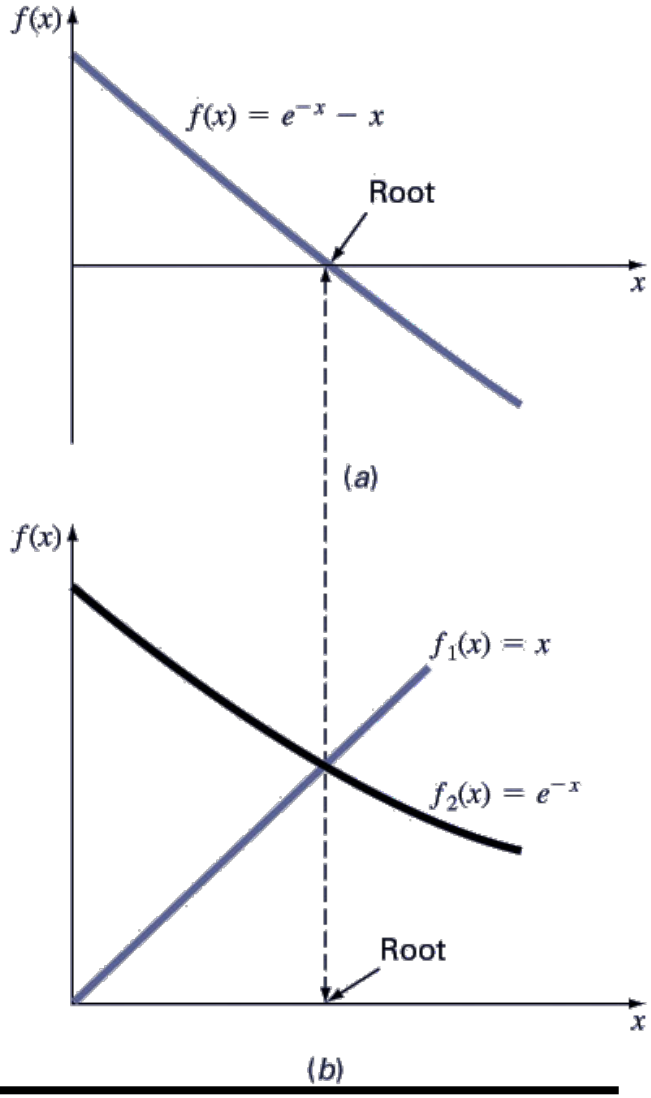
$$\varepsilon_a = \left| \frac{x_{i+1} - x_i}{x_{i+1}} \right| 100\%$$

Iteración de punto fijo - ejemplo

- Solve $f(x)=e^{-x}-x$
- Re-write as $x=g(x)$ by isolating x (example: $x=e^{-x}$)
- Start with an initial guess (here, 0)

i	x_i	$\|\varepsilon_a\|$ %	$\|\varepsilon_t\|$ %	$\|\varepsilon_t\|_i/\|\varepsilon_t\|_{i-1}$
0	0.0000		100.000	
1	1.0000	100.000	76.322	0.763
2	0.3679	171.828	35.135	0.460
3	0.6922	46.854	22.050	0.628
4	0.5005	38.309	11.755	0.533

- Continue until some tolerance is reached



Iteración de punto fijo - convergencia

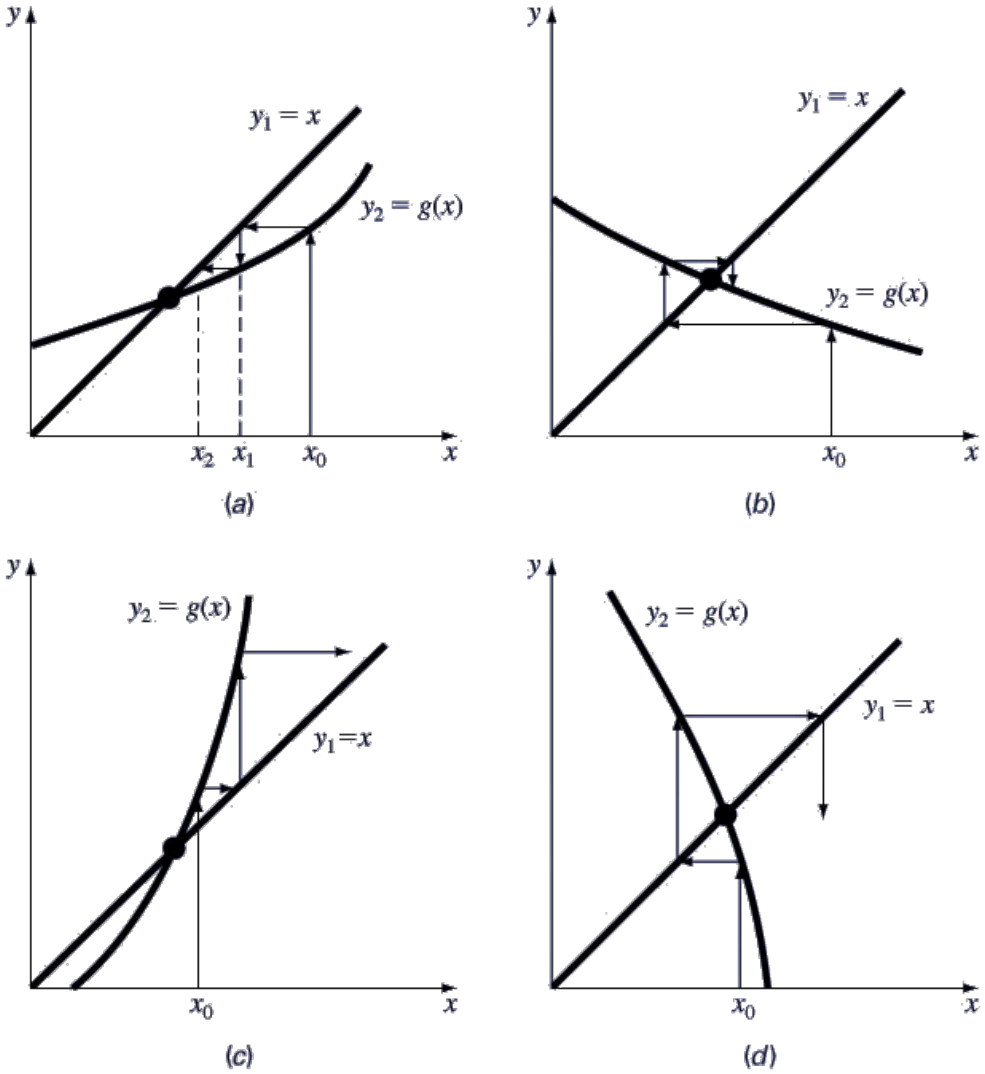
- Convergence of the simple fixed-point iteration method requires that the derivative of g(x) near the root has a magnitude less than 1.

a) Convergent, $0 \leq g' < 1$

b) Convergent, $-1 < g' \leq 0$

c) Divergent, $g' > 1$

d) Divergent, $g' < -1$

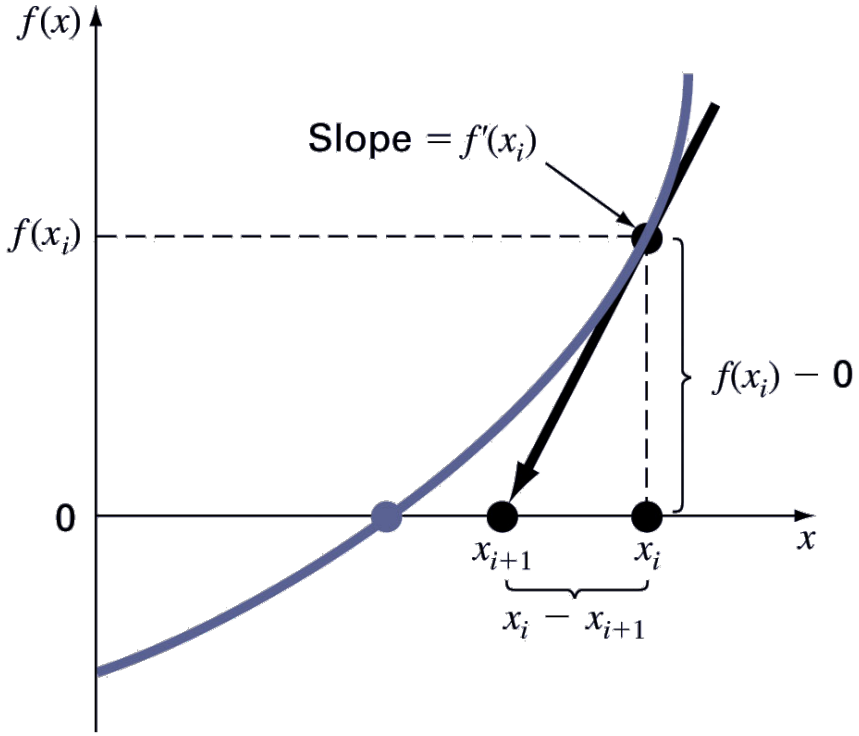


Método de Newton - Rapshon

- Based on forming the tangent line to the $f(x)$ curve at some guess x, then following the tangent line to where it crosses the x-axis.

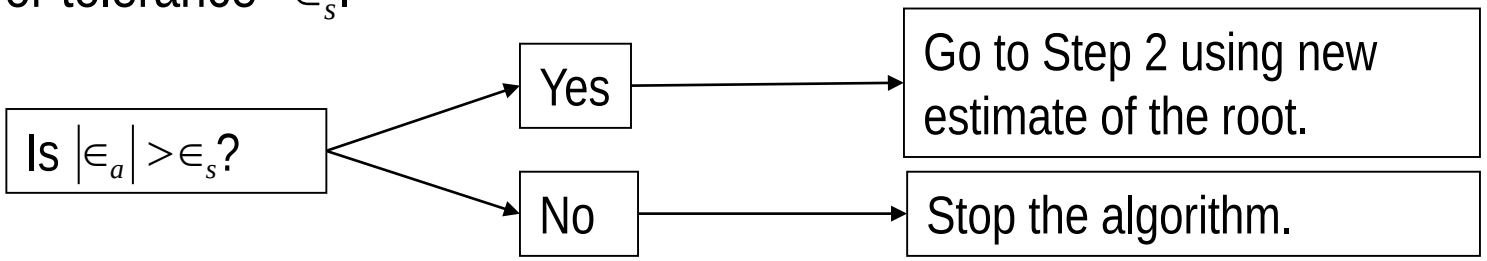
$$f'(x_i) = \frac{f(x_i) - 0}{x_i - x_{i+1}}$$

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$



Algoritmo del método de Newton-Rapshon

- Step 1. Evaluate $f'(x)$ symbolically
- Step 2. Use an initial guess of the root, x_i, to estimate the new value of the root, x_{i+1}, as $$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$
- Step 3. Find the absolute relative approximate error $|\epsilon_a|$ as $|\epsilon_a| = \left|\frac{x_{i+1} - x_i}{x_{i+1}}\right| \times 100$
- Step 4. Compare the absolute relative approximate error with the pre-specified error tolerance ϵ_s.



- Check if the number of iterations has exceeded the maximum number of iterations allowed. If so, one needs to terminate the algorithm and notify the user.

Algoritmo Newton-Rapshon - ejemplo

- The floating ball has a specific gravity of 0.6 and has a radius of 5.5 cm.

  The equation that gives the depth *x* to which the ball is submerged under water is given by

$$x^3 - 0.165x^2 + 3.993\times10^{-4} = 0$$

  Use the Newton's method of finding roots of equations to find

a) the depth 'x' to which the ball is submerged under water. Conduct three iterations to estimate the root of the above equation.

b) The absolute relative approximate error at the end of each iteration, and

c) The number of significant digits at least correct at the end of each iteration.

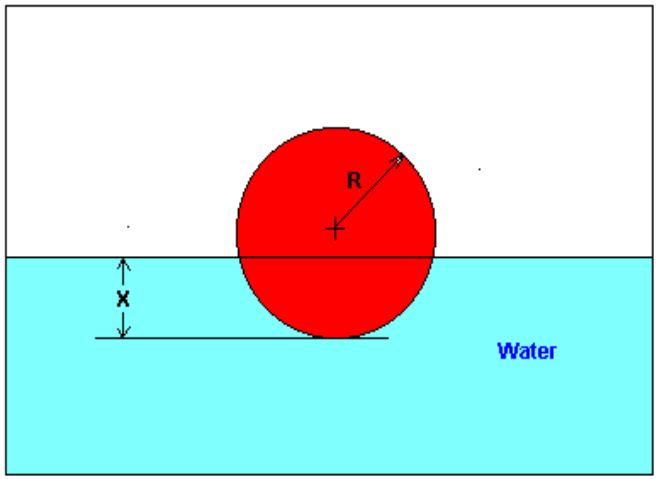
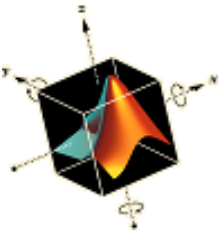


Diagram of the floating ball



Algoritmo Newton-Rapshon - ejemplo

Solve for $f'(x)$

$$f(x) = x^3 - 0.165x^2 + 3.993\times10^{-4}$$

$$f'(x) = 3x^2 - 0.33x$$

Let us assume the initial guess of the root of $f(x)=0$ is $x_0 = 0.05\text{m}$. This is a reasonable guess ($x=0$ and $x=0.11\text{m}$ are not good choices) as the extreme values of the depth x would be 0 and the diameter (0.11 m) of the ball.

Iteration 1

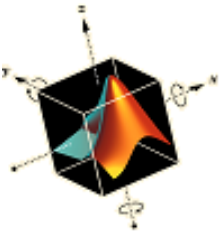
The estimate of the root is

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} = 0.05 - \frac{(0.05)^3 - 0.165(0.05)^2 + 3.993\times10^{-4}}{3(0.05)^2 - 0.33(0.05)}$$

$$= 0.05 - \frac{1.118\times10^{-4}}{-9\times10^{-3}} = 0.05 - (-0.01242) = 0.06242$$

The absolute relative approximate error $|\epsilon_a|$ at the end of Iteration 1 is

$$|\epsilon_a| = \left|\frac{x_1 - x_0}{x_1}\right| \times 100 = \left|\frac{0.06242 - 0.05}{0.06242}\right| \times 100 = 19.90\%$$



Algoritmo Newton-Rapshon - ejemplo

Iteration 2

The estimate of the root is

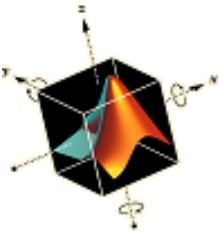
$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)} = 0.06242 - \frac{(0.06242)^3 - 0.165(0.06242)^2 + 3.993\times10^{-4}}{3(0.06242)^2 - 0.33(0.06242)}$$

$$= 0.06242 - \frac{-3.97781\times10^{-7}}{-8.90973\times10^{-3}} = 0.06242 - \left(4.4646\times10^{-5}\right) = 0.06238$$

The absolute relative approximate error $|\in_a|$ at the end of Iteration 2 is

$$|\in_a| = \left|\frac{x_2 - x_1}{x_2}\right| \times 100 = \left|\frac{0.06238 - 0.06242}{0.06238}\right| \times 100 = 0.0716\%$$

The maximum value of m for which $|\in_a| \le 0.5\times10^{2-m}$ is 2.844. Hence, the number of significant digits at least correct in the answer is 2.



Algoritmo Newton-Rapshon - ejemplo

Iteration 3

The estimate of the root is

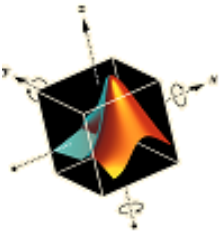
$$x_3 = x_2 - \frac{f(x_2)}{f'(x_2)} = 0.06238 - \frac{(0.06238)^3 - 0.165(0.06238)^2 + 3.993\times10^{-4}}{3(0.06238)^2 - 0.33(0.06238)}$$

$$= 0.06238 - \frac{4.44\times10^{-11}}{-8.91171\times10^{-3}} = 0.06238 - \left(-4.9822\times10^{-9}\right) = 0.06238$$

The absolute relative approximate error $|\in_a|$ at the end of Iteration 2 is

$$|\in_a| = \left|\frac{x_2 - x_1}{x_2}\right| \times 100 = \left|\frac{0.06238 - 0.06238}{0.06238}\right| \times 100 = 0\%$$

The number of significant digits at least correct is 4, as only 4 significant digits are carried through all the calculations.



Características del método

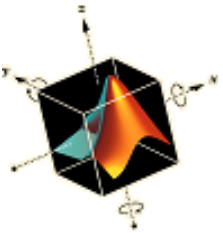
- Advantages:
  - Converges fast (*quadratic convergence*), if it converges.
  - Requires only one guess
- Drawbacks
  - Divergence at inflection points

    Selection of the initial guess or an iteration value of the root that is close to the inflection point of the function $f(x)$ may start diverging away from the root in ther Newton-Raphson method.

    For example, to find the root of the equation

    $$f(x) = (x-1)^3 + 0.512 = 0$$

    The Newton-Raphson method reduces to $x_{i+1} = x_i - \dfrac{(x_i^3 - 1)^3 + 0.512}{3(x_i - 1)^2}$



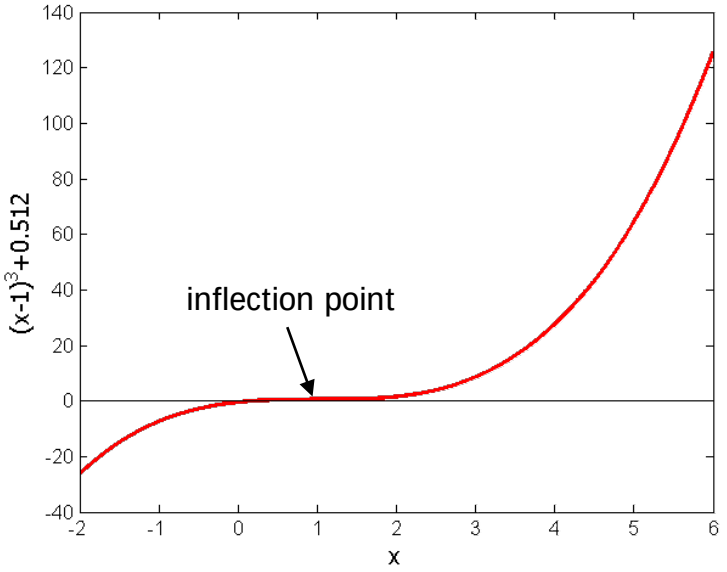
Características del método

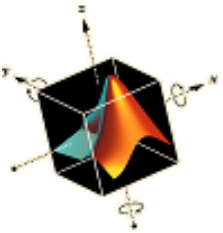
- Divergence at inflection points

  The following table shows the iterated values of the root of the equation. The root starts to diverge at Iter. 6 because the previous estimate of 0.92589 is close to the inflection point of $x = 1$. Eventually after 12 more iterations the root converges to the exact value of $x = 0.2$.

Divergence near inflection point.

Iter.	x_i
0	5.0000
1	3.6560
2	2.7465
3	2.1084
4	1.6000
5	0.92589
6	−30.119
7	−19.746
...	...
18	0.2000





Características del método

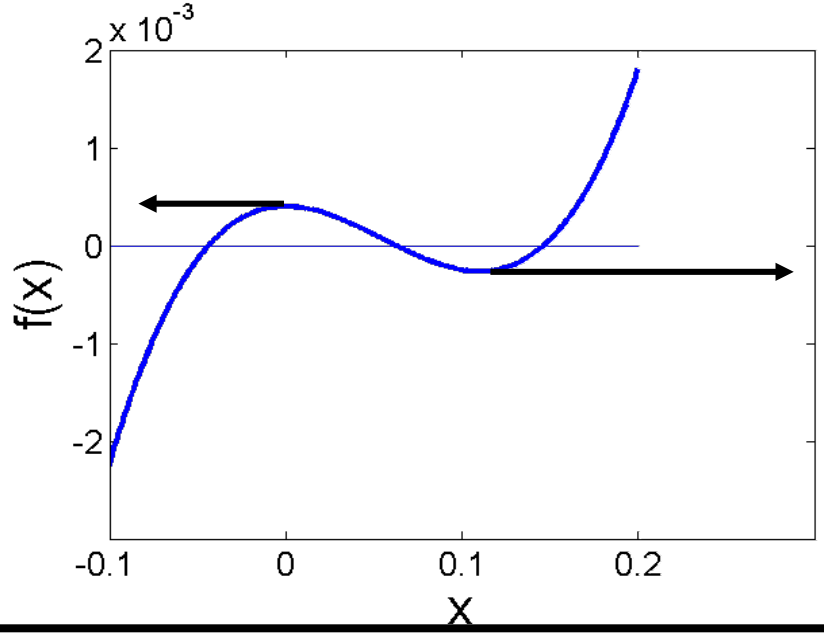
- Division by zero

  For the equation $f(x) = x^3 - 0.03x^2 + 2.4\times10^{-6} = 0$

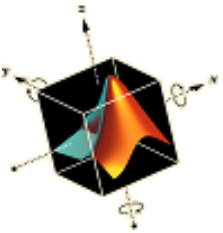
  the Newton-Raphson method reduces to

$$x_{i+1} = x_i - \frac{x_i^3 - 0.03x_i^2 + 2.4\times10^{-6}}{3x_i^2 - 0.06x_i}$$

  For $x_0 = 0$ or $x_0 = 0.02$, the denominator will equal zero.



Pitfall of division by zero or near a zero number



Características del método

- Drawbacks
  - Oscillations near local maximum and minimum

    Results obtained from the Newton-Raphson method may oscillate about the local maximum or minimum without converging on a root but converging on the local maximum or minimum.

    Eventually, it may lead to division by a number close to zero and may diverge.

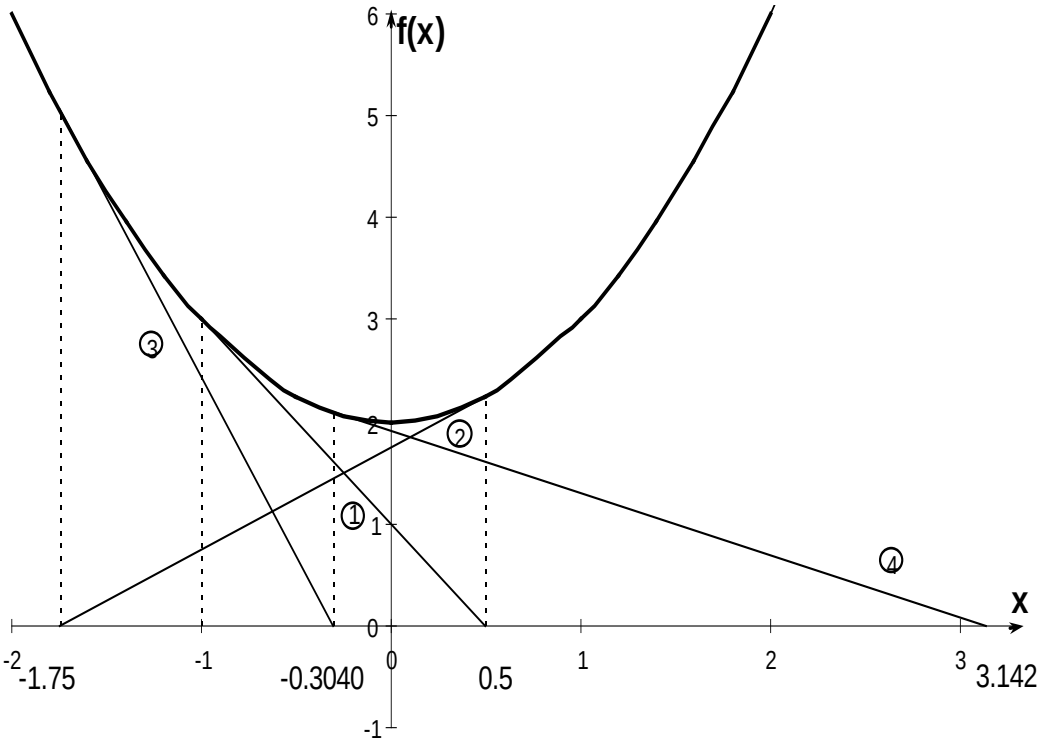
    For example for $f(x) = x^2 + 2 = 0$ the equation has no real roots.

Características del método

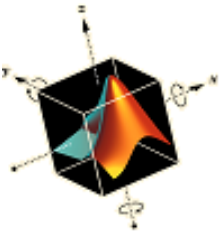
- Drawbacks
  - Oscillations near local maximum and minimum

Oscillations near local maxima and mimima in Newton-Raphson method.

Iter.	x_i	$f(x_i)$	$\lvert\epsilon_a\rvert\%$
0	–1.0000	3.00	
1	0.5	2.25	300.00
2	–1.75	5.063	128.571
3	–0.30357	2.092	476.47
4	3.1423	11.874	109.66
5	1.2529	3.570	150.80
6	–0.17166	2.029	829.88
7	5.7395	34.942	102.99
8	2.6955	9.266	112.93
9	0.97678	2.954	175.96



Oscillations around local minima for $f(x) = x^2 + 2$



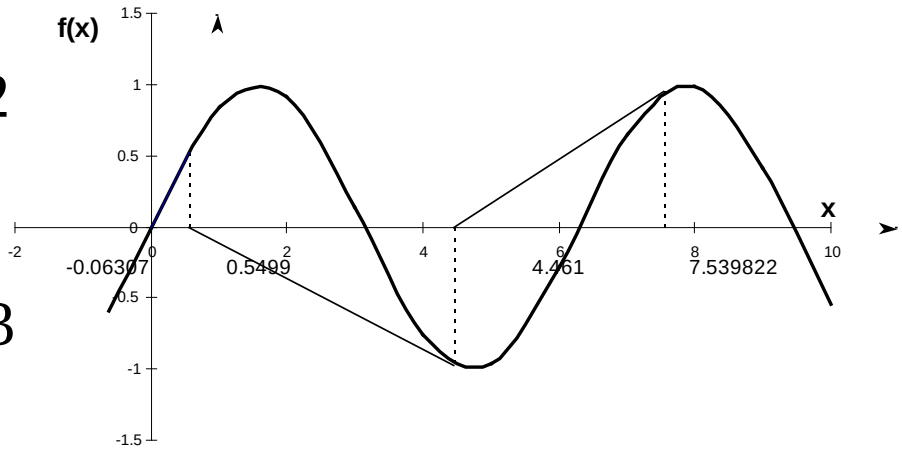
Características del método

- Drawbacks
  - Root Jumping

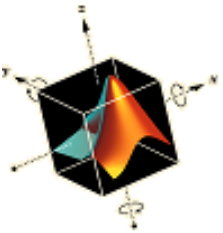
    In some cases where the function $f(x)$ is oscillating and has a number of roots, one may choose an initial guess close to a root. However, the guesses may jump and converge to some other root.

    For example $f(x) = \sin x = 0$
    Choose $x_0 = 2.4\pi = 7.539822$

    It will converge to $x = 0$
    instead of $x = 2\pi = 6.2831853$



Root jumping from intended location of root for $f(x) = \sin x = 0$.



Métodos de la secante

- A potential problem in implementing the Newton-Raphson method is the evaluation of the derivative - there are certain functions whose derivatives may be difficult or inconvenient to evaluate.
- For these cases, the derivative can be approximated by a backward finite divided difference:

$$f'(x_i) \cong \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}}$$

- Substituting this eq. into the eq. of Newton's method $x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$ gives the secant method

$$x_{i+1} = x_i - \frac{f(x_i)(x_i - x_{i-1})}{f(x_i) - f(x_{i-1})}$$

Métodos de la secante

- The secant method can also be derived from geometry:

The Geometric Similar Triangles

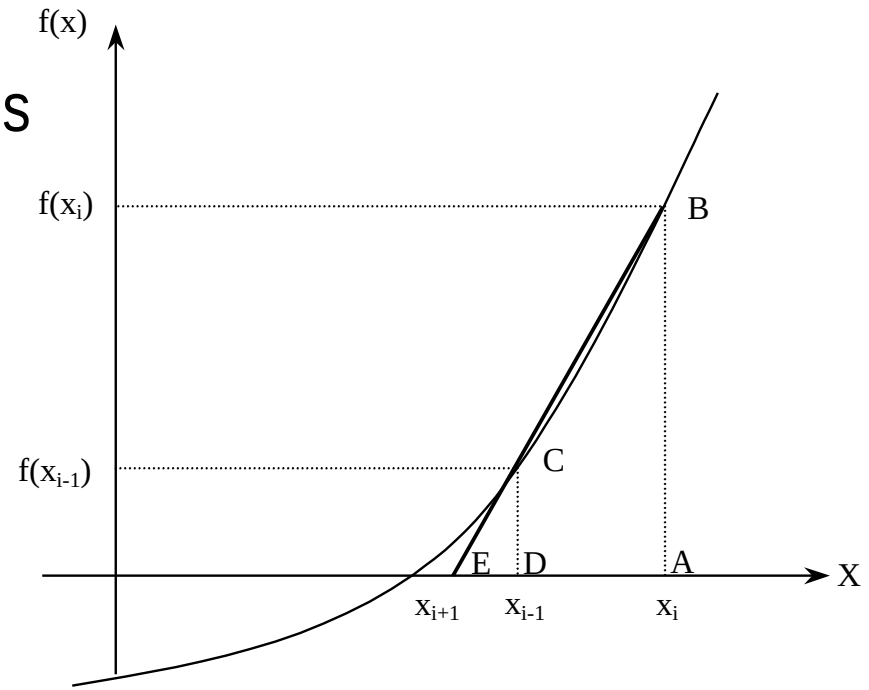
$$\frac{AB}{AE} = \frac{DC}{DE}$$

can be written as

$$\frac{f(x_i)}{x_i - x_{i+1}} = \frac{f(x_{i-1})}{x_{i-1} - x_{i+1}}$$

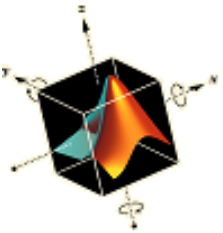
rearranging,

$$x_{i+1} = x_i - \frac{f(x_i)(x_i - x_{i-1})}{f(x_i) - f(x_{i-1})}$$



Geometrical representation of the Secant method.

Note: this method requires *two* initial estimates of x but does *not* require an analytical expression of the derivative.



Algoritmo del método de secante

- Step 1. Calculate the next estimate of the root from two initial guesses

$$x_{i+1} = x_i - \frac{f(x_i)(x_i - x_{i-1})}{f(x_i) - f(x_{i-1})}$$

Find the absolute relative approximate error

$$\left|\epsilon_a\right| = \left|\frac{x_{i+1} - x_i}{x_{i+1}}\right| \times 100$$

- Step 2. Find if the absolute relative approximate error is greater than the prespecified relative error tolerance.

If so, go back to step 1, else stop the algorithm.

Also check if the number of iterations has exceeded the maximum number of iterations.

Algoritmo de secante - ejemplo

- The floating ball has a specific gravity of 0.6 and has a radius of 5.5 cm.

  The equation that gives the depth *x* to which the ball is submerged under water is given by

$$x^3 - 0.165x^2 + 3.993\times10^{-4} = 0$$

  Use the Secant method of finding roots of equations to find

a) the depth 'x' to which the ball is submerged under water. Conduct three iterations to estimate the root of the above equation.

b) Find the absolute relative approximate error and the number of significant digits at least correct at the end of each iteration.

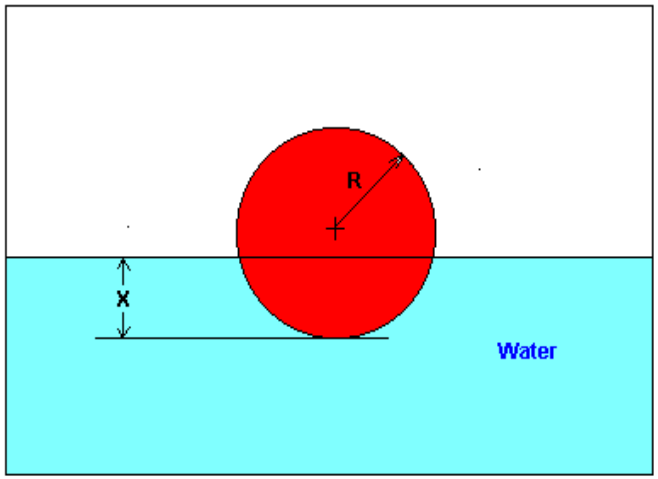
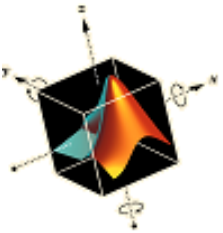


Diagram of the floating ball



Algoritmo del método de secante -ejemplo

Solution

Let us assume the initial guesses of the root of $f(x)=0$ as $x_{-1}=0.02$ and $x_0=0.05$.

Iteration 1

The estimate of the root is $x_1 = x_0 - \dfrac{f(x_0)(x_0 - x_{-1})}{f(x_0) - f(x_{-1})}$

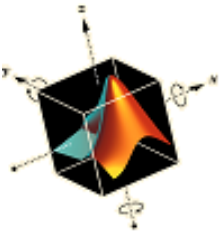
$$= 0.05 - \frac{\left(0.05^3 - 0.165(0.05)^2 + 3.993\times10^{-4}\right)(0.05-0.02)}{\left(0.05^3 - 0.165(0.05)^2 + 3.993\times10^{-4}\right) - \left(0.02^3 - 0.165(0.02)^2 + 3.993\times10^{-4}\right)}$$

$$= 0.06461$$

The absolute relative approximate error $|\epsilon_a|$ at the end of Iteration 1 is

$$|\epsilon_a| = \left|\frac{x_1 - x_0}{x_1}\right| \times 100 = \left|\frac{0.06461 - 0.05}{0.06461}\right| \times 100 = 22.62\%$$

The number of significant digits at least correct is 0, as you need an absolute relative approximate error of 5% or less for one significant digits to be correct in the result.



Algoritmo del método de secante -ejemplo

Iteration 2

The estimate of the root is $x_2 = x_1 - \dfrac{f(x_1)(x_1 - x_0)}{f(x_1) - f(x_0)}$

$$= 0.06461 - \frac{\left(0.06461^3 - 0.165(0.06461)^2 + 3.993\times10^{-4}\right)(0.06461 - 0.05)}{\left(0.06461^3 - 0.165(0.06461)^2 + 3.993\times10^{-4}\right) - \left(0.05^3 - 0.165(0.05)^2 + 3.993\times10^{-4}\right)}$$

$$= 0.06241$$

The absolute relative approximate error $|\epsilon_a|$ at the end of Iteration 2 is

$$|\epsilon_a| = \left|\frac{x_2 - x_1}{x_2}\right| \times 100 = \left|\frac{0.06241 - 0.06461}{0.06241}\right| \times 100 = 3.525\ \%$$

The number of significant digits at least correct is 1, as you need an absolute relative approximate error of 5% or less.

Algoritmo del método de secante -ejemplo

Iteration 3

The estimate of the root is $x_3 = x_2 - \frac{f(x_2)(x_2 - x_1)}{f(x_2) - f(x_1)}$

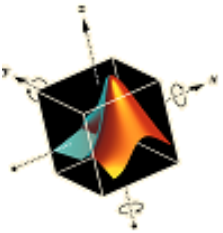
$$= 0.06241 - \frac{\left(0.06241^3 - 0.165(0.06241)^2 + 3.993\times10^{-4}\right)(0.06241 - 0.06461)}{\left(0.06241^3 - 0.165(0.06241)^2 + 3.993\times10^{-4}\right) - \left(0.05^3 - 0.165(0.06461)^2 + 3.993\times10^{-4}\right)}$$

$$= 0.06238$$

The absolute relative approximate error $|\epsilon_a|$ at the end of Iteration 3 is

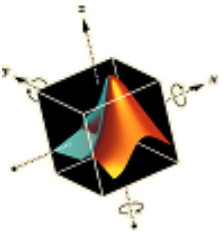
$$|\epsilon_a| = \left|\frac{x_3 - x_2}{x_3}\right| \times 100 = \left|\frac{0.06238 - 0.06241}{0.06238}\right| \times 100 = 0.0595\ \%$$

The number of significant digits at least correct is 5, as you need an absolute relative approximate error of 0.5% or less.



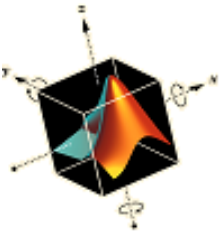
Características del método

- Advantages:
  - Converges fast, if it converges.
  - Requires two guesses that do not need to bracket the root
- Drawbacks
  - Division by zero
  - Root jumping



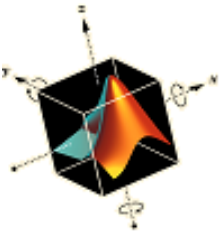
Función `fzero` MATLAB

- MATLAB's `fzero` provides the best qualities of both bracketing methods and open methods.
  - Using an initial guess:
    `x = fzero(function, x0)`
    `[x, fx] = fzero(function, x0)`
    - *`function`* is a function handle to the function being evaluated
    - *`x0`* is the initial guess
    - *`x`* is the location of the root
    - *`fx`* is the function evaluated at that root
  - Using an initial bracket:
    `x = fzero(function, [x0 x1])`
    `[x, fx] = fzero(function, [x0 x1])`
    - As above, except *`x0`* and *`x1`* are guesses that *must* bracket a sign change



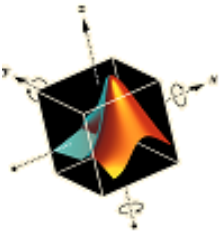
Opciones `fzero`

- Options may be passed to fzero as a third input argument - the options are a data structure created by the `optimset` command
- *options* = optimset(‘par_1’, val_1, ‘par_2’, val_2,…)
  - par_n is the name of the parameter to be set
  - val_n is the value to which to set that parameter
  - The parameters commonly used with `fzero` are:
    - `display`: when set to ‘iter’ displays a detailed record of all the iterations
    - `tolx`: A positive scalar that sets a termination tolerance on x.



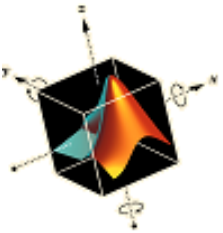
Fzero - ejemplo

- `options = optimset('display', 'iter');`
  - Sets options to display each iteration of root finding process
- `[x, fx] = fzero(@(x) x^10-1, 0.5, options)`
  - Uses fzero to find roots of $f(x)=x^{10}-1$ starting with an initial guess of $x=0.5$.
- MATLAB reports `x=1, fx=0` after 35 function counts



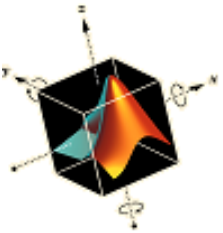
Polinomios

- MATLAB has a built in program called `roots` to determine all the roots of a polynomial - including imaginary and complex ones.
- `x = roots(c)`
  - `x` is a column vector containing the roots
  - `c` is a row vector containing the polynomial coefficients
- Example:
  - Find the roots of $f(x)=x^5-3.5x^4+2.75x^3+2.125x^2-3.875x+1.25$
  - `x = roots([1 -3.5 2.75 2.125 -3.875 1.25])`

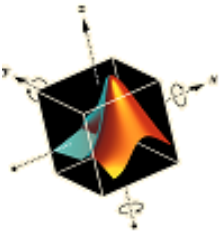


Polinomios

- MATLAB's `poly` function can be used to determine polynomial coefficients if roots are given:
  - `b = poly([0.5 -1])`
    - Finds $f(x)$ where $f(x)=0$ for x=0.5 and x=-1
    - MATLAB reports `b = [1.000 0.5000 -0.5000]`
    - This corresponds to $f(x)=x^2+0.5x-0.5$
- MATLAB's `polyval` function can evaluate a polynomial at one or more points:
  - `a = [1 -3.5 2.75 2.125 -3.875 1.25];`
    - If used as coefficients of a polynomial, this corresponds to $f(x)=x^5-3.5x^4+2.75x^3+2.125x^2-3.875x+1.25$
  - `polyval(a, 1)`
    - This calculates $f(1)$, which MATLAB reports as -0.2500



Integración

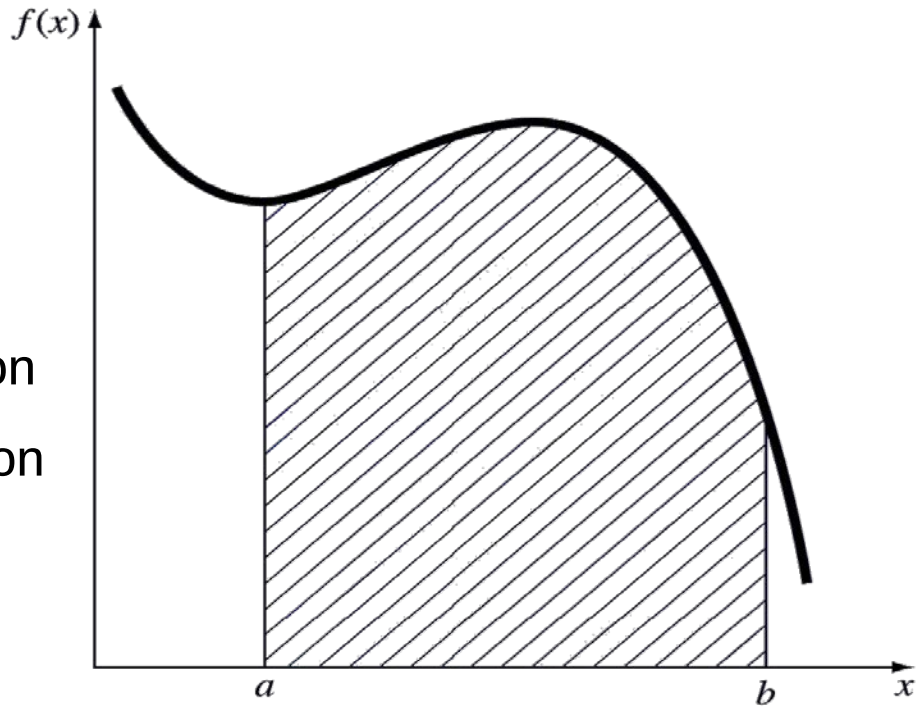


Integración

- Integration:

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

is the total value, or summation, of $f(x)\,dx$ over the range from a to b:

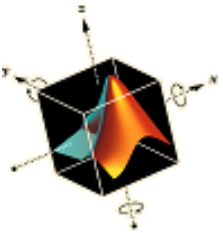


where:

f(x) is the integrand

a= lower limit of integration

b= upper limit of integration

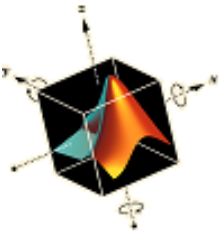


Fórmulas Newton - Cotes

- The *Newton-Cotes* formulas are the most common numerical integration schemes.
- Generally, they are based on replacing a complicated function or tabulated data with a polynomial that is easy to integrate:

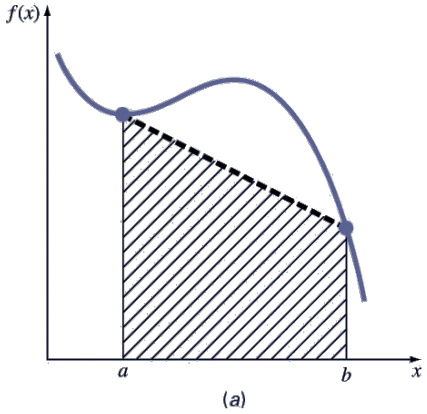
$$I = \int_a^b f(x)\,dx \cong \int_a^b f_n(x)\,dx$$

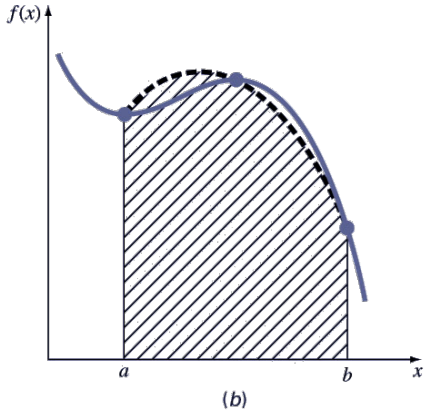
where $f_n(x)$ is an n^{th} order interpolating polynomial.

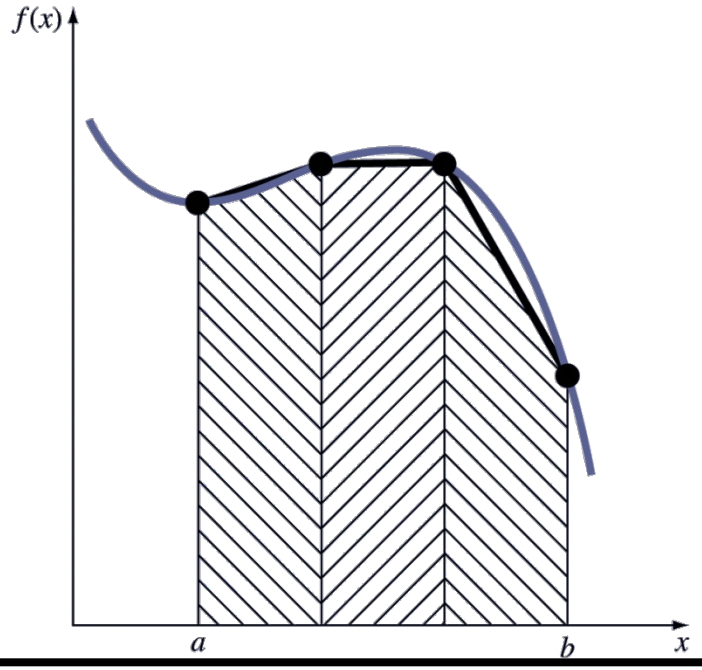


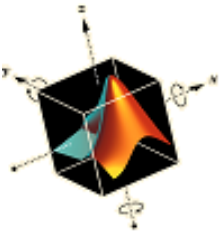
Fórmulas Newton–Cotes - ejemplos

- The integrating function can be polynomials for any order - for example, (a) straight lines or (b) parabolas.
- The integral can be approximated in one step or in a series of steps to improve accuracy.









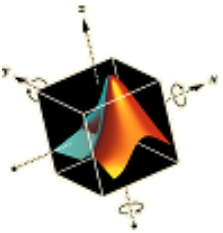
Bases de la regla trapezoidal

- Trapezoidal Rule is based on the Newton-Cotes Formula that states if one can approximate the integrand as an nth order polynomial.

$$I = \int_a^b f(x)dx \quad \text{where} \quad f(x) \approx f_n(x)$$

$$f_n(x) = a_0 + a_1 x + \ldots + a_{n-1}x^{n-1} + a_n x^n$$

- Then the integral of that function is approximated by the integral of that nth order polynomial. $\int_a^b f(x) \approx \int_a^b f_n(x)$
- Trapezoidal Rule assumes n=1, that is, the area under the linear polynomial, $\int_a^b f(x)dx = (b-a)\left[\frac{f(a)+f(b)}{2}\right]$

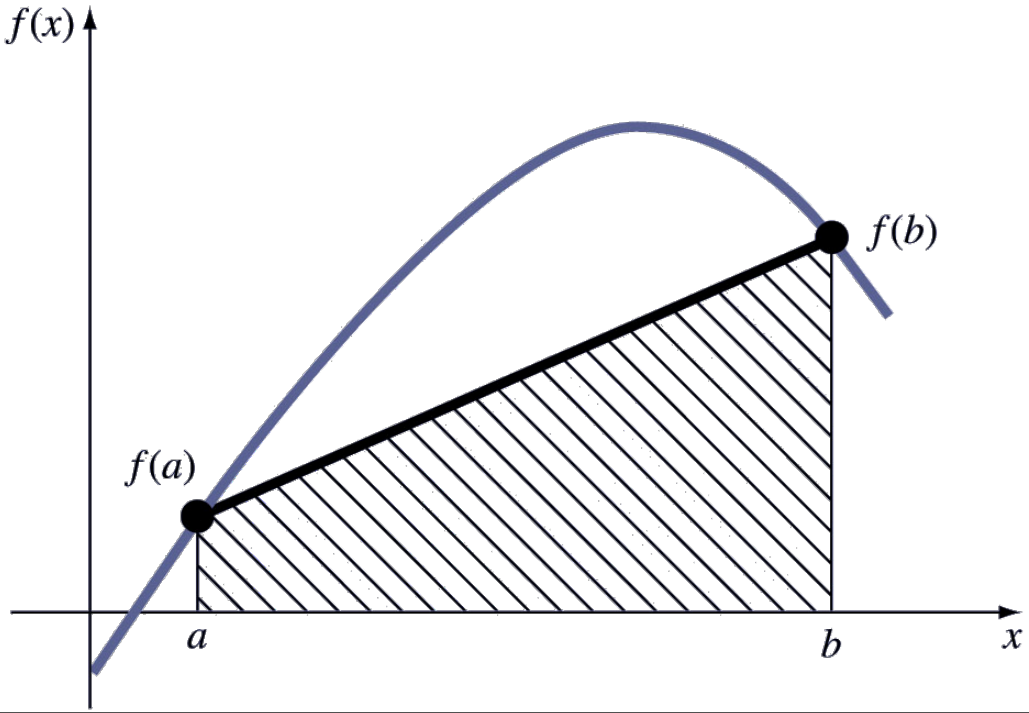


Regla trapezoidal

- The *trapezoidal rule* uses a straight-line approximation for the function:

$$I = \int_a^b f_n(x)\,dx \qquad I = \int_a^b \left[f(a) + \frac{f(b) - f(a)}{b-a}(x-a) \right] dx$$

$$I = (b-a)\frac{f(a) + f(b)}{2}$$



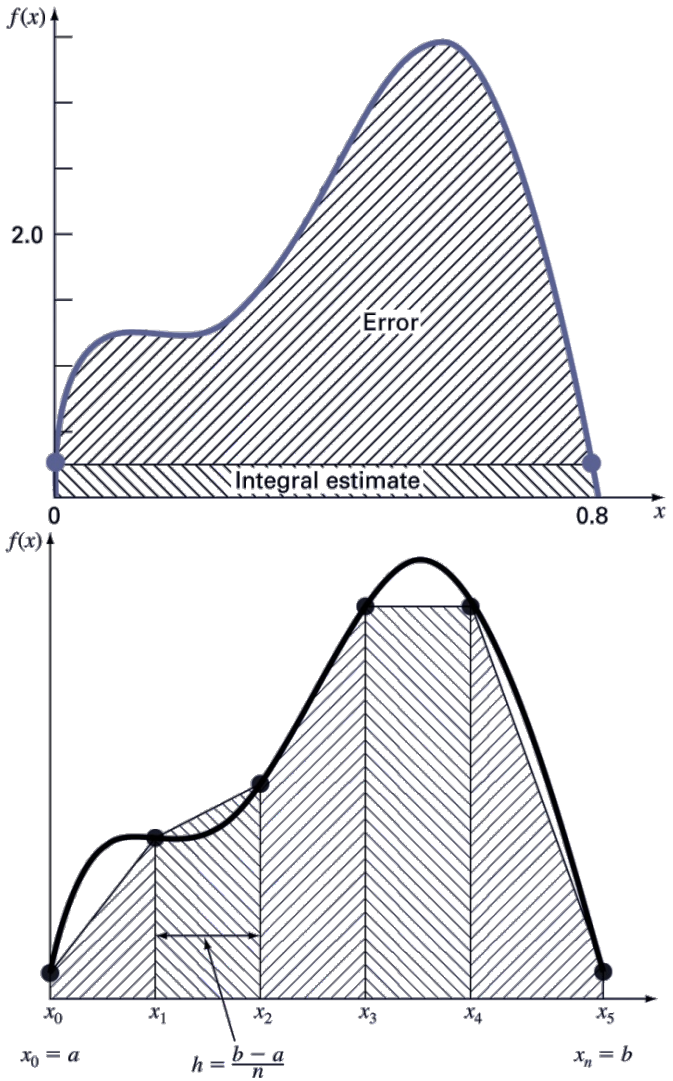
Error de la regla trapezoidal

- An estimate for the local truncation error of a single application of the trapezoidal rule is:

$$E_t = -\frac{1}{12} f''(\xi)(b-a)^3$$

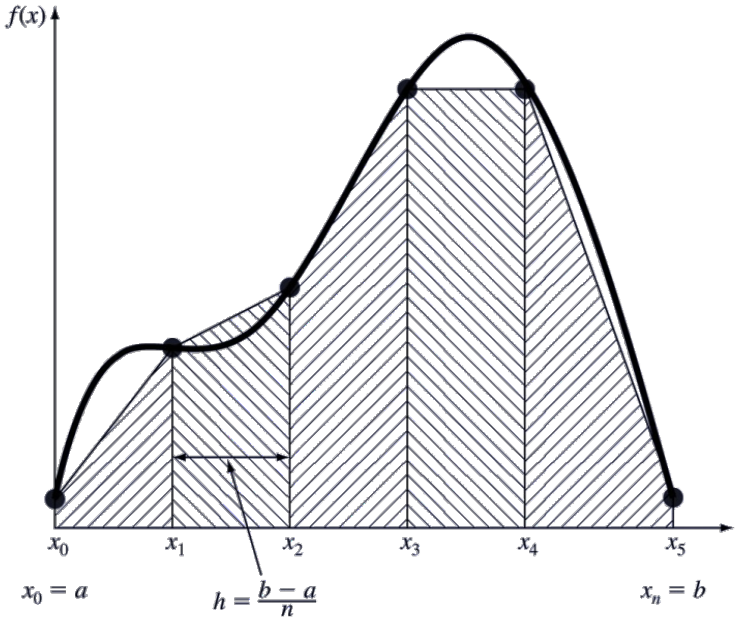
where ξ is somewhere between *a* and *b*.

- This formula indicates that the error is dependent upon the curvature of the actual function as well as the distance between the points.
- Error can thus be reduced by breaking the curve into parts.



Regla trapezoidal compuesta

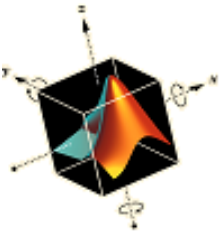
- Assuming *n*+1 data points are evenly spaced, there will be *n* intervals over which to integrate.
- The total integral can be calculated by integrating each subinterval and then adding them together:



$$I = \int_{x_0}^{x_n} f_n(x)\,dx = \int_{x_0}^{x_1} f_n(x)\,dx + \int_{x_1}^{x_2} f_n(x)\,dx + \cdots + \int_{x_{n-1}}^{x_n} f_n(x)\,dx$$

$$I = (x_1 - x_0)\frac{f(x_0) + f(x_1)}{2} + (x_2 - x_1)\frac{f(x_1) + f(x_2)}{2} + \cdots + (x_n - x_{n-1})\frac{f(x_{n-1}) + f(x_n)}{2}$$

$$I = \frac{h}{2}\left[f(x_0) + 2\sum_{i=1}^{n-1} f(x_i) + f(x_n) \right] \qquad \text{where} \qquad h = \frac{b-a}{n}$$



Error de la regla trapezoidal compuesta

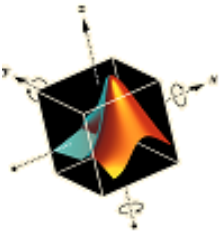
- The true error for a single segment Trapezoidal rule is given by:

$$E_t = \frac{(b-a)^3}{12} f''(\zeta), \quad a < \zeta < b$$ where ζ is some point in $[a,b]$

- The error in the multiple segment Trapezoidal rule is simply the sum of the errors from each segment, where the error in each segment is that of the single segment Trapezoidal rule.
- The total error in multiple segment Trapezoidal rule is

$$E_t = \sum_{i=1}^{n} E_i = \frac{h^3}{12} \sum_{i=1}^{n} f''(\zeta_i) = \frac{(b-a)^3}{12n^2} \frac{\sum_{i=1}^{n} f''(\zeta_i)}{n}$$

- The term $\frac{\sum_{i=1}^{n} f''(\zeta_i)}{n}$ is an approximate average value of $f''(x), a < x < b$
- Note: as the number of segments are doubled, the true error gets approximately quartered.



Regla trapezoidal compuesta - ejemplo

- The vertical distance covered by a rocket from to seconds is given by:

$$x = \int_8^{30} \left(2000 ln \left[\frac{140000}{140000 - 2100t} \right] - 9.8t \right) dt$$

a) Use n-segment Trapezoidal rule to find the distance covered.

b) Find the true error, E_t for part (a).

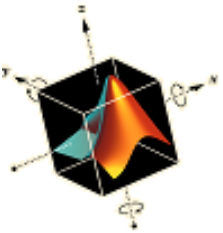
c) Find the absolute relative true error, $|\epsilon_a|$ for part (a).

Exact Value=11061 m

Solution:

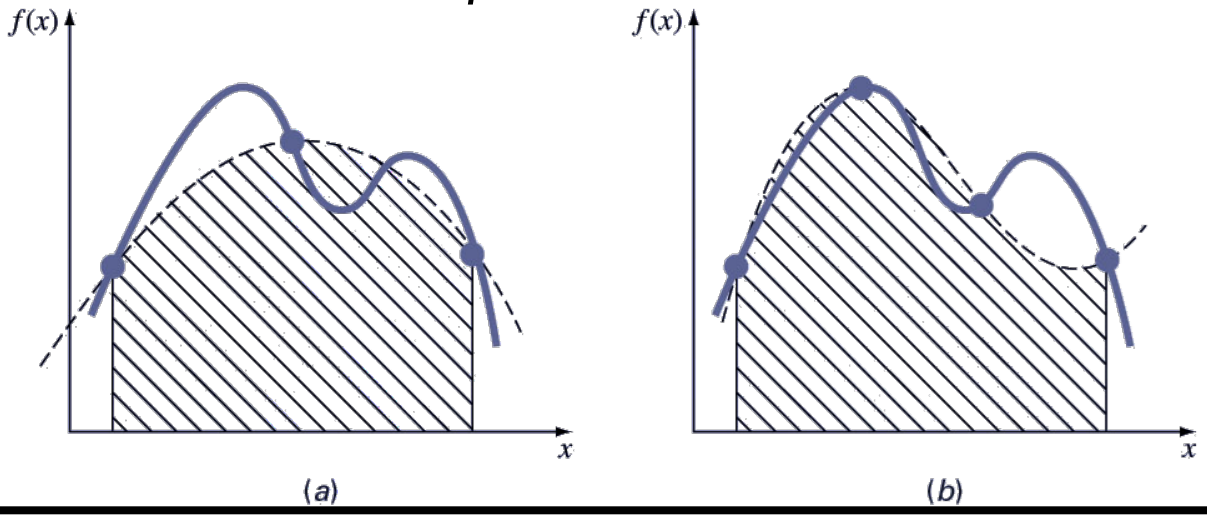
Multiple Segment Trapezoidal Rule Values

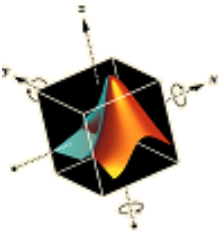
n	Value	E_t	$\|\epsilon_t\|\%$	$\|\epsilon_a\|\%$
1	11868	-807	7.296	---
2	11266	-205	1.853	5.343
3	11153	-91.4	0.8265	1.019
4	11113	-51.5	0.4655	0.3594
5	11094	-33.0	0.2981	0.1669
6	11084	-22.9	0.2070	0.09082
7	11078	-16.8	0.1521	0.05482
8	11074	-12.9	0.1165	0.03560
...	...	...	...	...
16	11065	-3.22	0.02913	0.00401



Reglas de Simpson

- One drawback of the trapezoidal rule is that the error is related to the second derivative of the function.
- More complicated approximation formulas can improve the accuracy for curves - these include using (a) 2nd and (b) 3rd order polynomials.
- The formulas that result from taking the integrals under these polynomials are called *Simpson's rules*.





Regla de Simpson 1/3

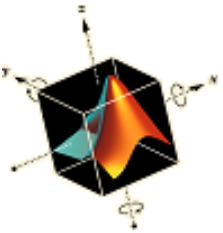
- Simpson's 1/3 rule corresponds to using second-order polynomials. Using the Lagrange form for a quadratic fit of three points:

$$f_n(x)=\frac{(x-x_1)}{(x_0-x_1)}\frac{(x-x_2)}{(x_0-x_2)}f(x_0)+\frac{(x-x_0)}{(x_1-x_0)}\frac{(x-x_2)}{(x_1-x_2)}f(x_1)+\frac{(x-x_0)}{(x_2-x_0)}\frac{(x-x_1)}{(x_2-x_1)}f(x_2)$$

Integration over the three points simplifies to:

$$I=\int_{x_0}^{x_2} f_n(x)\,dx$$

$$I=\frac{h}{3}\left[f(x_0)+4f(x_1)+f(x_2)\right] \qquad h=\frac{x_2-x_0}{2}$$



Error de regla de Simpson 1/3

- An estimate for the local truncation error of a single application of Simpson's 1/3 rule is:

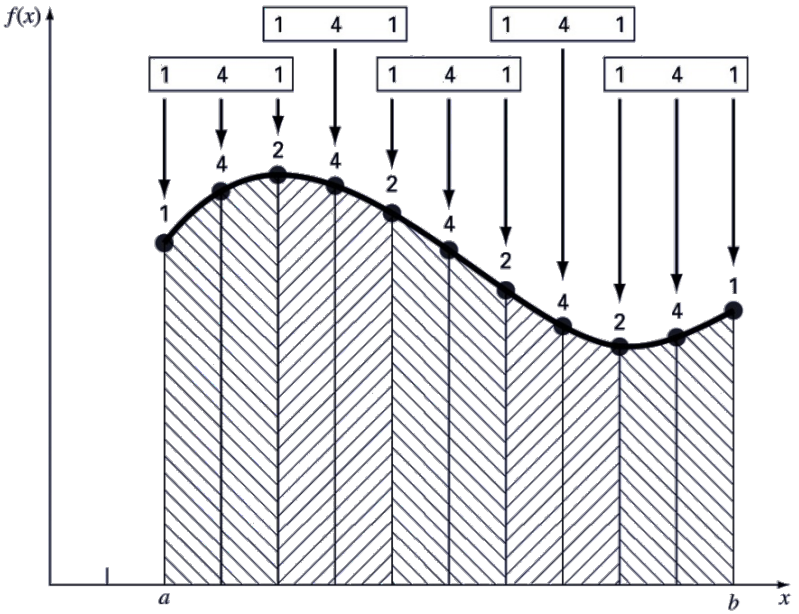
$$E_t = -\frac{1}{2880} f^{(4)}(\xi)(b-a)^5$$

where again ξ is somewhere between *a* and *b*.

- This formula indicates that the error is dependent upon the fourth-derivative of the actual function as well as the distance between the points.
- Note that the error is dependent on the fifth power of the step size (rather than the third for the trapezoidal rule).
- Error can thus be reduced by breaking the curve into parts.

Regla de Simpson 1/3 compuesta

- Simpson's 1/3 rule can be used on a set of subintervals in much the same way the trapezoidal rule was, except there *must* be an odd number of points.
- Because of the heavy weighting of the internal points, the formula is a little more complicated than for the trapezoidal rule:



$$I = \int_{x_0}^{x_n} f_n(x)\,dx = \int_{x_0}^{x_2} f_n(x)\,dx + \int_{x_2}^{x_4} f_n(x)\,dx + \cdots + \int_{x_{n-2}}^{x_n} f_n(x)\,dx$$

$$I = \frac{h}{3}\left[f(x_0) + 4f(x_1) + f(x_2)\right] + \frac{h}{3}\left[f(x_2) + 4f(x_3) + f(x_4)\right] + \cdots + \frac{h}{3}\left[f(x_{n-2}) + 4f(x_{n-1}) + f(x_n)\right]$$

$$I = \frac{h}{3}\left[f(x_0) + 4\sum_{\substack{i=1 \\ i,\,\text{odd}}}^{n-1} f(x_i) + 2\sum_{\substack{j=2 \\ j,\,\text{even}}}^{n-2} f(x_i) + f(x_n) \right] \qquad h = \frac{x_n - x_0}{n}, \quad n = \text{interval number (even)}$$

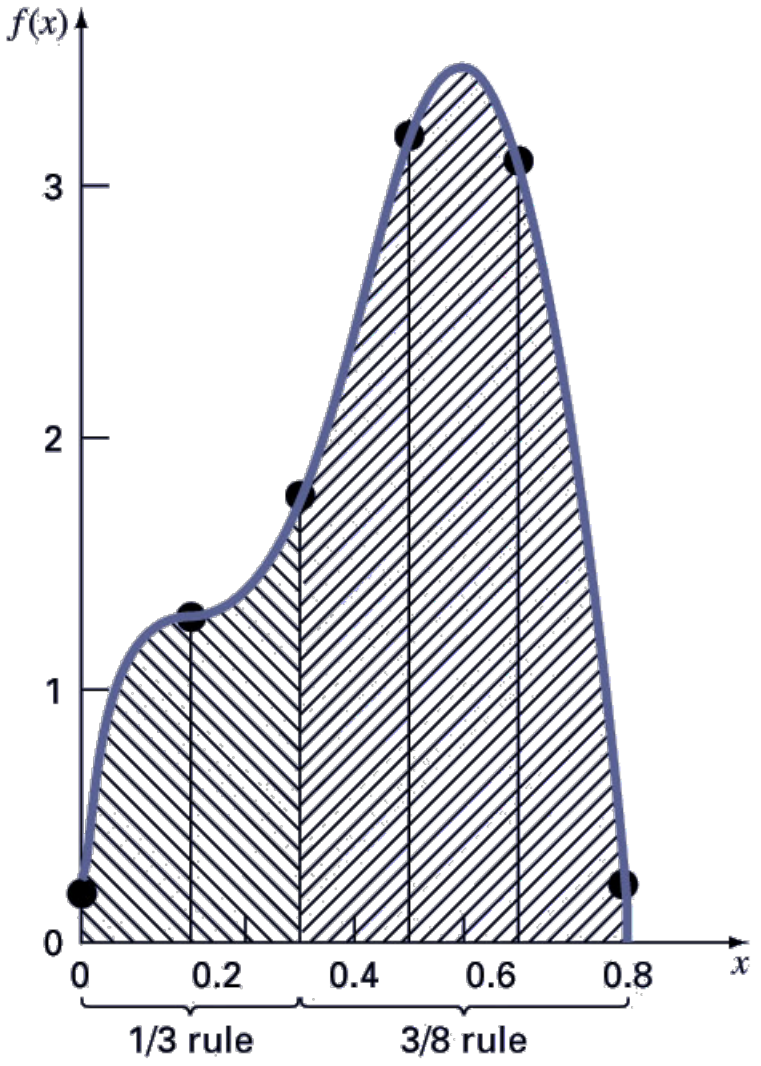
Regla de Simpson 3/8

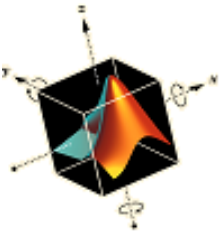
- Simpson's 3/8 rule corresponds to using third-order polynomials to fit four points. Integration over the four points simplifies to:

$$I = \int_{x_0}^{x_3} f_n(x)\,dx \qquad h = \frac{x_3 - x_0}{3}$$

$$I = \frac{3h}{8}\left[f(x_0) + 3f(x_1) + 3f(x_2) + f(x_3)\right]$$

- Simpson's 3/8 rule is generally used in concert with Simpson's 1/3 rule when the number of segments is odd.





Regla de Simpson 3/8 compuesta

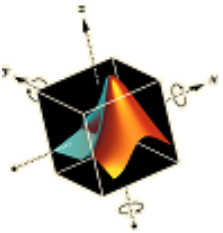
- Using n = number of equal (small) segments multiple of 3, the width h can be defined as $$h = \frac{b-a}{3n}$$

- The integral can be expressed as

$$I \approx \int_{x_0=a}^{x_3} f_3(x)dx + \int_{x_3}^{x_6} f_3(x)dx + \ldots\ldots\ldots + \int_{x_{n-3}}^{x_n=b} f_3(x)dx$$

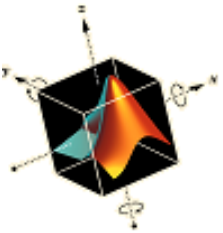
- Substituting Simpson 3/8 rule gives

$$I = \frac{3h}{8}\left\{ f(x_0) + 3\sum_{i=1,4,7,..}^{n-2} f(x_i) + 3\sum_{i=2,5,8,..}^{n-1} f(x_i) + 2\sum_{i=3,6,9,..}^{n-3} f(x_i) + f(x_n) \right\}$$



Reglas de orden mayor

- Higher-order Newton-Cotes formulas may also be used - in general, the higher the order of the polynomial used, the higher the derivative of the function in the error estimate and the higher the power of the step size.
- As in Simpson's 1/3 and 3/8 rule, the even-segment-odd-point formulas have truncation errors that are the same order as formulas adding one more point. For this reason, the even-segment-odd-point formulas are usually the methods of preference.



Regla de Simpson 1/3 compuesta - ejemplo

- The vertical distance covered by a rocket from to seconds is given by:

$$x = \int_8^{30} \left(2000 ln \left[\frac{140000}{140000 - 2100t} \right] - 9.8t \right) dt$$

a) Use n segment Simpson's 1/3rd Rule to find the approximate value of x.

b) Find the true error, E_t for part (a).

c) Find the absolute relative true error, $|\epsilon_a|$ for part (a).

Exact Value=11061.34 m

Solution:

Multiple Segment Simpson's 1/3rd RuleValues

n	Value	E_t	$\|\epsilon_t\|\%$
2	11065.72	4.38	0.0396
4	11061.64	0.30	0.0027
6	11061.40	0.06	0.0005
8	11061.35	0.01	0.0001
10	11061.34	0.00	0.0000

Regla de Simpson 3/8 compuesta - ejemplo

- Given $x=\int_{8}^{30}\left(2000\,ln\left[\frac{140000}{140000-2100t}\right]-9.8t\right)dt$. Use 3/8 multiple segment Simpson rule with 6 segment to solve it.

Solution: $h=\frac{30-8}{6}=3.6666 \quad \{x_0, f(x_0)\}=\{8,177.2667\}$

$\{x_1, f(x_1)\}=\{11.6666,270.4104\}$; *where* $x_1=x_0+h=11.6666$

$\{x_2, f(x_2)\}=\{15.3333,372.4629\}$; *where* $x_2=x_0+2h=15.3333$

$\{x_3, f(x_3)\}=\{19,484.7455\}$; *where* $x_3=x_0+3h=19$

$\{x_4, f(x_4)\}=\{22.6666,608.8976\}$; *where* $x_4=x_0+4h=22.6666$

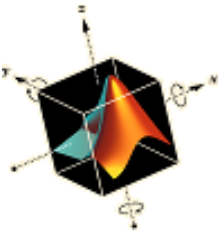
$\{x_5, f(x_5)\}=\{26.3333,746.9870\}$; *where* $x_5=x_0+5h=26.3333$

$\{x_6, f(x_6)\}=\{30,901.6740\}$; *where* $x_6=x_0+6h=30$

Applying Simpson 3/8 rule gives

$$I=\frac{3}{8}(3.6666)\left\{177.2667+3\sum_{i=1,4,..}^{n-2=4}f(x_i)+3\sum_{i=2,5,..}^{n-1=5}f(x_i)+2\sum_{i=3,6,..}^{n-3=3}f(x_i)+901.6740\right\}$$

$$I=11,601.4696$$



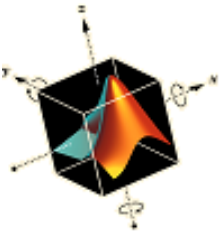
Extrapolación de Richardson

- *Richard extrapolation* methods use two estimates of an integral to compute a third, more accurate approximation.
- If two $O(h^2)$ estimates $I(h_1)$ and $I(h_2)$ are calculated for an integral using step sizes of h_1 and h_2, respectively, an improved $O(h^4)$ estimate may be formed using:

$$I = I(h_2) + \frac{1}{(h_1/h_2)^2 - 1}\left[I(h_2) - I(h_1)\right]$$

- For the special case where the interval is halved ($h_2 = h_1/2$), this becomes:

$$I = \frac{4}{3} I(h_2) - \frac{1}{3} I(h_1)$$



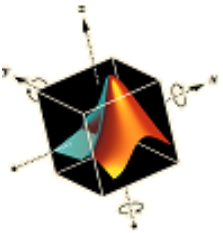
Extrapolación de Richardson

- For the cases where there are two $O(h^4)$ estimates and the interval is halved ($h_m=h_l/2$), an improved $O(h^6)$ estimate may be formed using:

$$I = \frac{16}{15} I_m - \frac{1}{15} I_l$$

- For the cases where there are two $O(h^6)$ estimates and the interval is halved ($h_m=h_l/2$), an improved $O(h^8)$ estimate may be formed using:

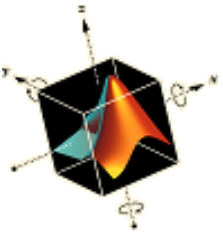
$$I = \frac{64}{63} I_m - \frac{1}{63} I_l$$



Algoritmo de integración de Romberg

- Note that the weighting factors for the Richardson extrapolation add up to 1 and that as accuracy increases, the approximation using the smaller step size is given greater weight.
- In general, $$I_{j,k} = \frac{4^{k-1} I_{j+1,k-1} - I_{j,k-1}}{4^{k-1} - 1}$$

  where $i_{j+1,k-1}$ and $i_{j,k-1}$ are the more and less accurate integrals, respectively, and $i_{j,k}$ is the new approximation. k is the level of integration and j is used to determine which approximation is more accurate.

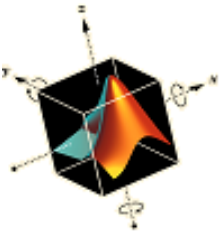


Integración Romberg

- A general expression for Romberg integration can be written as

$$I_{k,j} = I_{k-1,j+1} + \frac{I_{k-1,j+1} - I_{k-1,j}}{4^{k-1} - 1}, k \geq 2$$

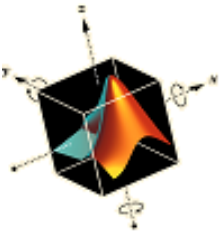
The index k represents the order of extrapolation. k=1 represents the values obtained from the regular Trapezoidal rule, k=2 represents values obtained using the true estimate as $O(h^2)$. The index j represents the more and less accurate estimate of the integral.



Iteraciones del algoritmo Romberg

- The chart below shows the process by which lower level integrations are combined to produce more accurate estimates:

	$O(h^2)$	$O(h^4)$	$O(h^6)$	$O(h^8)$
	0.172800	1.367467		
(a)	1.068800			
	0.172800	1.367467	1.640533	
	1.068800	1.623467		
(b)	1.484800			
	0.172800	1.367467	1.640533	1.640533
	1.068800	1.623467	1.640533	
	1.484800	1.639467		
(c)	1.600800			



Integración con intervalos diferentes

- Previous formulas were simplified based on equispaced data points - though this is not always the case.
- The trapezoidal rule may be used with data containing unequal segments:

$$I = \int_{x_0}^{x_n} f_n(x)\,dx = \int_{x_0}^{x_1} f_n(x)\,dx + \int_{x_1}^{x_2} f_n(x)\,dx + \cdots + \int_{x_{n-1}}^{x_n} f_n(x)\,dx$$

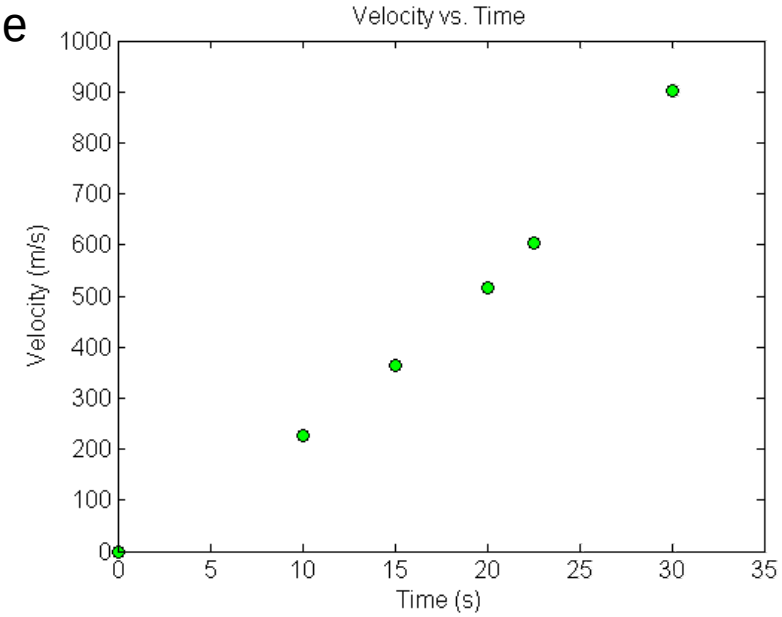
$$I = (x_1 - x_0)\frac{f(x_0) + f(x_1)}{2} + (x_2 - x_1)\frac{f(x_1) + f(x_2)}{2} + \cdots + (x_n - x_{n-1})\frac{f(x_{n-1}) + f(x_n)}{2}$$

Integración discreta - ejemplo

- The upward velocity of a rocket is given as a function of time in the following table.

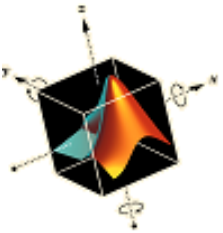
Velocity as a function of time

$t, (\mathrm{s})$	$v(t), (\mathrm{m/s})$
0	0
10	227.04
15	362.78
20	517.35
22.5	602.97
30	901.67



Velocity vs. time data

- Using quadratic splines, find the distance covered between t=11 and t=16 seconds.



Integración discreta - ejemplo

Solution

Spline equations by intervals:

$$
\begin{aligned}
v(t) &= a_1 t^2 + b_1 t + c_1, && 0 \le t \le 10 \\
&= a_2 t^2 + b_2 t + c_2, && 10 \le t \le 15 \\
&= a_3 t^2 + b_3 t + c_3, && 15 \le t \le 20 \\
&= a_4 t^2 + b_4 t + c_4, && 20 \le t \le 22.5 \\
&= a_5 t^2 + b_5 t + c_5, && 22.5 \le t \le 30
\end{aligned}
$$

Solving the spline equations:

$$
\begin{aligned}
v(t) &= 22.704t, && 0 \le t \le 10 \\
&= 0.8888t^2 + 4.928t + 88.88, && 10 \le t \le 15 \\
&= -0.1356t^2 + 35.66t - 141.61, && 15 \le t \le 20 \\
&= 1.6048t^2 - 33.956t + 554.55, && 20 \le t \le 22.5 \\
&= 0.20889t^2 + 28.86t - 152.13, && 22.5 \le t \le 30
\end{aligned}
$$

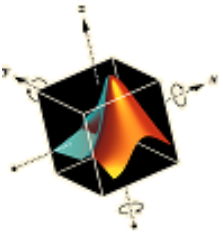
$$S(16) - S(11) = \int_{11}^{16} v(t)dt$$

$$S(16) - S(11) = \int_{11}^{16} v(t)dt = \int_{11}^{15} v(t)dt + \int_{15}^{16} v(t)dt$$

$$= \int_{11}^{15} (0.8888t^2 + 4.928t + 88.88)dt$$

$$+ \int_{15}^{16} (-0.1356t^2 + 35.66t - 141.61)dt$$

$$= 1595.9 \text{ m}$$



Funciones Matlab

- MATLAB has built-in functions to evaluate integrals based on the trapezoidal rule

```
z = trapz(y)
z = trapz(x, y)
```

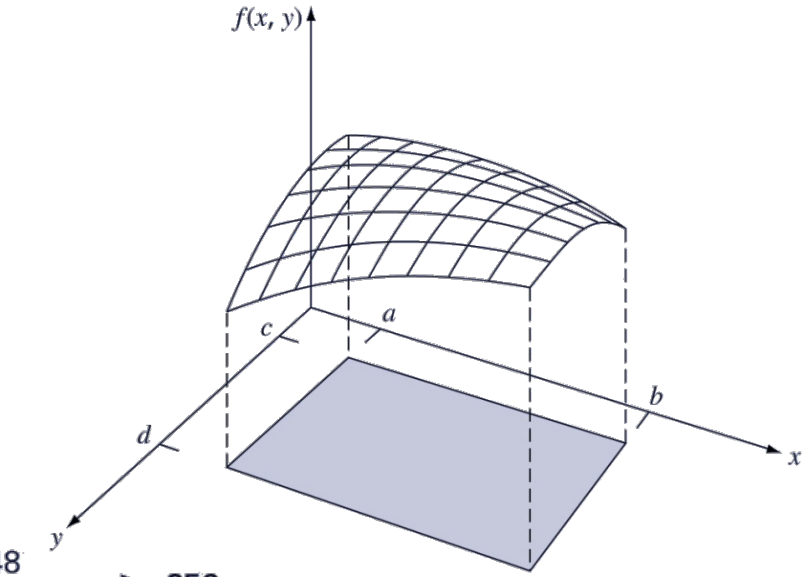
produces the integral of `y` with respect to `x`. If `x` is omitted, the program assumes h=1.

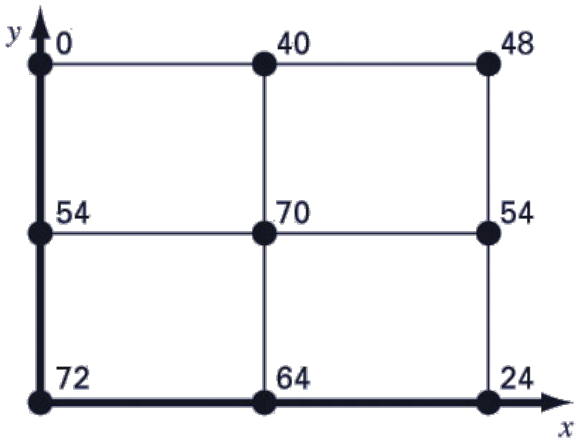
```
z = cumtrapz(y)
z = cumtrapz(x, y)
```

produces the cumulative integral of `y` with respect to `x`. If `x` is omitted, the program assumes h=1.

Integrales múltiples

- Multiple integrals can be determined numerically by first integrating in one dimension, then a second, and so on for all dimensions of the problem.



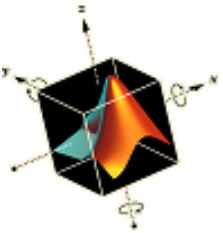


$$(8-0)\frac{0+2(40)+48}{4} \rightarrow \mathbf{256}$$

$$(8-0)\frac{54+2(70)+54}{4} \rightarrow \mathbf{496}$$

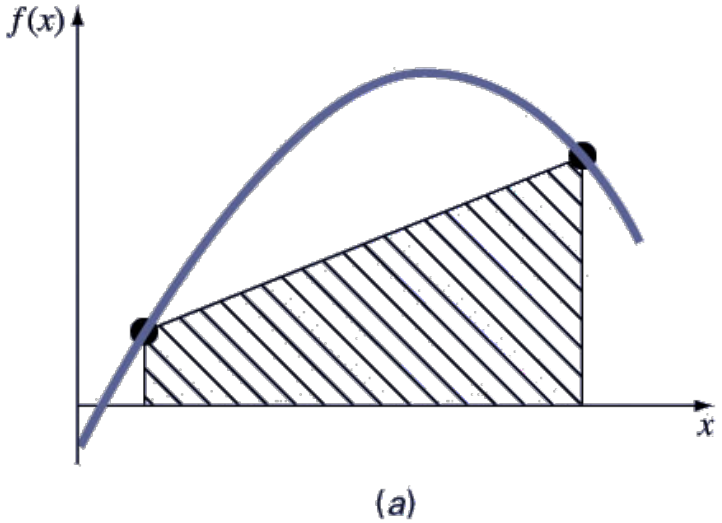
$$(8-0)\frac{72+2(64)+24}{4} \rightarrow \mathbf{448}$$

$$(6-0)\frac{256+2(496)+448}{4} = \mathbf{2688}$$

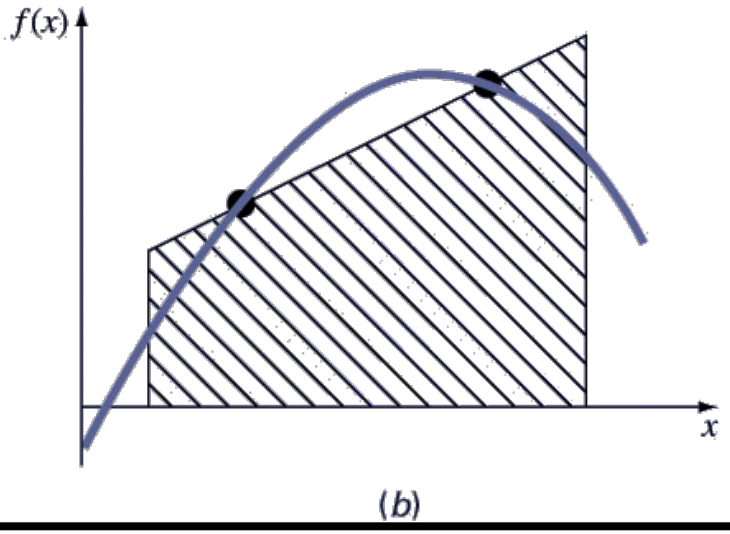


Cuadratura de Gauss

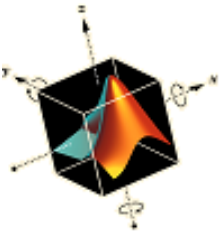
- *Gauss quadrature* describes a class of techniques for evaluating the area under a straight line by joining *any* two points on a curve rather than simply choosing the endpoints.
- The key is to choose the line that balances the positive and negative errors.



(a)



(b)

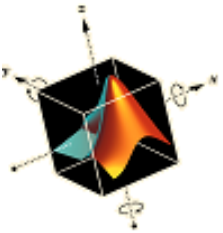


Fórmulas Gauss - Legendre

- The *Gauss-Legendre* formulas seem to optimize estimates to integrals for functions over intervals from -1 to 1.
- Integrals over other intervals require a change in variables to set the limits from -1 to 1.
- The integral estimates are of the form:

$$I \cong c_0 f(x_0) + c_1 f(x_1) + \cdots + c_{n-1} f(x_{n-1})$$

where the c_i and x_i are calculated to ensure that the method exactly integrates up to $(2n\text{-}1)^{th}$ order polynomials over the interval from -1 to 1.



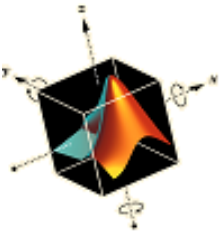
Bases de la regla cuadratura Gaussiana

- The two-point Gauss Quadrature Rule is an extension of the Trapezoidal Rule approximation where the arguments of the function are not predetermined as a and b but as unknowns x_1 and x_2. In the two-point Gauss Quadrature Rule, the integral is approximated as

$$I = \int_a^b f(x)dx \approx c_1 f(x_1) + c_2 f(x_2)$$

- The four unknowns x_1, x_2, c_1 and c_2 are found by assuming that the formula gives exact results for integrating a general third order polynomial,

$$f(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3.$$



Bases de la regla cuadratura Gaussiana

- Hence

$$\int_a^b f(x)dx = \int_a^b \left(a_0 + a_1 x + a_2 x^2 + a_3 x^3\right)dx$$

$$= a_0(b-a) + a_1\left(\frac{b^2-a^2}{2}\right) + a_2\left(\frac{b^3-a^3}{3}\right) + a_3\left(\frac{b^4-a^4}{4}\right)$$

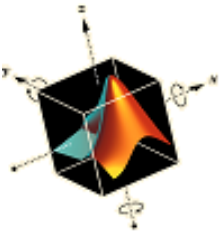
- It follows that

$$\int_a^b f(x)dx = c_1\left(a_0 + a_1 x_1 + a_2 x_1^2 + a_3 x_1^3\right) + c_2\left(a_0 + a_1 x_2 + a_2 x_2^2 + a_3 x_2^3\right)$$

- Resolving

$$x_1 = \left(\frac{b-a}{2}\right)\left(-\frac{1}{\sqrt{3}}\right) + \frac{b+a}{2} \qquad x_2 = \left(\frac{b-a}{2}\right)\left(\frac{1}{\sqrt{3}}\right) + \frac{b+a}{2}$$

$$c_1 = \frac{b-a}{2} \qquad c_2 = \frac{b-a}{2}$$



Fórmulas de cuadratura Gaussiana de mayor orden

- The three-point Gauss Quadrature Rule is

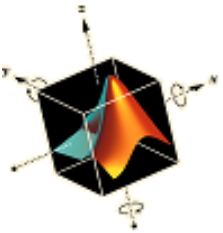
$$\int_a^b f(x)dx \approx c_1 f(x_1) + c_2 f(x_2) + c_3 f(x_3)$$

- The coefficients c_1, c_2 y c_3 and the functional arguments x_1, x_2, and x_3 are found by assuming the formula gives exact results for integrating a fifth order polynomial,

$$\int_a^b \left(a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5\right) dx$$

- General n-point rules would approximate the integral

$$\int_a^b f(x)dx \approx c_1 f(x_1) + c_2 f(x_2) + \ldots\ldots + c_n f(x_n)$$



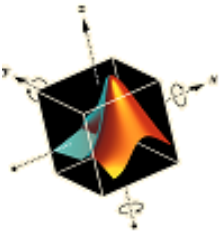
Argumentos y pesos para fórmulas de cuadratura Gaussiana de n-puntos

- In handbooks, coefficients and arguments given for n-point Gauss Quadrature Rule are given for integrals as shown in the Table.

$$\int_{-1}^{1} g(x)dx \cong \sum_{i=1}^{n} c_i g(x_i)$$

Weighting factors c and function arguments x used in Gauss Quadrature Formulas.

Points	Weighting Factors	Function Arguments
2	$c_1 = 1.000000000$ $c_2 = 1.000000000$	$x_1 = -0.577350269$ $x_2 = 0.577350269$
3	$c_1 = 0.555555556$ $c_2 = 0.888888889$ $c_3 = 0.555555556$	$x_1 = -0.774596669$ $x_2 = 0.000000000$ $x_3 = 0.774596669$
4	$c_1 = 0.347854845$ $c_2 = 0.652145155$ $c_3 = 0.652145155$ $c_4 = 0.347854845$	$x_1 = -0.861136312$ $x_2 = -0.339981044$ $x_3 = 0.339981044$ $x_4 = 0.861136312$



Argumentos y pesos para fórmulas de cuadratura Gaussiana de n-puntos

- Any integral with limits of $[a, b]$ can be converted into an integral with limits $[-1, 1]$ with a variable transformation

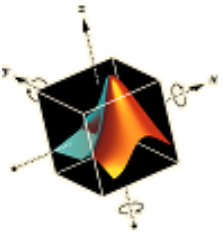
$$x = mt + c$$

Weighting factors c and function arguments x used in Gauss Quadrature Formulas.

Points	Weighting Factors	Function Arguments
5	c_1 = 0.236926885 c_2 = 0.478628670 c_3 = 0.568888889 c_4 = 0.478628670 c_5 = 0.236926885	x_1 = -0.906179846 x_2 = -0.538469310 x_3 = 0.000000000 x_4 = 0.538469310 x_5 = 0.906179846
6	c_1 = 0.171324492 c_2 = 0.360761573 c_3 = 0.467913935 c_4 = 0.467913935 c_5 = 0.360761573 c_6 = 0.171324492	x_1 = -0.932469514 x_2 = -0.661209386 x_3 = -0.2386191860 x_4 = 0.2386191860 x_5 = 0.661209386 x_6 = 0.932469514

We can derive

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



Cuadratura Gaussiana - ejemplo

- The vertical distance covered by a rocket from to seconds is given by:

$$x = \int_{8}^{30} \left(2000 ln\left[\frac{140000}{140000 - 2100t} \right] - 9.8t \right) dt$$

a) Use two-point Gauss Quadrature Rule to find the approximate value of x.

b) Find the true error, E_t for part (a).

c) Find the absolute relative true error, $|\epsilon_a|$ for part (a).

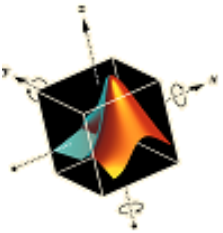
Solution:

First, change the limits of integration from [8,30] to [-1,1]

$$\int_{8}^{30} f(t)dt = \frac{30-8}{2} \int_{-1}^{1} f\left(\frac{30-8}{2} x + \frac{30+8}{2} \right) dx = 11 \int_{-1}^{1} f\left(11x+19\right) dx$$

The weighting factors and function argument values from Table for the two point rule are

$$c_1 = 1.000000000 \qquad c_2 = 1.000000000$$
$$x_1 = -0.577350269 \qquad x_2 = 0.577350269$$



Cuadratura Gaussiana - ejemplo

Replacing in the Gauss Quadrature formula

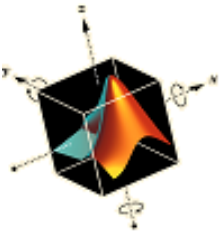
$$11\int_{-1}^{1} f(11x+19)dx \approx 11c_1 f(11x_1+19)+11c_2 f(11x_2+19)$$
$$=11f(11(-0.5773503)+19)+11f(11(0.5773503)+19)$$
$$=11058.44\ m$$

b) The true error, E_t is (exact value = 11061.34m)

$$E_t = True\ Value - Approximate\ Value$$
$$=11061.34-11058.44=2.9000\ m$$

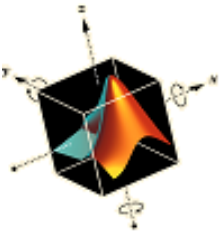
c) The absolute relative true error, $|\epsilon_t|$ is

$$|\epsilon_t| = \left|\frac{11061.34-11058.44}{11061.34}\right| \times 100\%$$
$$=0.0262\%$$



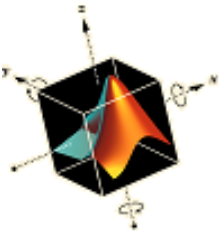
Cuadratura adaptativa

- Methods such as Simpson's 1/3 rule has a disadvantage in that it uses equally spaced points - if a function has regions of abrupt changes, small steps must be used over the *entire domain* to achieve a certain accuracy.
- *Adaptive quadrature* methods for integrating functions automatically adjust the step size so that small steps are taken in regions of sharp variations and larger steps are taken where the function changes gradually.

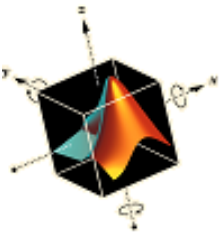


Cuadratura adaptativa en Matlab

- MATLAB has two built-in functions for implementing adaptive quadrature:
  - `quad`: uses adaptive Simpson quadrature; possibly more efficient for low accuracies or nonsmooth functions
  - `quadl`: uses Lobatto quadrature; possibly more efficient for high accuracies and smooth functions
- `q = quad(fun, a, b, tol, trace, p1, p2, …)`
  - *`fun`* : function to be integrates
  - *`a, b`*: integration bounds
  - *`tol`*: desired absolute tolerance (default: 10-6)
  - *`trace`*: flag to display details or not
  - *`p1, p2,`* ...: extra parameters for `fun`
  - `quadl` has the same arguments

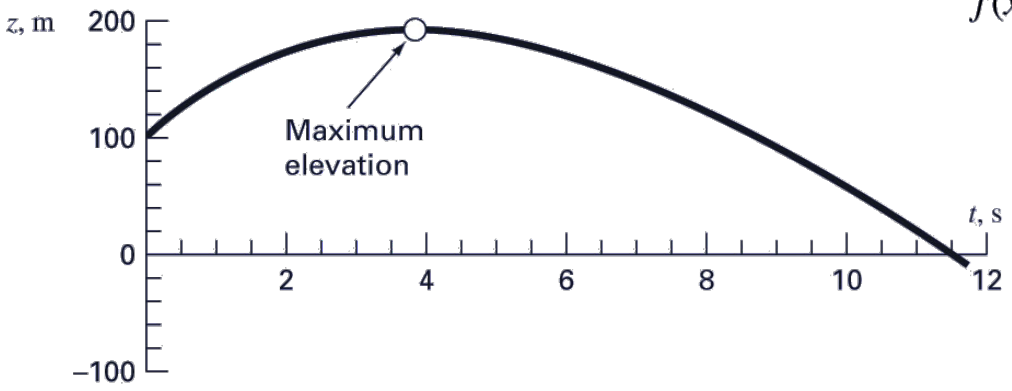


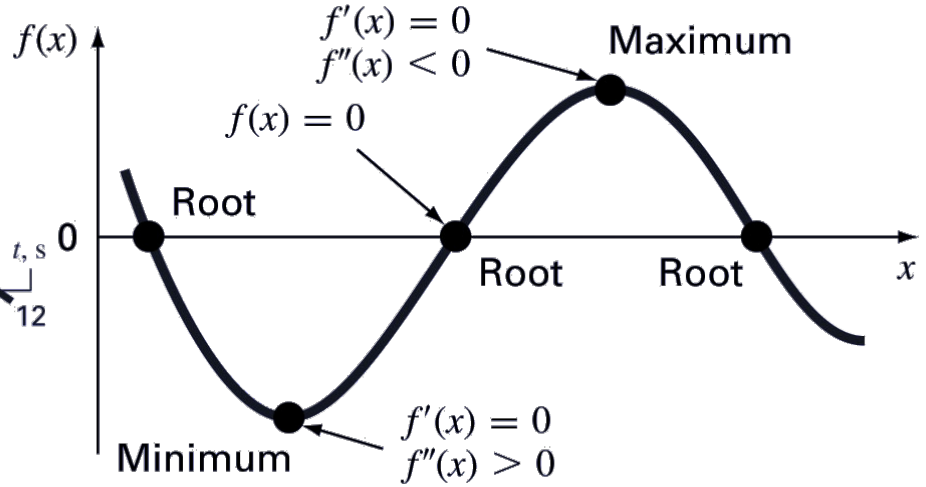
Optimización

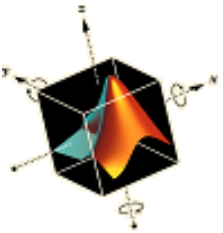


Optimización

- Optimization is the process of creating something that is as effective as possible.
- From a mathematical perspective, optimization deals with finding the maxima and minima of a function that depends on one or more variables.

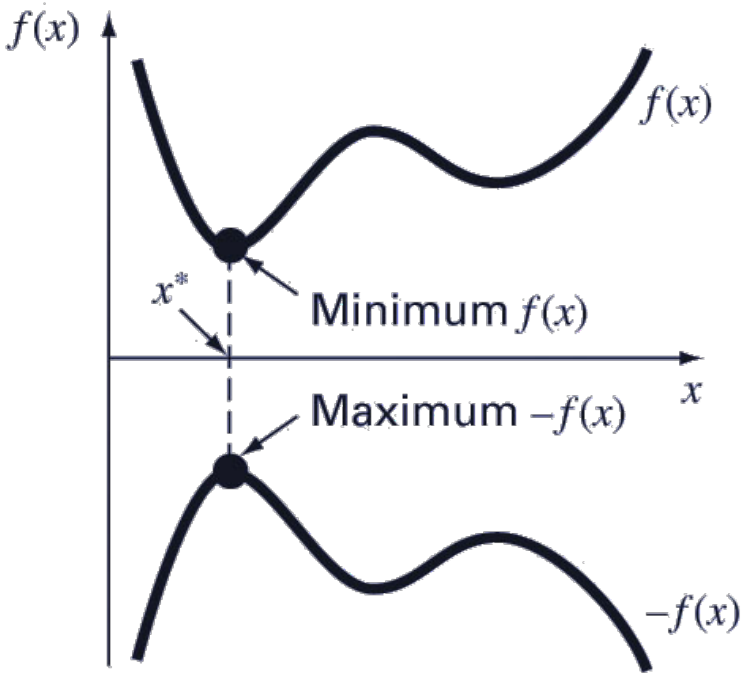


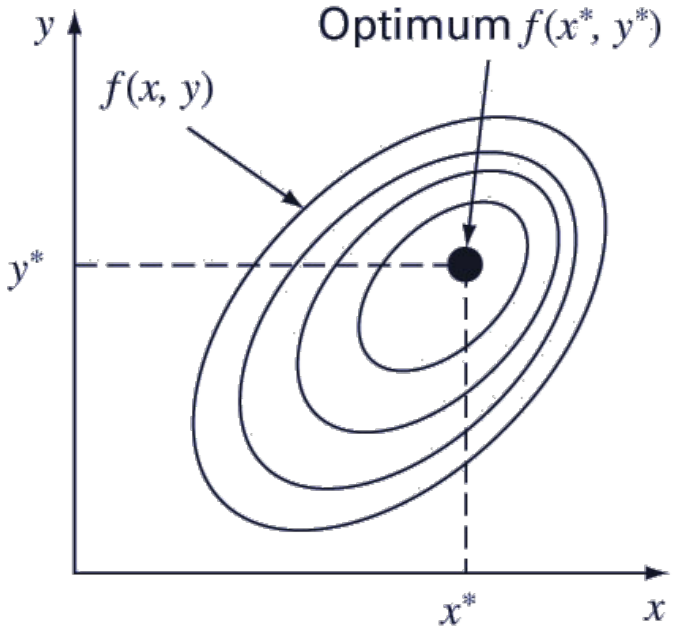


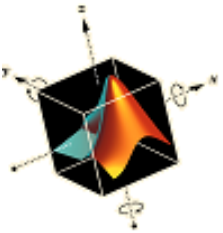


Optimización multidimensional

- One-dimensional problems involve functions that depend on a single dependent variable -for example, *f*(*x*).
- Multidimensional problems involve functions that depend on two or more dependent variables - for example, *f*(*x*,*y*)

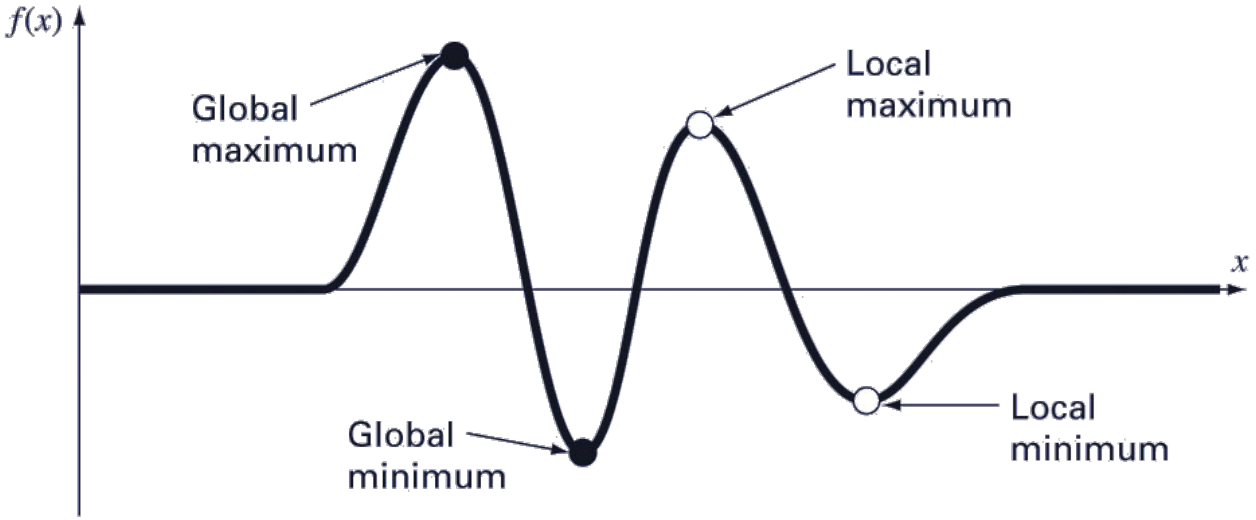






Optimización global vs. local

- A *global optimum* represents the very best solution while a *local optimum* is better than its immediate neighbors. Cases that include local optima are called *multimodal*.
- Generally desire to find the global optimum.



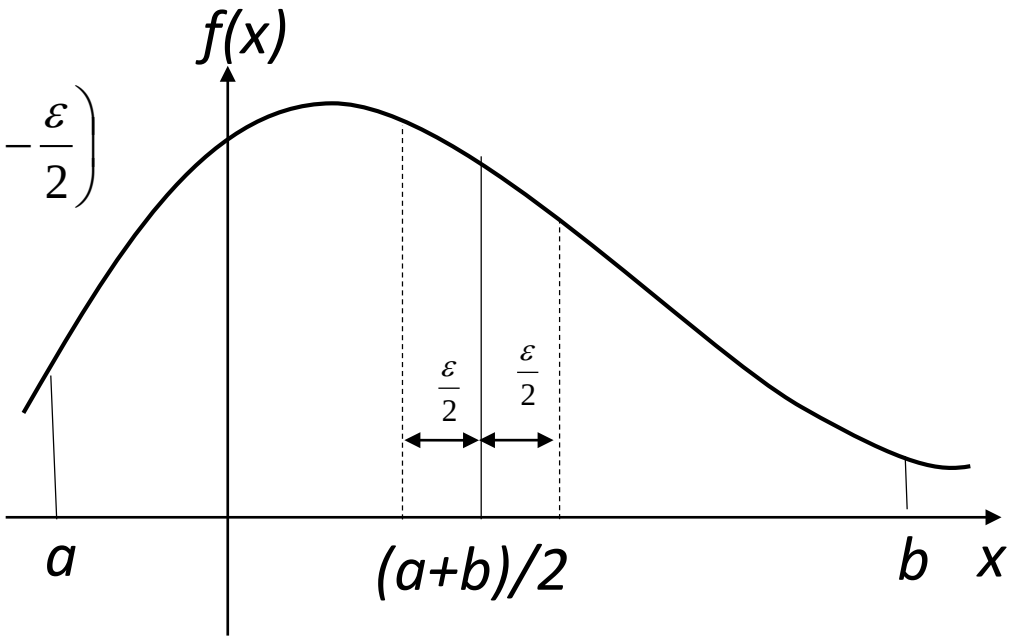
Método de búsqueda - intervalos iguales

- Search algorithm for finding a minimum on an interval [x_a x_b] with a *single* minimum (*unimodal* interval)

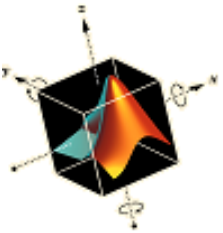
- Choose an interval [a, b] over which the optima occurs
- Compute $f\left(\frac{a+b}{2}+\frac{\varepsilon}{2}\right)$ and $f\left(\frac{a+b}{2}-\frac{\varepsilon}{2}\right)$
- If $f\left(\frac{a+b}{2}+\frac{\varepsilon}{2}\right) > f\left(\frac{a+b}{2}-\frac{\varepsilon}{2}\right)$

  then the interval in which the maximum occurs is $\left[\frac{a+b}{2}-\frac{\varepsilon}{2}, b\right]$

  otherwise it occurs $\left[a, \frac{a+b}{2}+\frac{\varepsilon}{2}\right]$

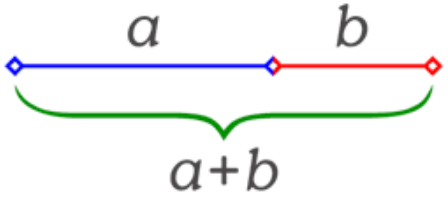


Equal interval search method.



Método de búsqueda – sección áurea

- The Equal Interval method is inefficient when ε is small. The Golden Section Search method divides the search more efficiently closing in on the optima in fewer iterations.
- Uses the *golden ratio* ϕ=1.6180… to determine location of two interior points x_1 and x_2; by using the golden ratio, one of the interior points can be re-used in the next iteration.



$$\frac{a+b}{a}=\frac{a}{b} \quad if \quad \phi=\frac{a}{b}$$

$$1+\phi^{-1}=\phi \qquad \phi^2-\phi-1=0$$

$$\phi=\frac{1+\sqrt{5}}{2}\approx 1.61803398874989...$$

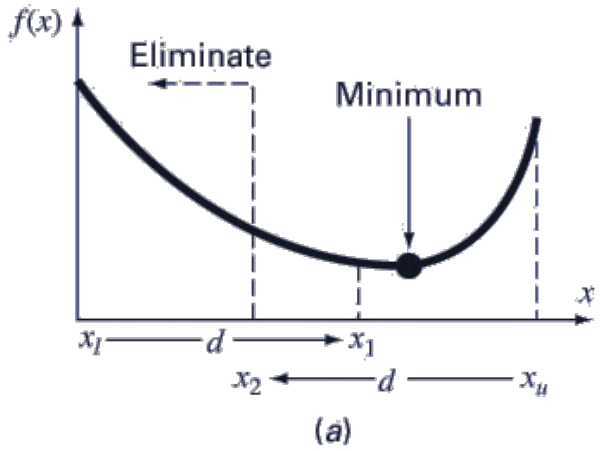
$$d=(\phi-1)(x_u-x_l)$$

$$x_1=x_l+d$$

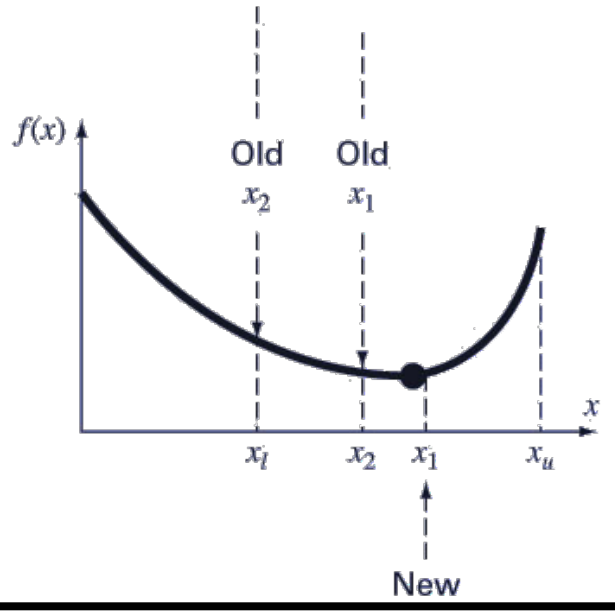
$$x_2=x_u-d$$

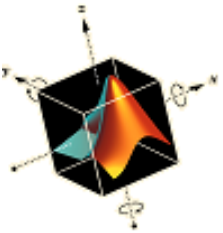
Método de búsqueda – sección áurea

- If $f(x_1)<f(x_2)$, x_2 becomes the new lower limit and x_1 becomes the new x_2 (as in figure).
- If $f(x_2)<f(x_1)$, x_1 becomes the new upper limit and x_2 becomes the new x_1.
- In either case, only one new interior point is needed and the function is only evaluated one more time.



(a)

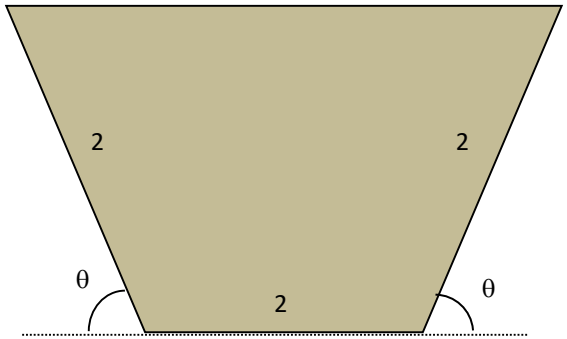




Método de sección áurea- ejemplo

- The cross-sectional area A of a gutter with equal base and edge length of 2 is given by $A = 4\sin\theta(1+\cos\theta)$
  Find the angle θ which maximizes the cross-sectional area of the gutter.
  Using an initial interval of $[0, \pi/2]$.

Solution: The function to be maximized is $f(\theta) = 4\sin\theta(1+\cos\theta)$



Iteration	x_l	x_u	X_1	x_2	$f(x_1)$	$f(x_2)$	ε
1	0.0000	1.5714	0.9712	0.6002	5.1657	4.1238	1.5714
2	0.6002	1.5714	1.2005	0.9712	5.0784	5.1657	0.9712
3	0.6002	1.2005	0.9712	0.8295	5.1657	4.9426	0.6002
4	0.8295	1.2005	1.0588	0.9712	5.1955	5.1657	0.3710
5	0.9712	1.2005	1.1129	1.0588	5.1740	5.1955	0.2293
6	0.9712	1.1129	1.0588	1.0253	5.1955	5.1937	0.1417
7	1.0253	1.1129	1.0794	1.0588	5.1908	5.1955	0.0876
8	1.0253	1.0794	1.0588	1.0460	5.1955	5.1961	0.0541
9	1.0253	1.0588	1.0460	1.0381	5.1961	5.1957	**0.0334**

Optimal solution (theor.)
60 degrees = 1.0472 rad.
Area = 5.1962

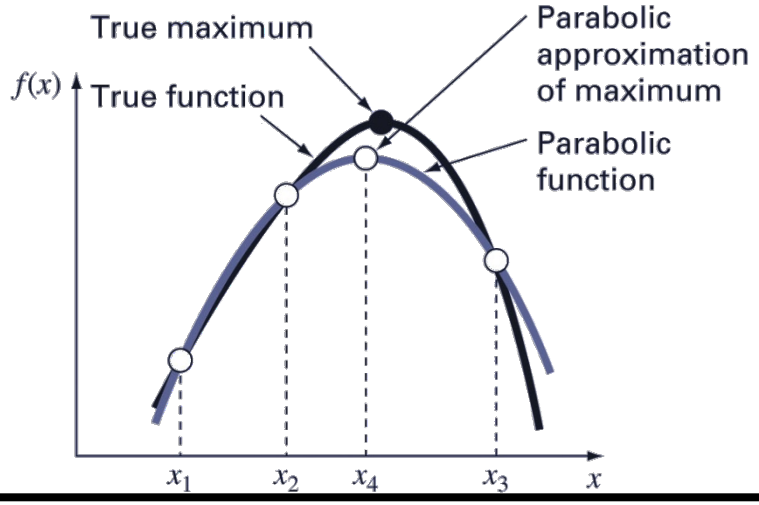
$$\frac{x_u + x_l}{2} = \frac{1.0253 + 1.0588}{2} = 1.0420 \qquad f(1.0420) = 5.1960$$

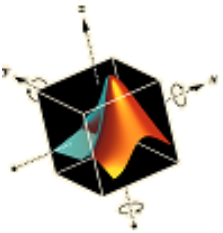
Método de búsqueda – interpolación parabólica

- Another algorithm uses parabolic interpolation of three points to estimate optimum location.
- The location of the maximum/minimum of a parabola defined as the interpolation of three points (x_1, x_2, and x_3) is:

$$x_4 = x_2 - \frac{1}{2}\frac{(x_2 - x_1)^2[f(x_2) - f(x_3)] - (x_2 - x_3)^2[f(x_2) - f(x_1)]}{(x_2 - x_1)[f(x_2) - f(x_3)] - (x_2 - x_3)[f(x_2) - f(x_1)]}$$

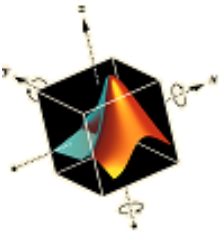
- The new point x_4 and the two surrounding it (either x_1 and x_2 or x_2 and x_3) are used for the next iteration of the algorithm.





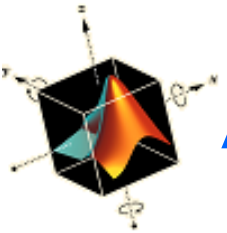
Función `fminbnd`

- MATLAB has a built-in function, `fminbnd`, which combines the golden-section search and the parabolic interpolation.
  - `[xmin, fval] = fminbnd(function, x1, x2)`
- Options may be passed through a fourth argument using `optimset`, similar to `fzero`.



Método de Newton

- Open search method.
- Unlike Golden Section Search method:
  - Lower and upper search boundaries are not required.
  - A good initial estimate of the solution is required.
  - The objective function must be twice differentiable.
  - May not converge to the optimal solution.
- The derivative of the function $f(x)$, $f'(x)=0$ at the function's maximum and minimum. The optimal (min. and max.) can be found by applying the NR method to the derivative, essentially obtaining $x_{i+1} = x_i - \frac{f'(x_i)}{f''(x_i)}$

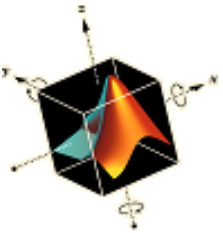


Algoritmo del método de Newton-Rapshon

- **Initialization.** Determine a reasonably good estimate for the maxima or the minima of the function $f(x)$.
- **Step 1.** Determine $f'(x)$ and $f''(x)$.
- **Step 2.** Substitute x_i (initial estimate x_0 for the first iteration) and $f'(x)$ into $f''(x)$

$$x_{i+1} = x_i - \frac{f'(x_i)}{f''(x_i)}$$

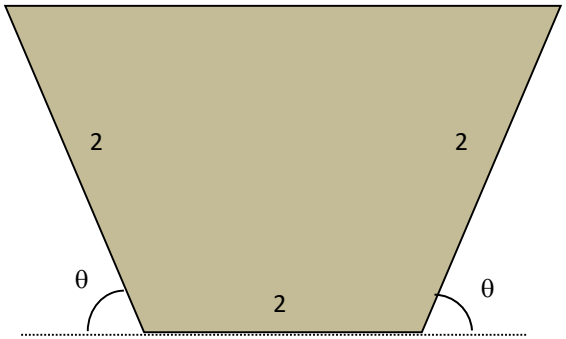
  to determine x_{i+1} and the function value in iteration i.
- **Step 3.** If the value of the first derivative of the function is zero then you have reached the optimum (maxima or minima). Otherwise, repeat Step 2 with the new value of x_i.



Método de Newton Raphson - ejemplo

- The cross-sectional area A of a gutter with equal base and edge length of 2 is given by $A = 4\sin\theta(1+\cos\theta)$
  Find the angle θ which maximizes the cross-sectional area of the gutter.
  Use $\theta_0 = \pi/4$ as the initial estimate of the solution.

Solution: The function to be maximized is $f(\theta) = 4\sin\theta(1+\cos\theta)$

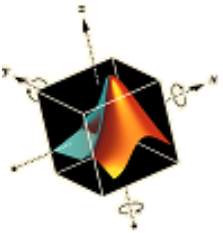


$$f'(\theta) = 4(\cos\theta + \cos^2\theta - \sin^2\theta)$$

$$f''(\theta) = -4\sin\theta(1+4\cos\theta)$$

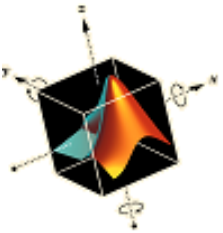
Iteration	θ	$f'(\theta)$	$f''(\theta)$	θ estimate	$f(\theta)$
1	0.7854	2.8284	-10.8284	1.0466	5.1962
2	1.0466	0.0062	-10.3959	1.0472	5.1962
3	1.0472	1.06E-06	-10.3923	1.0472	5.1962
4	1.0472	3.06E-14	-10.3923	1.0472	5.1962
5	1.0472	1.3322E-15	-10.3923	1.0472	5.1962

Optimal solution (theor.)
60 degrees = 1.0472 rad.
Area = 5.1962



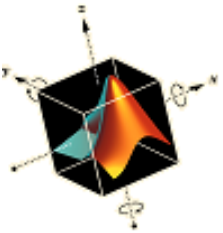
Métodos Multidimensionales de Búsqueda Directa

- Obvious approach is to enumerate all possible solutions and find the min or the max.
  - Very generally applicable but computationally complex.
- Direct search methods are open.
- A good initial estimate of the solution is required.
- The objective function need not be differentiable.



Métodos cíclico coordinado

- Starts from an initial point and looks for an optimal solution along each coordinate direction iteratively.
- For a function with two independent variables *x* and *y*, starting at an initial point (x_0,y_0), the first iteration will first move along direction (1, 0) until an optimal solution is found for the function.
- The next search involves searching along the direction (0,1) to determine the optimal value for the function.
- Once searches in all directions are completed, the process is repeated in the next iteration and iterations continue until convergence occurs.
- The search along each coordinate direction can be conducted using anyone of the one-dimensional search techniques previously covered.



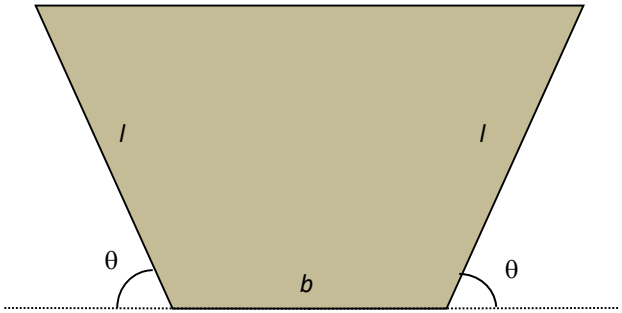
Método cíclico coordinado - ejemplo

- The cross-sectional area A of a gutter with base length *b* and edge length of *l* is given by $A = \frac{1}{2}(b + b + 2l\cos\theta)l\sin\theta$

  Assuming that the width of material to be bent into the gutter shape is 6, find the angle θ and edge length *l* which maximizes the cross-sectional area of the gutter.

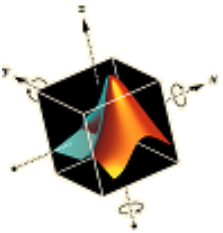
Solution: Recognizing that the base length *b* can be expressed as $b = 6 - 2l$

We can re-write the area function as $f(l,\theta) = (6 - 2l + l\cos\theta)l\sin\theta$



Use $(0, \pi/4)$ as the initial estimate of the solution and use Golden Search method to determine optimal solution in each dimension.

To use the golden search method we will use 0 and 3 as the lower and upper bounds for the search region.

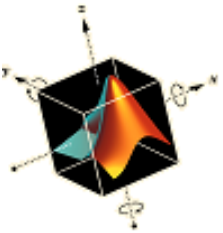


Método cíclico coordinado - ejemplo

Iteration 1 along (1, 0)

Iteration	x_l	x_u	x_1	x_2	$f(x_1)$	$f(x_2)$	ε
1	0.0000	3.0000	1.8541	1.1459	3.6143	2.6941	3.0000
2	1.1459	3.0000	2.2918	1.8541	3.8985	3.6143	1.8541
3	1.8541	3.0000	2.5623	2.2918	3.9655	3.8985	1.1459
4	2.2918	3.0000	2.7295	2.5623	3.9654	3.9655	0.7082
5	2.2918	2.7295	2.5623	2.4590	3.9655	3.9497	0.4377
6	2.4590	2.7295	2.6262	2.5623	3.9692	3.9655	0.2705
7	2.5623	2.7295	2.6656	2.6262	3.9692	3.9692	0.1672
8	2.5623	2.6656	2.6262	2.6018	3.9692	3.9683	0.1033
9	2.6018	2.6656	2.6412	2.6262	3.9694	3.9692	0.0639
10	2.6262	2.6656	2.6506	2.6412	3.9694	3.9694	0.0395

The maximum area of 3.6964 is obtained at point $(2.6459, \frac{22}{42})$

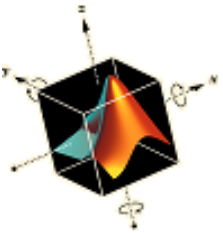


Método cíclico coordinado - ejemplo

Iteration 1 along (0, 1)

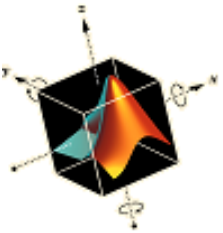
Iteration	x_l	x_u	x_1	x_2	$f(x_1)$	$f(x_2)$	ε
1	0.0000	1.5714	0.9712	0.6002	4.8084	4.3215	1.5714
2	0.6002	1.5714	1.2005	0.9712	4.1088	4.8084	0.9712
3	0.6002	1.2005	0.9712	0.8295	4.8084	4.8689	0.6002
4	0.6002	0.9712	0.8295	0.7419	4.8689	4.7533	0.3710
5	0.7419	0.9712	0.8836	0.8295	4.8816	4.8689	0.2293
6	0.8295	0.9712	0.9171	0.8836	4.8672	4.8816	0.1417
7	0.8295	0.9171	0.8836	0.8630	4.8816	4.8820	0.0876
8	0.8295	0.8836	0.8630	0.8502	4.8820	4.8790	0.0541
9	0.8502	0.8836	0.8708	0.8630	4.8826	4.8820	0.0334

The maximum area of 4.8823 is obtained at point (2.6459,0.87)



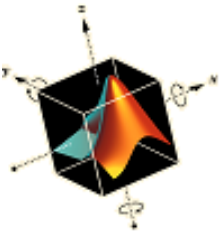
Método cíclico coordinado - ejemplo

- Since this is a two-dimensional search problem, the two searches along the two dimensions completes the first iteration.
- In the next iteration we return to the first dimension for which we conducted a search and start the second iteration with a search along this dimension.
- After the fifth cycle, the optimal solution of (2.0016, 10420) with an area of 5.1960 is obtained.
- The optimal solution to the problem is exactly 60 degrees which is 1.0472 radians and an edge and base length of 2 inches. The area of the gutter at this point is 5.1962.



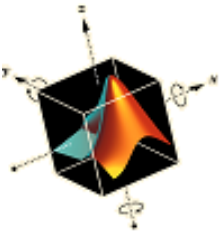
Método de gradiente multidimensional

- Use information from the derivatives of the optimization function to guide the search.
- Finds solutions quicker compared with direct search methods.
- A good initial estimate of the solution is required.
- The objective function needs to be differentiable.



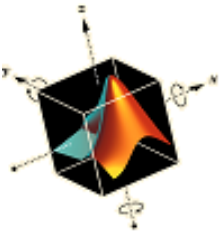
Gradiente

- The *gradient* is a vector operator denoted by ∇ (nabla symbol)
- When applied to a function , it represents the functions directional derivatives
- The gradient is the special case where the direction of the gradient is the direction of most or the *steepest ascent/descent*
- The gradient is calculated by $\nabla f = \frac{\partial f}{\partial x}\mathbf{i} + \frac{\partial f}{\partial y}\mathbf{j}$



Gradiente

- The *gradient* is a vector operator denoted by ∇ (nabla symbol)
- When applied to a function , it represents the functions directional derivatives
- The gradient is the special case where the direction of the gradient is the direction of most or the *steepest ascent/descent*
- The gradient is calculated by $\nabla f = \frac{\partial f}{\partial x}\mathbf{i} + \frac{\partial f}{\partial y}\mathbf{j}$



Gradiente - ejemplo

Calculate the gradient to determine the direction of the steepest slope at point (2, 1) for the function

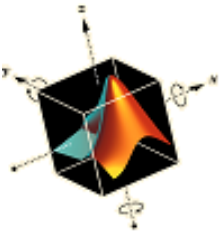
$$f(x, y) = x^2 y^2$$

Solution: To calculate the gradient we would need to calculate

$$\frac{\partial f}{\partial x} = 2xy^2 = 2(2)(1)^2 = 4 \qquad \frac{\partial f}{\partial y} = 2x^2 y = 2(2)^2(1) = 8$$

which are used to determine the gradient at point (2,1) as

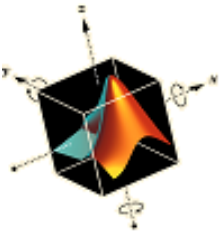
$$\nabla f = 4\mathbf{i} + 8\mathbf{j}$$



Hessiano

- The *Hessian* matrix or just the *Hessian* is the Jacobian matrix of second-order partial derivatives of a function.
- The determinant of the Hessian matrix is also referred to as the Hessian.
- For a two dimensional function the Hessian matrix is simply

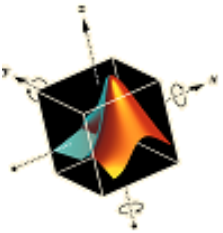
$$H = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix}$$



Hessiano

- The determinant of the Hessian matrix denoted by $|H|$ can have three cases:

1. If $|H| > 0$ and $\partial^2 f / \partial^2 x^2 > 0$ then $f(x, y)$ has a local minimum.
2. If $|H| > 0$ and $\partial^2 f / \partial^2 x^2 < 0$ then $f(x, y)$ has a local maximum.
3. If $|H| < 0$ then $f(x, y)$ has a saddle point.



Hessiano - ejemplo

Calculate the hessian matrix at point (2, 1) for the function $f(x,y) = x^2 y^2$

Solution: To calculate the Hessian matrix; the partial derivatives must be evaluated as

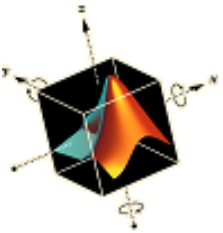
$$\frac{\partial^2 f}{\partial x^2} = 2y^2 = 2(1)^2 = 4$$

$$\frac{\partial^2 f}{\partial y^2} = 2x^2 = 2(2)^2 = 8$$

$$\frac{\partial^2 f}{\partial x \partial y} = \frac{\partial^2 f}{\partial y \partial x} = 4xy = 4(2)(1) = 8$$

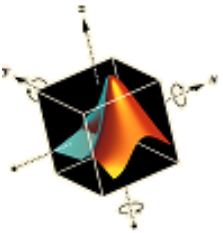
resulting in the Hessian matrix

$$H = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix} = \begin{bmatrix} 4 & 8 \\ 8 & 8 \end{bmatrix}$$



Método de ascenso/descenso rápido

- Starts from an initial point and looks for a local optimal solution along a gradient.
- The gradient at the initial solution is calculated.
- A new solution is found at the local optimum along the gradient.
- The subsequent iterations involve using the local optima along the new gradient as the initial point.



Método de ascenso/descenso rápido - ejemplo

Determine the minimum of the function $f(x,y)=x^2+y^2+2x+4$

Use the point (2,1) as the initial estimate of the optimal solution.

Solution:

Iteration 1: To calculate the gradient the partial derivatives must be evaluated as

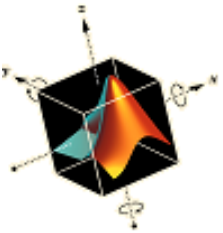
$$\frac{\partial f}{\partial x}=2x+2=2(2)+2=4 \qquad \frac{\partial f}{\partial y}=2y=2(1)=2 \qquad \nabla f=4\mathbf{i}+2\mathbf{j}$$

Now the function $f(x,y)$ can be expressed along the direction of gradient as

$$f\left(x_0+\frac{\partial f}{\partial x}h, y_0+\frac{\partial f}{\partial y}h\right)=f(2+4h,1+2h)=(2+4h)^2+(1+2h)^2+2(2+4h)+4$$

$$g(h)=20h^2+28h+13$$

This is a simple function and it is easy to determine $h^*=-0.7$ by taking the first derivative and solving for its roots.



Método de ascenso/descenso rápido - ejemplo

This means that traveling a step size of $h = -0.7$ along the gradient reaches a minimum value for the function in this direction. These values are substituted back to calculate a new value for x and y as follows:

$$x = 2 + 4(-0.7) = -0.8 \qquad y = 1 + 2(-0.7) = -0.4$$

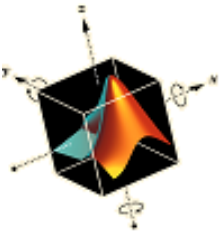
Note that: $f(2,1) = 13 \quad f(-0.8,-0.4) = 3.2$

Iteration 2: The new initial point is (-0.8, -0.4).We calculate the gradient at this point as $\frac{\partial f}{\partial x} = 2x + 2 = 2(-0.8) + 2 = 0.4 \qquad \frac{\partial f}{\partial y} = 2y = 2(-0.4) = -0.8$

$$\nabla f = 0.4\mathbf{i} - 0.8\mathbf{j}$$

$$f\left(x_0 + \frac{\partial f}{\partial x}h, y_0 + \frac{\partial f}{\partial y}h\right) = f(-0.8 + 0.4h, -0.4 - 0.8h)$$

$$= (-0.8 + 0.4h)^2 + (0.4 - 0.8h)^2 + 2(-0.8 + 0.4h) + 4$$



Método de ascenso/descenso rápido - ejemplo

$$g(h) = 0.8h^2 + 0.8h + 3.2 \qquad h^* = -0.5$$

$$x = -0.8 + 0.4(-0.5) = -1$$

$$y = -0.4 - 0.8(-0.5) = 0$$

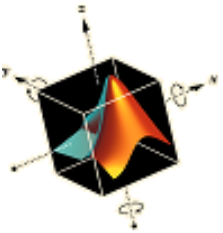
$$f(-0.8,-0.4) = 3.2 \qquad f(-1,0) = 3$$

Iteration 3: The new initial point is (-1, 0).We calculate the gradient at this point as

$$\frac{\partial f}{\partial x} = 2x + 2 = 2(-1) + 2 = 0 \qquad \frac{\partial f}{\partial y} = 2y = 2(0) = 0$$

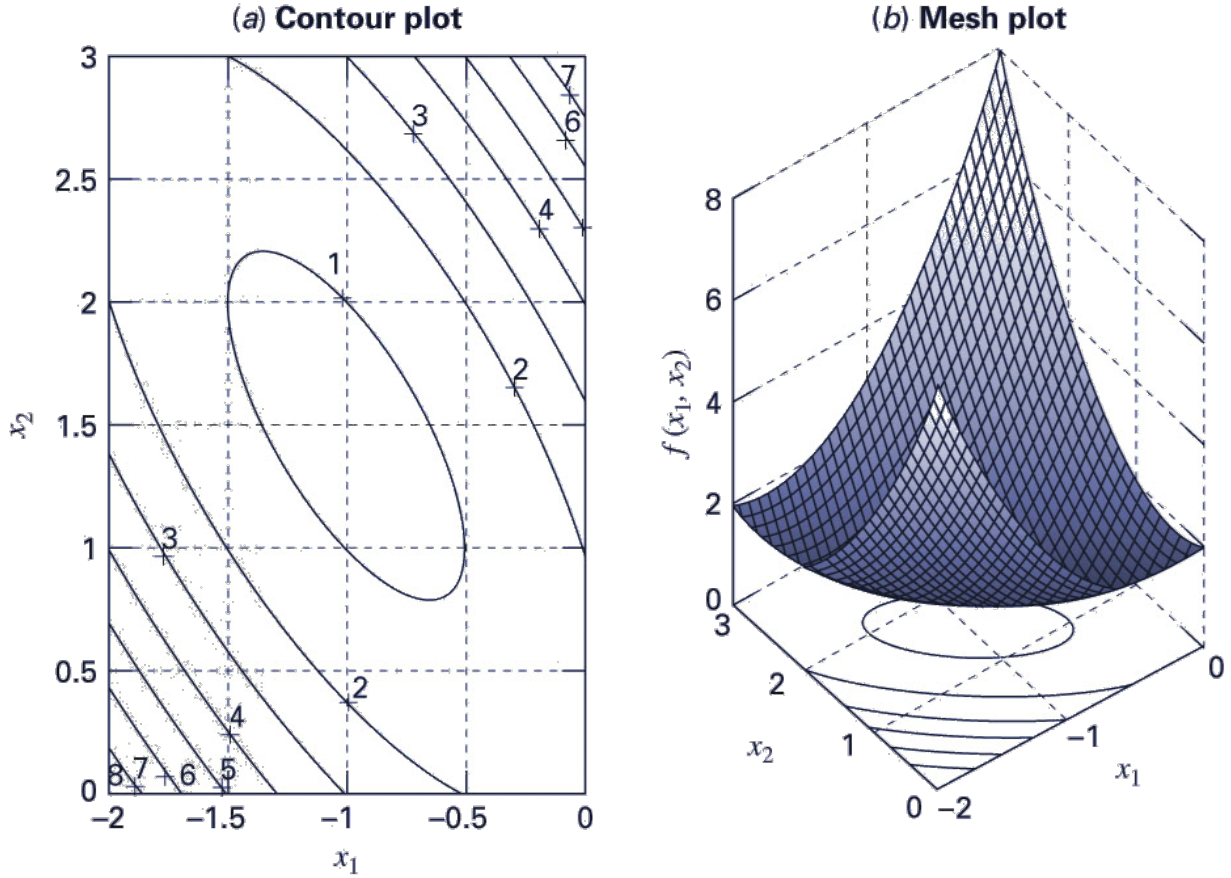
$$\nabla f = 0\mathbf{i} + 0\mathbf{j}$$

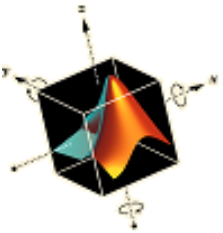
This indicates that the current location is a local optimum along this gradient and no improvement can be gained by moving in any direction. The minimum of the function is at point (-1,0).



Visualización multidimensional

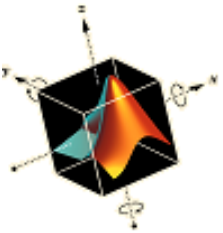
- Functions of two-dimensions may be visualized using contour or surface/mesh plots.





Función Matlab `fminsearch`

- MATLAB has a built-in function, `fminsearch`, that can be used to determine the minimum of a multidimensional function.
  - `[`*`xmin,fval`*`]=fminsearch(`*`function, x0`*`)`
  - *`xmin`* in this case will be a row vector containing the location of the minimum, while *`x0`* is an initial guess. Note that *`x0`* must contain as many entries as the function expects of it.
- The `function` must be written in terms of a single variable, where different dimensions are represented by different indices of that variable.

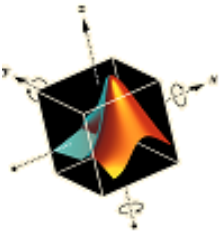


Función Matlab `fminsearch`

- To minimize
  $f(x,y)=2+x-y+2x^2+2xy+y^2$
  rewrite as
  $f(x_1, x_2)=2+x_1-x_2+2(x_1)^2+2x_1x_2+(x_2)^2$

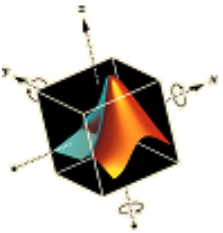
```
f=@(x) 2+x(1)-x(2)+2*x(1)^2+2*x(1)*x(2)+x(2)^2
[x, fval] = fminsearch(f, [-0.5, 0.5])
```

- Note that *x0* has two entries - *f* is expecting it to contain two values.
- MATLAB reports the minimum value is 0.7500 at a location of [-1.000 1.5000]



Programación lineal

- Linear programming is an optimization approach that deals with problems that have specific *constraints*.
- The problems previously discussed did not consider any constraints on the values of the independent variables.
- In linear programming, the independent variables which are frequently used to model concepts such as availability of resources or required ratio of resources are constrained to be more than, less than or equal to a specific value.



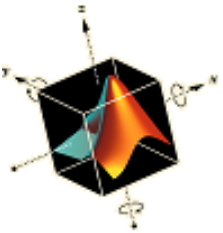
Programación lineal

- The simplest linear program requires an objective function and a set of constraints.
- The objective function is expressed as (for a maximization problem)

$$\max\ z = c_1 x_1 + c_2 x_2 + \ldots + c_n x_n$$

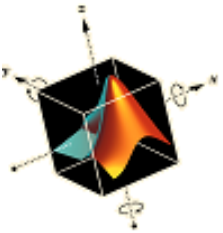
- The constraints are also a linear combination of the form

$$a_{i1} x_1 + a_{i2} x_2 + \ldots + a_{in} x_n \leq b_i$$



Programación lineal - ejemplo

- A Company produces inexpensive tables and chairs.
- Processes require a certain amount of hours of carpentry work and in the painting and varnishing department.
- Each table takes 4 hours of carpentry and 2 hours of painting and varnishing.
- Each chair requires 3 of carpentry and 1 hour of painting and varnishing.
- There are 240 hours of carpentry time available and 100 hours of painting and varnishing.
- Each table yields a profit of $70 and each chair a profit of $50.
- The company wants to determine the best combination of tables and chairs to produce to reach the maximum profit.



Programación lineal - ejemplo

Equations:

The decision variables in this problem are

T = number of tables to be produced per week

C = number of chairs to be produced per week

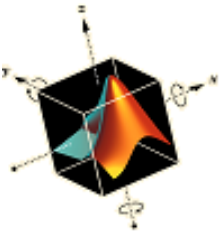
The objective function is: $\max\ z = 70T + 50C$

Constraints:

$4T + 3C \leq 240$ hours of carpentry time

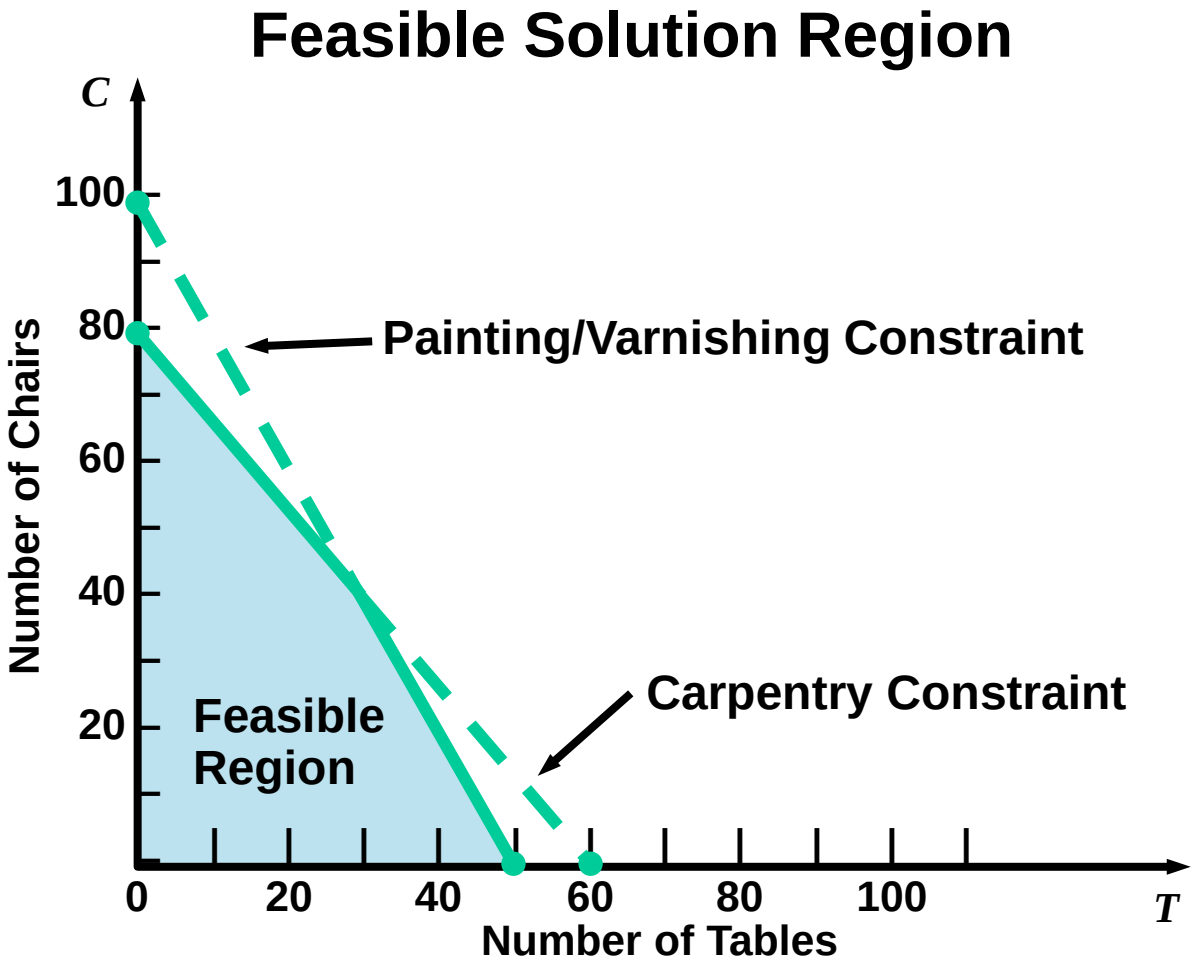
$2T + 1C \leq 100$ hours of varnishing time

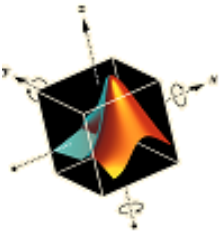
$T, C \geq 0$ non negative



Programación lineal - ejemplo

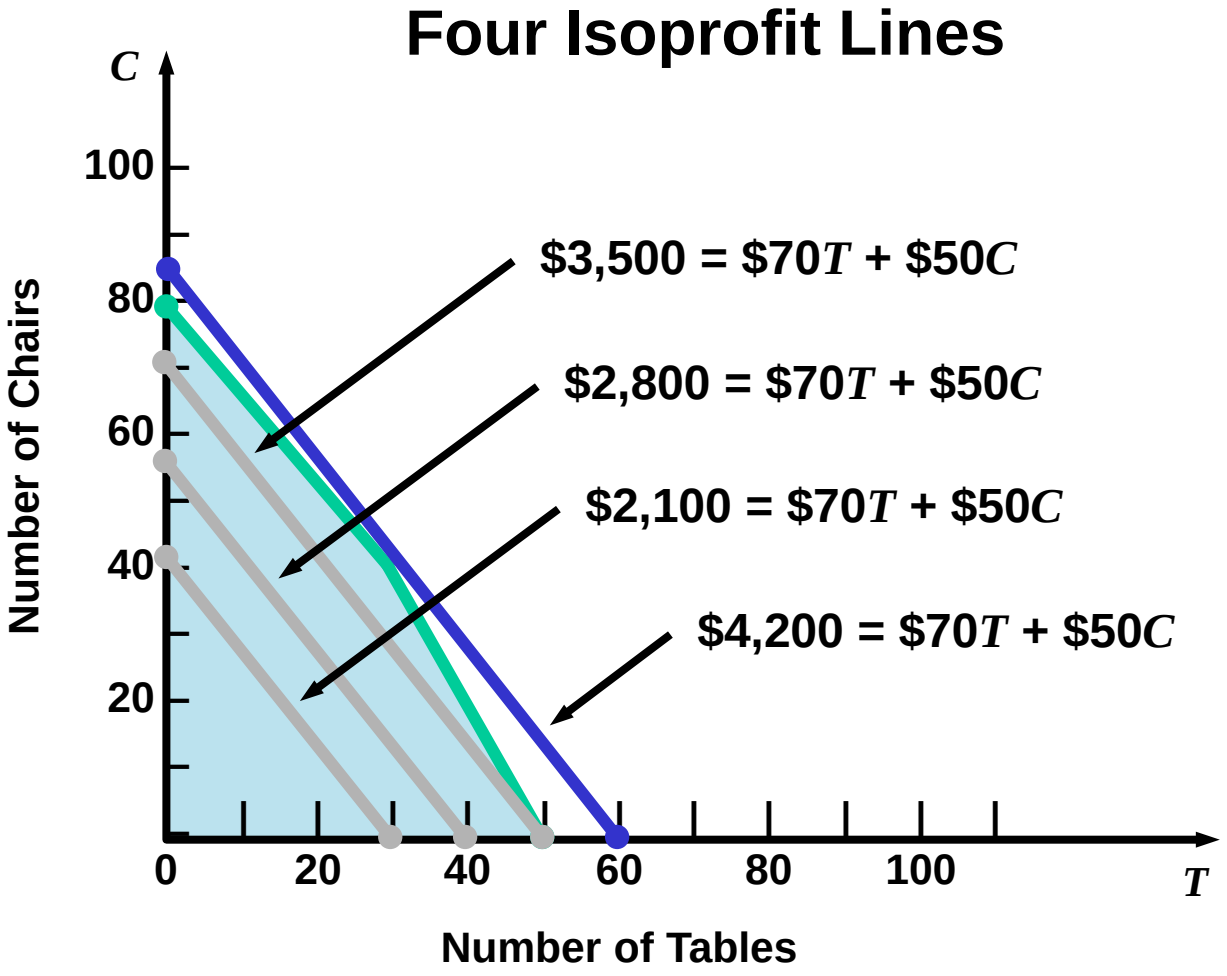
Graphical solution:

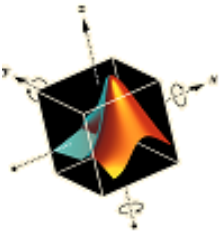




Programación lineal - ejemplo

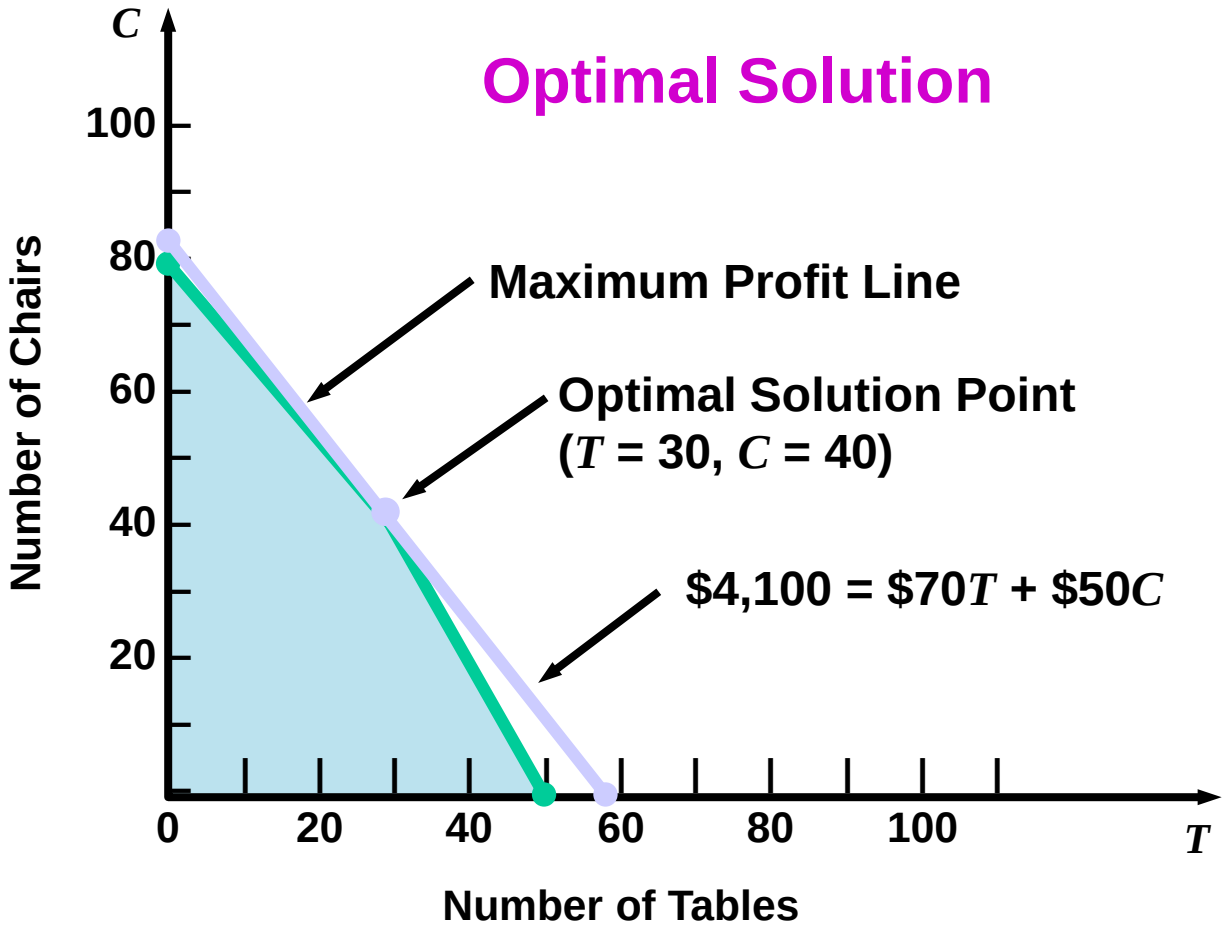
Graphical solution by isoprofit line method:

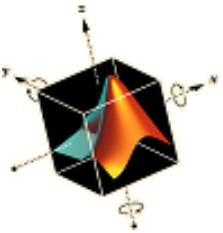




Programación lineal - ejemplo

Graphical solution by isoprofit line method :

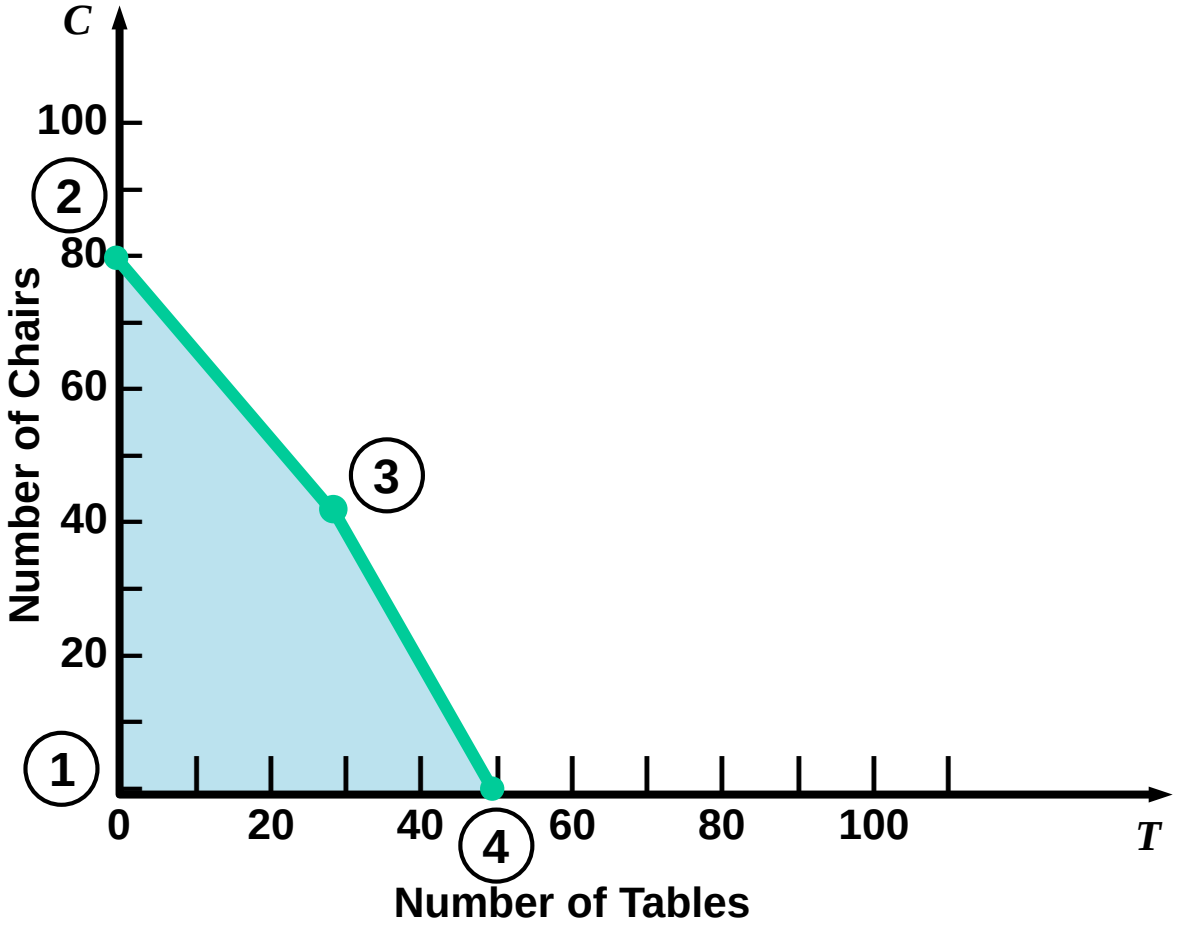


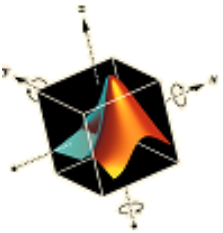


Programación lineal - ejemplo

Solution by corner point method:

It involves looking at the profit at every corner point of the feasible region.
The mathematical theory behind LP is that the optimal solution must lie at one of the corner points, or extreme point, in the feasible region.



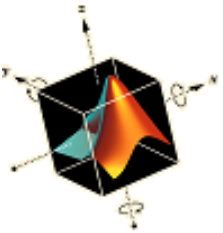


Programación lineal - ejemplo

Solution by corner point method:

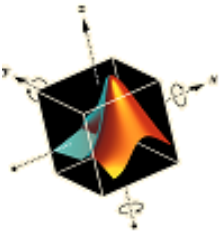
Point ① ($T = 0$, $C = 0$)	Profit = \$70(0) + \$50(0) = \$0
Point ② ($T = 0$, $C = 80$)	Profit = \$70(0) + \$50(80) = \$4,000
Point ④ ($T = 50$, $C = 0$)	Profit = \$70(50) + \$50(0) = \$3,500
Point ③ ($T = 30$, $C = 40$)	Profit = \$70(30) + \$50(40) = \$4,100

Because Point ③ returns the highest profit, this is the optimal solution.



Holgura y excedente

- **Slack** is the amount of a resource that is not used. For a less-than-or-equal constraint:
  - Slack = Amount of resource available – amount of resource used.
- **Surplus** is used with a greater-than-or-equal constraint to indicate the amount by which the right hand side of the constraint is exceeded.
  - Surplus = Actual amount – minimum amount.



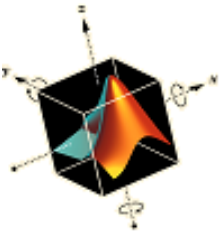
El método Simplex

- The graphical method is useful for problems with two decision variables and few problem constraints.
- For more decision variables and more problem constraints it is used an algebraic method called the *simplex method*, developed by G. Dantzig in 1947.
- The standard form of a maximization problem is:

Maximize of: $Z_{\max} = P = c_1 x_1 + c_2 x_2 + \ldots + c_n x_n$

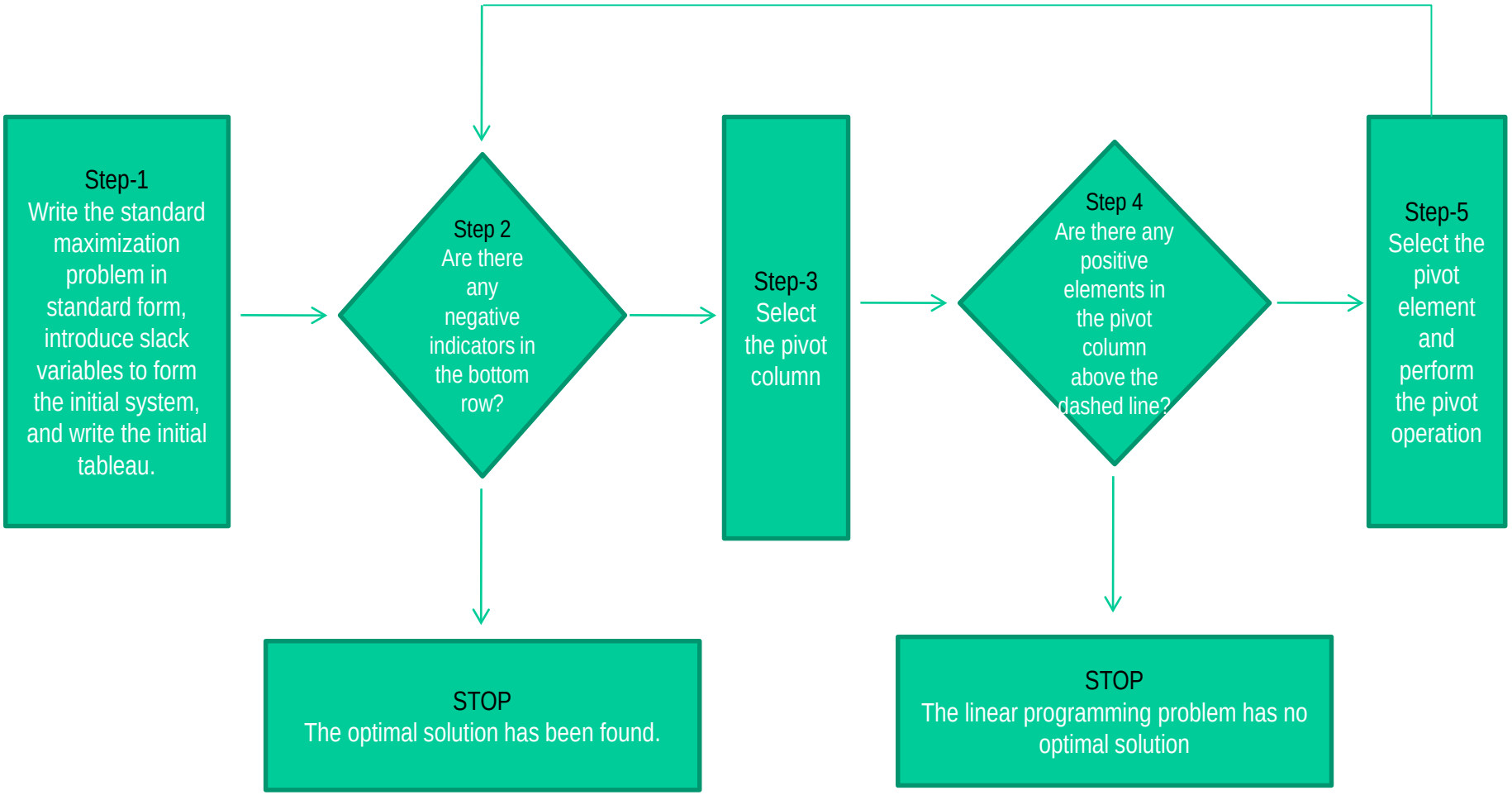
Constraints: $a_1 x_1 + a_2 x_2 + \ldots + a_n x_n \leq b \quad , b \geq 0$

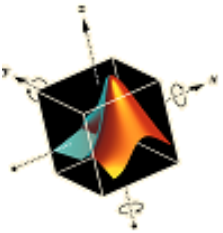
$x_1, x_2, \ldots, x_n \geq 0$



El método Simplex

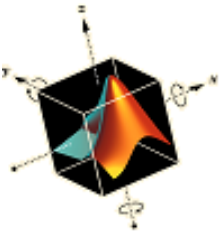
Simplex algorithm for standard maximization problems





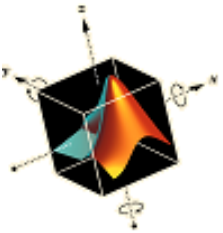
Algoritmo del método Simplex

1. Convert each inequality in the set of constraints to an equation by adding slack variables.
2. Create the initial simplex tableau.
3. Select the pivot column. (The column with the most negative value element in the last row.)
4. Select the pivot row. (The row with the smallest non-negative result when the last element in the row is divided by the corresponding in the pivot column.)
5. Use elementary row operations calculate new values for the pivot row so that the pivot is 1 (Divide every number in the row by the pivot number.)
6. Use elementary row operations to make all numbers in the pivot column equal to 0 except for the pivot number. If all entries in the bottom row are zero or positive, this the final tableau. If not, go back to step 3.
7. If you obtain a final tableau, then the linear programming problem has a maximum solution, which is given by the entry in the lower-right corner of the tableau.

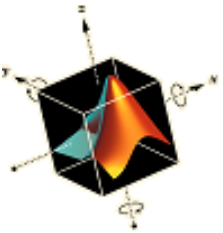


Aplicaciones de la programación lineal

- The Transportation Problem
- The Assignment Problem
- The Transshipment Problem
- Facility Location Analysis



Sistemas de ecuaciones lineales



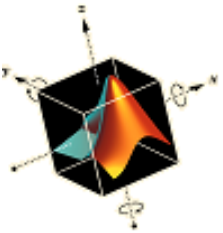
Matrices

- A *matrix* consists of a rectangular array of elements represented by a single symbol (example: [*A*]).
- An individual entry of a matrix is an *element* (example: a_{23})

Column 3

$$[A] = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ \cdot & \cdot & \cdot & & \cdot \\ \cdot & \cdot & \cdot & & \cdot \\ \cdot & \cdot & \cdot & & \cdot \\ a_{m1} & a_{m2} & a_{m3} & \cdots & a_{mn} \end{bmatrix}$$

← Row 2



Matrices especiales

- Matrices where *m=n* are called *square matrices*.
- There are a number of special forms of square matrices:

Symmetric

$$[A]=\begin{bmatrix}5 & 1 & 2\\ 1 & 3 & 7\\ 2 & 7 & 8\end{bmatrix}$$

Diagonal

$$[A]=\begin{bmatrix}a_{11} & & \\ & a_{22} & \\ & & a_{33}\end{bmatrix}$$

Identity

$$[A]=\begin{bmatrix}1 & & \\ & 1 & \\ & & 1\end{bmatrix}$$

Upper Triangular

$$[A]=\begin{bmatrix}a_{11} & a_{12} & a_{13}\\ & a_{22} & a_{23}\\ & & a_{33}\end{bmatrix}$$

Lower Triangular

$$[A]=\begin{bmatrix}a_{11} & & \\ a_{21} & a_{22} & \\ a_{31} & a_{32} & a_{33}\end{bmatrix}$$

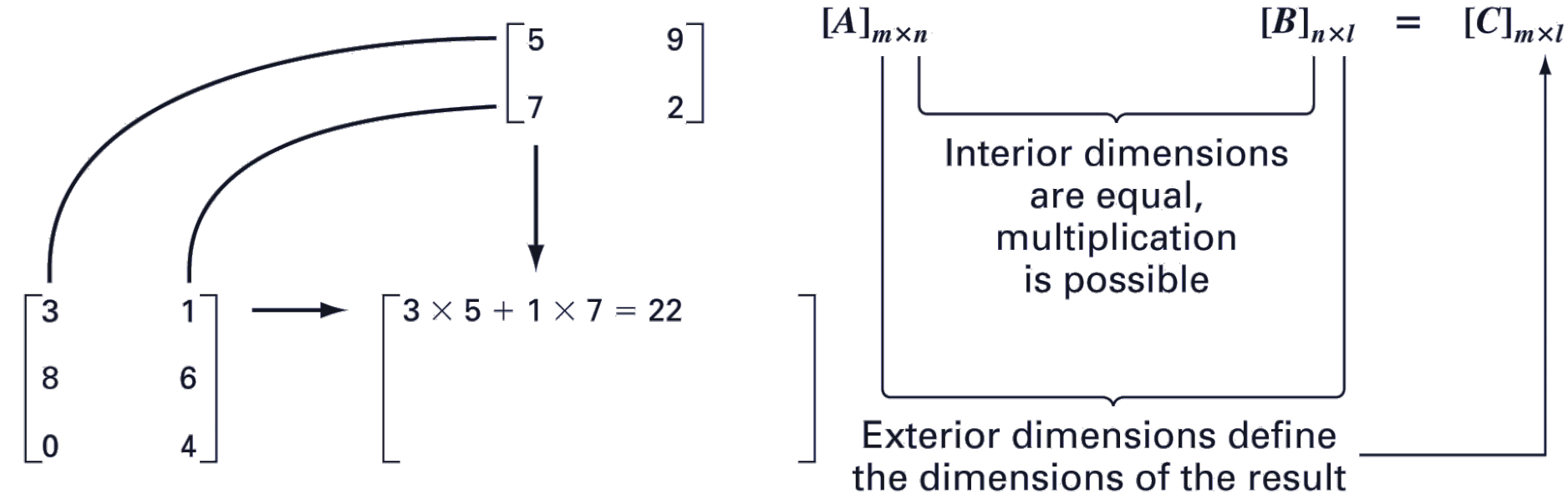
Banded

$$[A]=\begin{bmatrix}a_{11} & a_{12} & & \\ a_{21} & a_{22} & a_{23} & \\ & a_{32} & a_{33} & a_{34}\\ & & a_{43} & a_{44}\end{bmatrix}$$

Multiplicación matricial

- The elements in the matrix [C] that results from multiplying matrices [A] and [B] are calculated using:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$



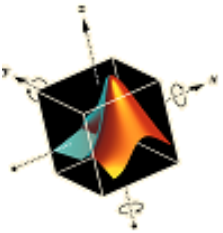
Representación de Algebra Lineal

- Matrices provide a concise notation for representing and solving simultaneous linear equations:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 &= b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 &= b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 &= b_3 \end{aligned}$$

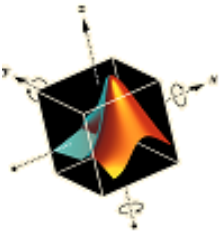
$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} b_1 \\ b_2 \\ b_3 \end{Bmatrix}$$

$$[A]\{x\} = \{b\}$$



Solución con Matlab

- MATLAB provides two direct ways to solve systems of linear algebraic equations [A]{x}={b}:
  - Left-division
    ```
    x = A\b
    ```
  - Matrix inversion
    ```
    x = inv(A)*b
    ```
- The matrix inverse is less efficient than left-division and also only works for square, non-singular systems.

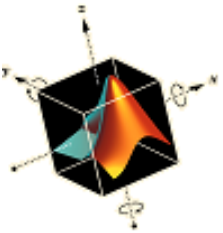


Determinantes

- The *determinant D*=|*A*| of a matrix is formed from the coefficients of [A].
- Determinants for small matrices are:

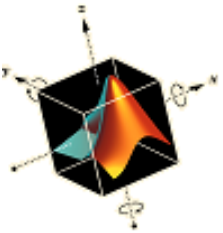
1×1	$\left\|a_{11}\right\| = a_{11}$
2×2	$\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} = a_{11}a_{22} - a_{12}a_{21}$
3×3	$\begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} = a_{11}\begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix} - a_{12}\begin{vmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{vmatrix} + a_{13}\begin{vmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{vmatrix}$

- Determinants for matrices larger than 3 x 3 can be very complicated.



Regla de Cramer

- *Cramer's Rule* states that each unknown in a system of linear algebraic equations may be expressed as a fraction of two determinants with denominator *D* and with the numerator obtained from *D* by replacing the column of coefficients of the unknown in question by the constants b_1, b_2, …, b_n.



Regla de Cramer - ejemplo

- Find x_2 in the following system of equations:

$$0.3x_1 + 0.52x_2 + x_3 = -0.01$$
$$0.5x_1 + x_2 + 1.9x_3 = 0.67$$
$$0.1x_1 + 0.3x_2 + 0.5x_3 = -0.44$$

- Find the determinant *D*

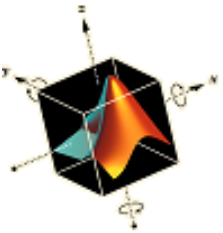
$$D = \begin{vmatrix} 0.3 & 0.52 & 1 \\ 0.5 & 1 & 1.9 \\ 0.1 & 0.3 & 0.5 \end{vmatrix} = 0.3\begin{vmatrix} 1 & 1.9 \\ 0.3 & 0.5 \end{vmatrix} - 0.52\begin{vmatrix} 0.5 & 1.9 \\ 0.1 & 0.5 \end{vmatrix} + 1\begin{vmatrix} 0.5 & 1 \\ 0.1 & 0.4 \end{vmatrix} = -0.0022$$

- Find determinant D_2 by replacing *D*'s second column with *b*

$$D_2 = \begin{vmatrix} 0.3 & -0.01 & 1 \\ 0.5 & 0.67 & 1.9 \\ 0.1 & -0.44 & 0.5 \end{vmatrix} = 0.3\begin{vmatrix} 0.67 & 1.9 \\ -0.44 & 0.5 \end{vmatrix} - 0.01\begin{vmatrix} 0.5 & 1.9 \\ 0.1 & 0.5 \end{vmatrix} + 1\begin{vmatrix} 0.5 & 0.67 \\ 0.1 & -0.44 \end{vmatrix} = 0.0649$$

- Divide

$$x_2 = \frac{D_2}{D} = \frac{0.0649}{-0.0022} = -29.5$$



Eliminación gaussiana

- For larger systems, Cramer's Rule can become unwieldy.
- Instead, a sequential process of removing unknowns from equations using *forward elimination* followed by *back substitution* may be used - this is Gauss elimination.
- "Naïve" Gauss elimination simply means the process does not check for potential problems resulting from division by zero.

Eliminación gaussiana

- Forward elimination
  - Starting with the first row, add or subtract multiples of that row to eliminate the first coefficient from the second row and beyond.
  - Continue this process with the second row to remove the second coefficient from the third row and beyond.
  - Stop when an upper triangular matrix remains.
- Back substitution
  - Starting with the *last* row, solve for the unknown, then substitute that value into the next highest row.
  - Because of the upper-triangular nature of the matrix, each row will contain only one more unknown.

$$\left[\begin{array}{ccc|c} a_{11} & a_{12} & a_{13} & b_1 \\ a_{21} & a_{22} & a_{23} & b_2 \\ a_{31} & a_{32} & a_{33} & b_3 \end{array}\right]$$

$$\downarrow$$

$$\left[\begin{array}{ccc|c} a_{11} & a_{12} & a_{13} & b_1 \\ & a'_{22} & a'_{23} & b'_2 \\ & & a''_{33} & b''_3 \end{array}\right]$$

(*a*) Forward elimination

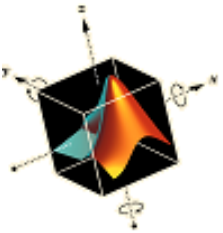
$$\downarrow$$

$$x_3 = b''_3/a''_{33}$$

$$x_2 = (b'_2 - a'_{23}x_3)/a'_{22}$$

$$x_1 = (b_1 - a_{13}x_3 - a_{12}x_2)/a_{11}$$

(*b*) Back substitution

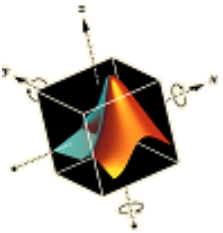


Eficiencia de la eliminación de Gauss

- The execution of Gauss elimination depends on the amount of *floating-point operations* (or flops). The flop count for an *n* x *n* system is:

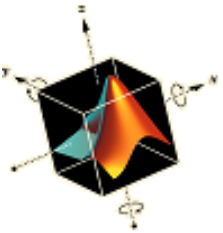
Forward Elimination	$\frac{2n^3}{3} + O(n^2)$
Back Substitution	$n^2 + O(n)$
Total	$\frac{2n^3}{3} + O(n^2)$

- Conclusions:
  - As the system gets larger, the computation time increases greatly.
  - Most of the effort is incurred in the elimination step.



Pivoteo

- Problems arise with naïve Gauss elimination if a coefficient along the diagonal is 0 (problem: division by 0) or close to 0 (problem: round-off error)
- One way to combat these issues is to determine the coefficient with the largest absolute value in the column below the pivot element. The rows can then be switched so that the largest element is the pivot element. This is called *partial pivoting.*
- If the rows to the right of the pivot element are also checked and columns switched, this is called *complete pivoting*.

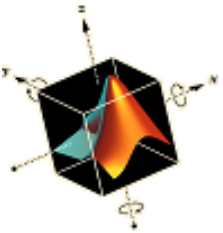


Sistemas tridiagonales

- A *tridiagonal* system is a banded system with a bandwidth of 3:

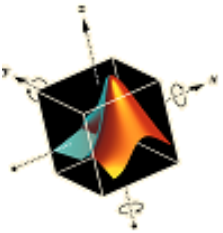
$$\begin{bmatrix} f_1 & g_1 & & & & & & \\ e_2 & f_2 & g_2 & & & & & \\ & e_3 & f_3 & g_3 & & & & \\ & & \cdot & \cdot & \cdot & & & \\ & & & \cdot & \cdot & \cdot & & \\ & & & & \cdot & \cdot & \cdot & \\ & & & & & e_{n-1} & f_{n-1} & g_{n-1} \\ & & & & & & e_n & f_n \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \\ \cdot \\ \cdot \\ \cdot \\ x_{n-1} \\ x_n \end{Bmatrix} = \begin{Bmatrix} r_1 \\ r_2 \\ r_3 \\ \cdot \\ \cdot \\ \cdot \\ r_{n-1} \\ r_n \end{Bmatrix}$$

- Tridiagonal systems can be solved using the same method as Gauss elimination, but with much less effort because most of the matrix elements are already 0.



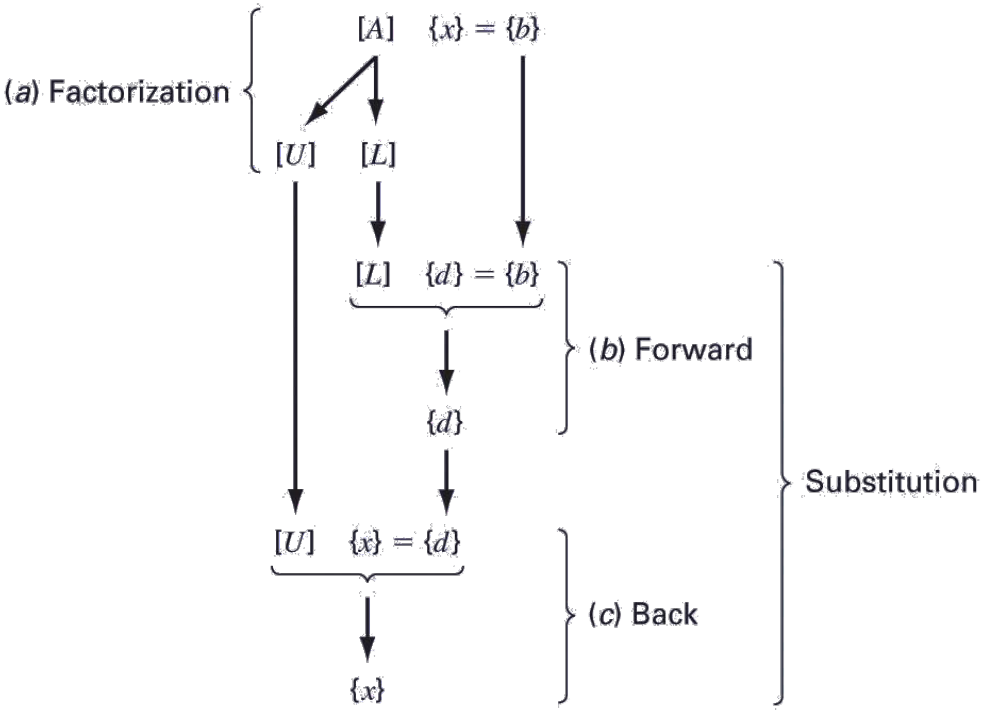
Factorización LU

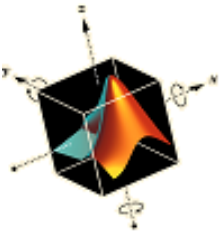
- Recall that the forward-elimination step of Gauss elimination comprises the bulk of the computational effort.
- *LU* factorization methods separate the time-consuming elimination of the matrix [*A*] from the manipulations of the right-hand-side [*b*].
- Once [*A*] has been factored (or decomposed), multiple right-hand-side vectors can be evaluated in an efficient manner.



Factorización LU

- *LU* factorization involves two steps:
  - Factorization to decompose the [*A*] matrix into a product of a lower triangular matrix [*L*] and an upper triangular matrix [*U*]. [*L*] has 1 for each entry on the diagonal.
  - Substitution to solve for {*x*}
- Gauss elimination can be implemented using *LU* factorization

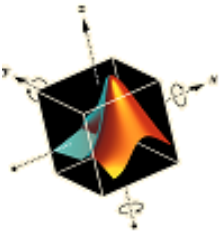




Eliminación de Gauss como Factorización LU

- [*A*]{*x*}={*b*} can be rewritten as [*L*][*U*]{*x*}={*b*} using *LU* factorization.
- The *LU* factorization algorithm requires the same total flops as for Gauss elimination.
- The main advantage is once [*A*] is decomposed, the same [*L*] and [*U*] can be used for multiple {*b*} vectors.
- MATLAB's `lu` function can be used to generate the [*L*] and [*U*] matrices:

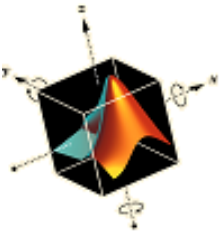
```
[L, U] = lu(A)
```



Eliminación de Gauss como Factorización LU

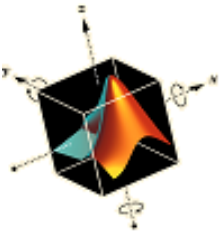
- To solve [*A*]{*x*}={*b*}, first decompose [*A*] to get [*L*][*U*]{*x*}={*b*}
- Set up and solve [*L*]{*d*}={*b*}, where {*d*} can be found using *forward* substitution.
- Set up and solve [*U*]{*x*}={*d*}, where {*x*} can be found using *backward* substitution.
- In MATLAB:

```
[L, U] = lu(A)
d = L\b
x = U\d
```



Factorización de Cholesky

- Symmetric systems occur commonly in both mathematical and engineering/science problem contexts, and there are special solution techniques available for such systems.
- The *Cholesky factorization* is one of the most popular of these techniques, and is based on the fact that a symmetric matrix can be decomposed as $[A]= [U]^T[U]$, where T stands for transpose.
- The rest of the process is similar to *LU* decomposition and Gauss elimination, except only one matrix, [*U*], needs to be stored.

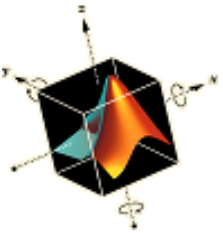


Factorización de Cholesky en Matlab

- MATLAB can perform a Cholesky factorization with the built-in `chol` command:

```
U = chol(A)
```

- MATLAB's left division operator \ examines the system to see which method will most efficiently solve the problem. This includes trying banded solvers, back and forward substitutions, Cholesky factorization for symmetric systems. If these do not work and the system is square, Gauss elimination with partial pivoting is used.

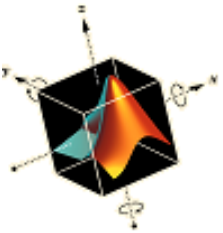


Matriz inversa

- Recall that if a matrix [*A*] is square, there is another matrix $[A]^{-1}$, called the inverse of [*A*], for which $[A][A]^{-1}=[A]^{-1}[A]=[I]$
- The inverse can be computed in a column by column fashion by generating solutions with unit vectors as the right-hand-side constants:

$$[A]\{x_1\}=\begin{Bmatrix}1\\0\\0\end{Bmatrix} \quad [A]\{x_2\}=\begin{Bmatrix}0\\1\\0\end{Bmatrix} \quad [A]\{x_3\}=\begin{Bmatrix}0\\0\\1\end{Bmatrix}$$

$$[A]^{-1}=\begin{bmatrix}x_1 & x_2 & x_3\end{bmatrix}$$

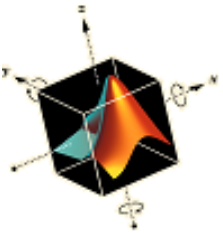


Matriz inversa y sistemas estímulo - respuesta

- Recall that *LU* factorization can be used to efficiently evaluate a system for multiple right-hand-side vectors - thus, it is ideal for evaluating the multiple unit vectors needed to compute the inverse.
- Many systems can be modeled as a linear combination of equations, and thus written as a matrix equation:

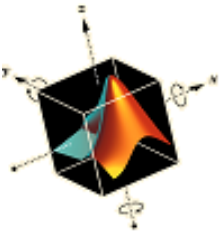
$$[\text{Interactions}]\{\text{response}\} = \{\text{stimuli}\}$$

- The system response can thus be found using the matrix inverse.



Normas vectoriales y matriciales

- A *norm* is a real-valued function that provides a measure of the size or “length” of multi-component mathematical entities such as vectors and matrices.
- Vector norms and matrix norms may be computed differently.



Normas vectoriales

- For a vector {*X*} of size *n*, the *p*-norm is:

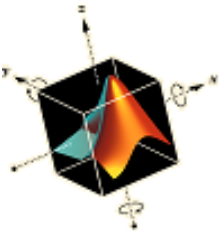
$$\|X\|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p}$$

- Important examples of vector *p*-norms include:

$p = 1$: sum of the absolute values $\quad \|X\|_1 = \sum_{i=1}^{n} |x_i|$

$p = 2$: Euclidian norm (length) $\quad \|X\|_2 = \|X\|_e = \sqrt{\sum_{i=1}^{n} x_i^2}$

$p = \infty$: maximum – magnitude $\quad \|X\|_\infty = \max_{1 \le i \le n} |x_i|$



Normas matriciales

- Common matrix norms for a matrix [*A*] include:

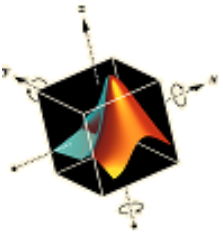
column - sum norm $\quad \|A\|_1 = \max_{1 \le j \le n} \sum_{i=1}^{n} |a_{ij}|$

Frobenius norm $\quad \|A\|_f = \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij}^2}$

row - sum norm $\quad \|A\|_\infty = \max_{1 \le i \le n} \sum_{j=1}^{n} |a_{ij}|$

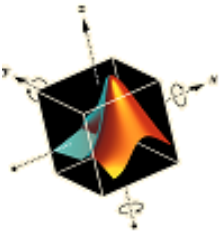
spectral norm (2 norm) $\quad \|A\|_2 = (\mu_{\max})^{1/2}$

- Note - μ_{max} is the largest eigenvalue of $[A]^T[A]$.



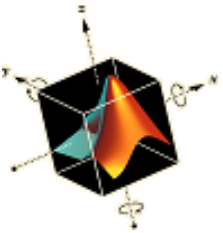
Número de condición de una matriz

- The *matrix condition number* Cond[*A*] is obtained by calculating Cond[*A*]=||*A*||·||A^{-1}||
- In can be shown that: $\frac{\|\Delta X\|}{\|X\|} \leq \text{Cond}[A] \frac{\|\Delta A\|}{\|A\|}$
- The relative error of the norm of the computed solution can be as large as the relative error of the norm of the coefficients of [*A*] multiplied by the condition number.
- If the coefficients of [*A*] are known to *t* digit precision, the solution [*X*] may be valid to only *t*-$\log_{10}$(Cond[*A*]) digits.



Comandos Matlab

- MATLAB has built-in functions to compute both norms and condition numbers:
  - `norm(`*X*`,`*p*`)`
    - Compute the *p* norm of vector *X*, where *p* can be any number, `inf`, or `'fro'` (for the Euclidean norm)
  - `norm(`*A*`,`*p*`)`
    - Compute a norm of matrix *A*, where *p* can be 1, 2, `inf`, or `'fro'` (for the Frobenius norm)
  - `cond(`*X*`,`*p*`)` or `cond(`*A*`,`*p*`)`
    - Calculate the condition number of vector *X* or matrix *A* using the norm specified by *p*.



Método iterativo: Gauss - Seidel

- The *Gauss-Seidel method* is the most commonly used iterative method for solving linear algebraic equations [*A*]{*x*}={*b*}.
- The method solves each equation in a system for a particular variable, and then uses that value in later equations to solve later variables. For a 3x3 system with nonzero elements along the diagonal, for example, the j^{th} iteration values are found from the $j-1^{th}$ iteration using:

$$x_1^j = \frac{b_1 - a_{12}x_2^{j-1} - a_{13}x_3^{j-1}}{a_{11}}$$

$$x_2^j = \frac{b_2 - a_{21}x_1^{j} - a_{23}x_3^{j-1}}{a_{22}}$$

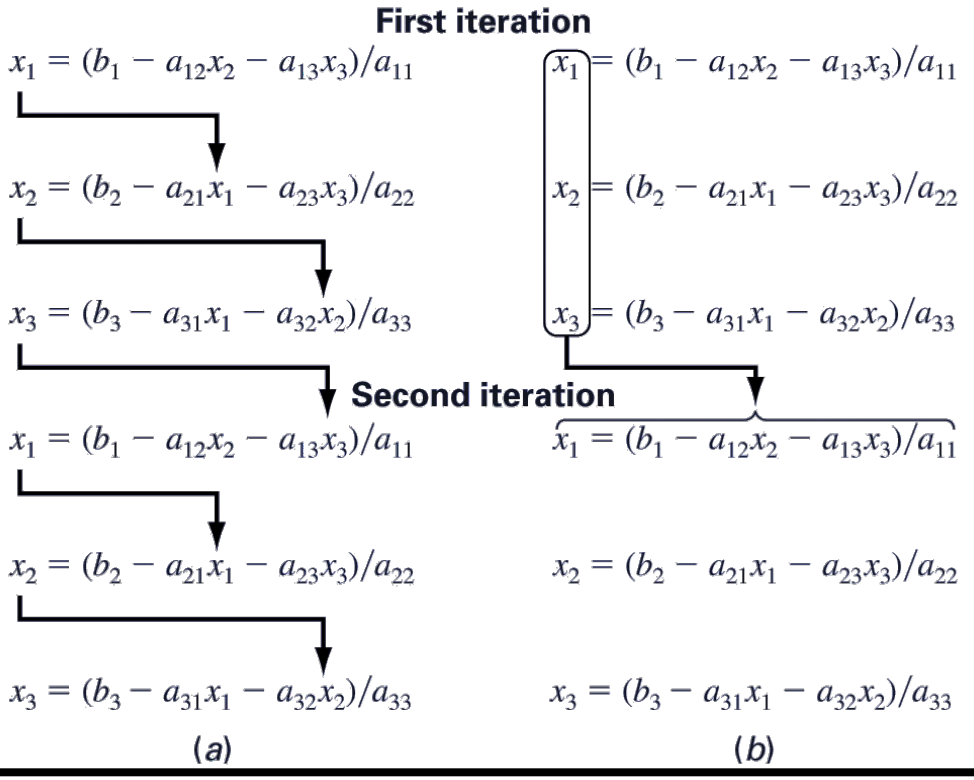
$$x_3^j = \frac{b_3 - a_{31}x_1^{j} - a_{32}x_2^{j}}{a_{33}}$$

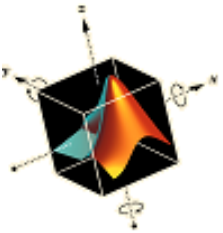
Iteración de Jacobi

- The *Jacobi iteration* is similar to the Gauss-Seidel method, except the j-1th information is used to update all variables in the jth iteration:

a) Gauss-Seidel

b) Jacobi



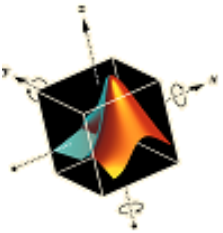


Convergencia

- The convergence of an iterative method can be calculated by determining the relative percent change of each element in {*x*}. For example, for the *i*th element in the *j*th iteration,

$$\varepsilon_{a,i} = \left| \frac{x_i^j - x_i^{j-1}}{x_i^j} \right| \times 100\%$$

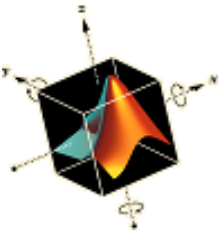
- The method is ended when all elements have converged to a set tolerance.



Dominancia diagonal

- The Gauss-Seidel method may diverge, but if the system is *diagonally dominant*, it will definitely converge.
- Diagonal dominance means:

$$|a_{ii}| > \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{ij}|$$



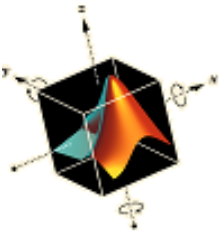
Relajación

- To enhance convergence, an iterative program can introduce *relaxation* where the value at a particular iteration is made up of a combination of the old value and the newly calculated value:

$$x_i^{\text{new}} = \lambda x_i^{\text{new}} + (1-\lambda)x_i^{\text{old}}$$

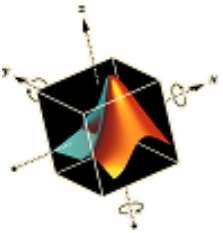
where λ is a weighting factor that is assigned a value between 0 and 2.

  - $0<\lambda<1$: underrelaxation
  - $\lambda=1$: no relaxation
  - $1<\lambda\leq 2$: overrelaxation



Sistemas no lineales

- Nonlinear systems can also be solved using the same strategy as the Gauss-Seidel method - solve each system for one of the unknowns and update each unknown using information from the previous iteration.
- This is called *successive substitution*.



Newton - Raphson

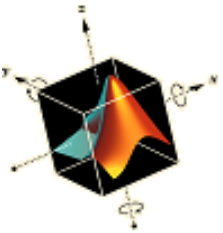
- Nonlinear systems may also be solved using the Newton-Raphson method for multiple variables.
- For a two-variable system, the Taylor series approximation and resulting Newton-Raphson equations are:

$$f_{1,i+1} = f_{1,i} + \left(x_{1,i+1} - x_{1,i}\right)\frac{\partial f_{1,i}}{\partial x_1} + \left(x_{2,i+1} - x_{2,i}\right)\frac{\partial f_{1,i}}{\partial x_2}$$

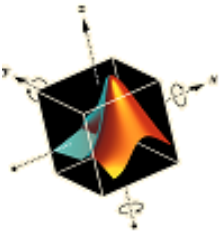
$$x_{1,i+1} = x_{1,i} - \frac{f_{1,i}\frac{\partial f_{2,i}}{\partial x_2} - f_{2,i}\frac{\partial f_{1,i}}{\partial x_2}}{\frac{\partial f_{1,i}}{\partial x_1}\frac{\partial f_{2,i}}{\partial x_2} - \frac{\partial f_{1,i}}{\partial x_2}\frac{\partial f_{2,i}}{\partial x_1}}$$

$$f_{2,i+1} = f_{2,i} + \left(x_{1,i+1} - x_{1,i}\right)\frac{\partial f_{2,i}}{\partial x_1} + \left(x_{2,i+1} - x_{2,i}\right)\frac{\partial f_{2,i}}{\partial x_2}$$

$$x_{2,i+1} = x_{2,i} - \frac{f_{2,i}\frac{\partial f_{1,i}}{\partial x_1} - f_{1,i}\frac{\partial f_{2,i}}{\partial x_1}}{\frac{\partial f_{1,i}}{\partial x_1}\frac{\partial f_{2,i}}{\partial x_2} - \frac{\partial f_{1,i}}{\partial x_2}\frac{\partial f_{2,i}}{\partial x_1}}$$



Autovalores



Base matemática

- In heterogeneous systems:

$$\mathbf{Ax} = \mathbf{b}$$

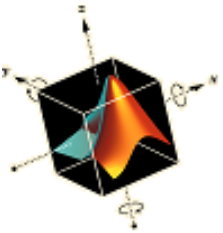
- Homogeneous systems:

$$\mathbf{Ax} = 0$$

- Trivial solution:

$$\mathbf{x} = 0$$

- Is there another way of formulating the system so that the solution would be meaningful?



Base matemática

- In a homogeneous system like:

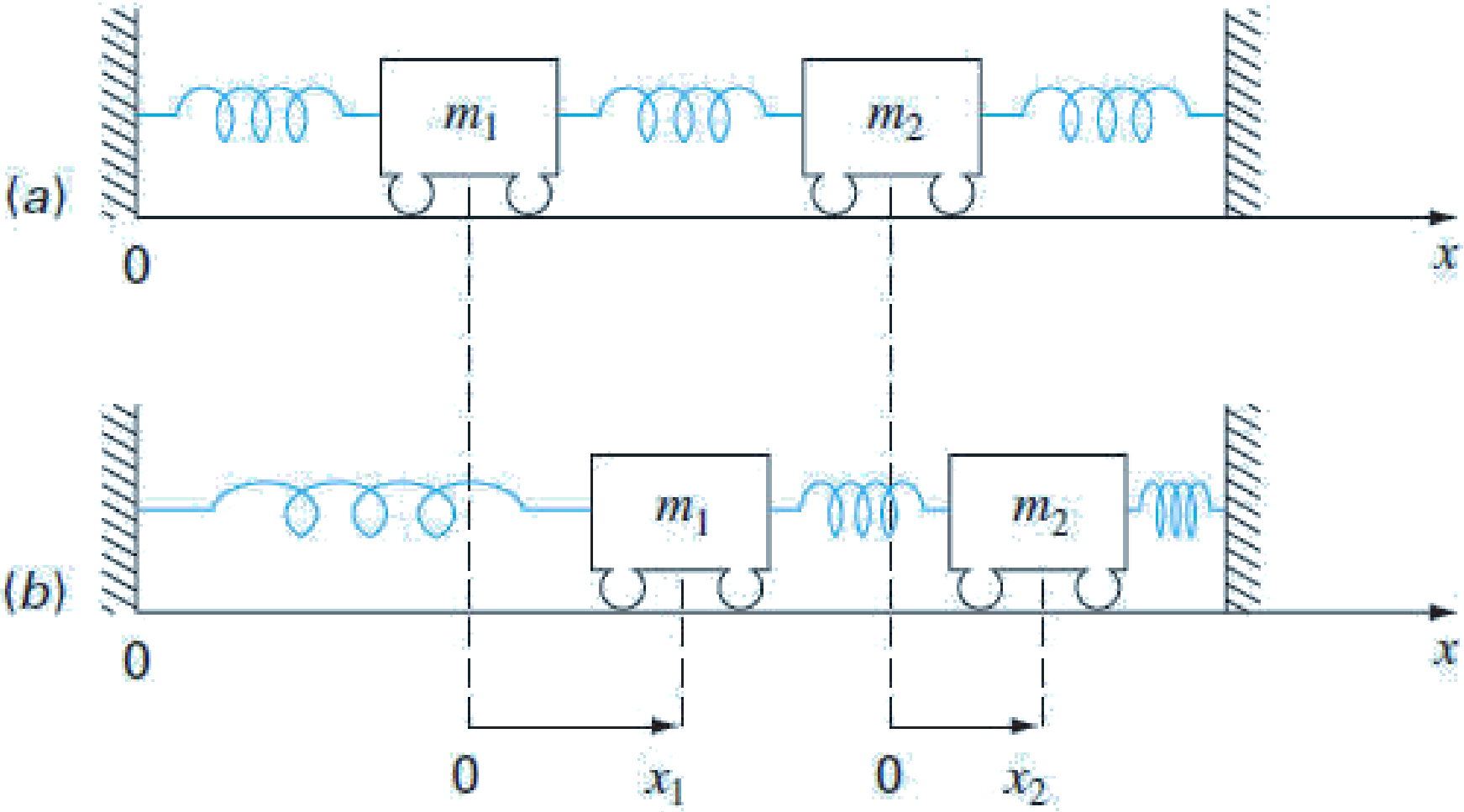
$$\begin{aligned}(a_{11}-\lambda)\,x_1 + a_{12}\,x_2 + a_{13}\,x_3 &= 0\\ a_{21}\,x_1 + (a_{22}-\lambda)\,x_2 + a_{23}\,x_3 &= 0\\ a_{31}\,x_1 + a_{32}\,x_2 + (a_{33}-\lambda)\,x_3 &= 0\end{aligned}$$

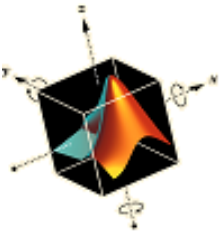
or in matrix form

$$(\mathbf{A}-\lambda\mathbf{I})\mathbf{x}=0$$

- For this case, there could be a value of λ that makes the equations equal zero. This is called an *eigenvalue*.

Oscilaciones o Vibraciones en sistemas masa - resorte





Vibraciones en sistemas masa – resorte: modelo con balances de fuerza

- From: $F = ma$

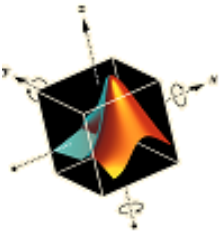
$$m_1 \frac{d^2x_1}{dt^2} = -k\,x_1 + k(x_2 - x_1)$$

$$m_2 \frac{d^2x_2}{dt^2} = -k(x_2 - x_1) - kx_2$$

- Collect terms:

$$m_1 \frac{d^2x_1}{dt^2} - k\,(-2x_1 + x_2) = 0$$

$$m_2 \frac{d^2x_2}{dt^2} - k\,(x_1 - 2x_2) = 0$$



Vibraciones en sistemas masa – resorte: modelo con balances de fuerza

- If $x_i = X_i \sin(\omega t)$ where $\omega = \dfrac{2\pi}{T_p}$
- Differentiate twice: $x_i'' = -X_i \omega^2 \sin(\omega t)$
- Substitute back into system and collect terms

$$\left(\frac{2k}{m_1} - \omega^2\right) X_1 - \frac{k}{m_1} X_2 = 0$$

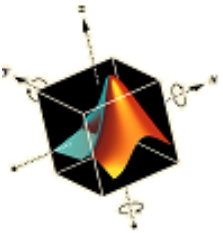
$$-\frac{k}{m_2} X_1 + \left(\frac{2k}{m_2} - \omega^2\right) X_2 = 0$$

Given $m_1 = m_2 = 40$ kg;
$k = 200$ N/m

$$\left(10 - \omega^2\right) X_1 - 5X_2 = 0$$

$$-5X_1 + \left(10 - \omega^2\right) X_2 = 0$$

- This is a homogeneous system where the eigenvalue represents the square of the fundamental frequency.



Vibraciones en sistemas masa – resorte: modelo con balances de fuerza

$$\begin{aligned}(10-\omega^2)X_1 - 5X_2 &= 0\\ -5X_1 + (10-\omega^2)X_2 &= 0\end{aligned} \qquad \begin{bmatrix}10-\omega^2 & -5\\ -5 & 10-\omega^2\end{bmatrix}\begin{bmatrix}X_1\\ X_2\end{bmatrix} = \begin{bmatrix}0\\ 0\end{bmatrix}$$

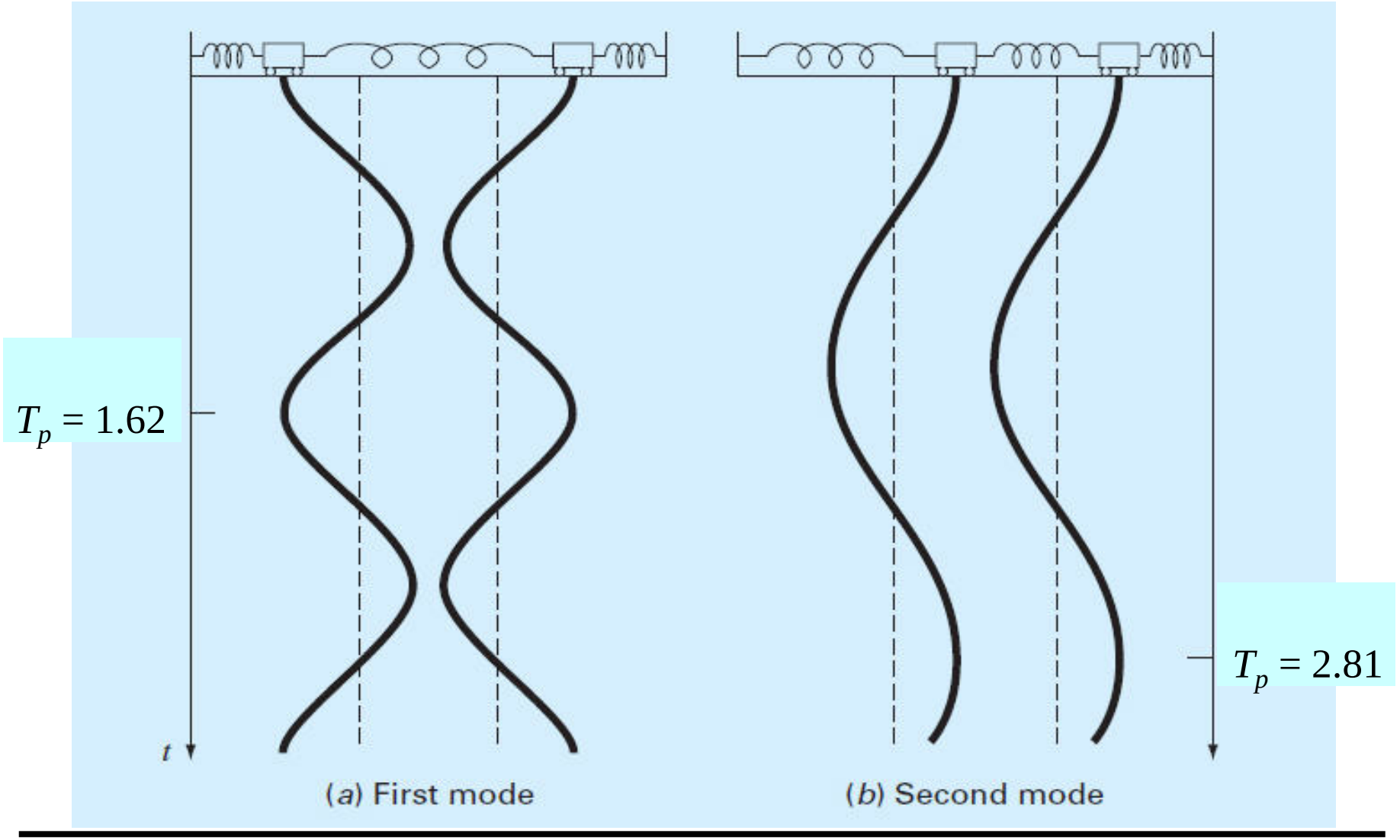
- Evaluate the determinant to yield a polynomial

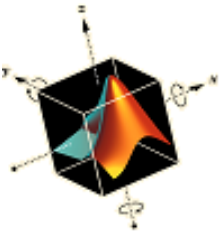
$$\begin{bmatrix}10-\omega^2 & -5\\ -5 & 10-\omega^2\end{bmatrix} = (\omega^2)^2 - 20\omega^2 + 75$$

- The two roots of this "characteristic polynomial" are the system's eigenvalues:

$$\omega^2 = \begin{matrix}15\\ 5\end{matrix} \quad \text{or} \quad \omega = \begin{matrix}3.873\ \text{Hz}\\ 2.236\ \text{Hz}\end{matrix} \quad T_p = \begin{matrix}2\pi/3.873 = 1.62\,\text{s}\\ 2\pi/2.236 = 2.81\text{s}\end{matrix}$$

Vibraciones en sistemas masa – resorte: modos principales de vibración





Método de potencia

- Iterative method to compute the largest eigenvalue (dominant) ***and*** its associated eigenvector.

$$(\mathbf{A}-\lambda\mathbf{I})\mathbf{x}=0 \qquad \mathbf{A}\mathbf{x}=\lambda\mathbf{x}$$

$$\mathbf{X}_i=\mathbf{A}\mathbf{X}_{i-1}, \quad i=1,2,3,\cdots$$

Vector $n\times 1$

Suppose that A has a dominant eigenvalue.

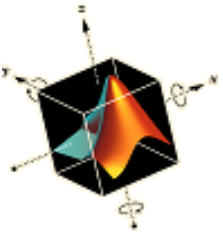
$$\mathbf{X}_1=\mathbf{A}\mathbf{X}_0$$

$$\mathbf{X}_2=\mathbf{A}\mathbf{X}_1=\mathbf{A}^2\mathbf{X}_0$$

$$\vdots$$

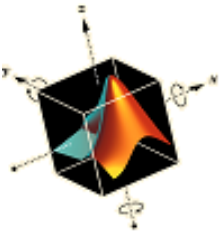
$$\mathbf{X}_m=\mathbf{A}\mathbf{X}_{m-1}=\mathbf{A}^m\mathbf{X}_0$$

$$\lambda_1\approx\frac{\mathbf{A}\mathbf{X}_m\cdot\mathbf{X}_m}{\mathbf{X}_m\cdot\mathbf{X}_m}$$ Rayleigh quotient



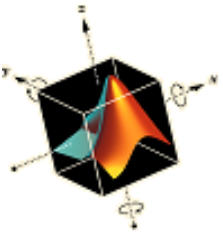
Cálculo de autovalores/autovectores en Matlab

```
>> A = [10 -5;-5 10]

A =
    10    -5
    -5    10

>> [v,lambda] = eig(A)

v =
   -0.7071   -0.7071
   -0.7071    0.7071
lambda =
     5     0
     0    15
```



Ecuaciones Diferenciales Ordinarias

Problemas de valor inicial



Ecuaciones diferenciales ordinarias

- Methods described here are for solving differential equations of the form:

$$\frac{dy}{dt} = f(t, y)$$

- The methods in this section are all *one-step* methods and have the general format:

$$y_{i+1} = y_i + \phi h$$

where ϕ is called an *increment function*, and is used to extrapolate from an old value y_i to a new value y_{i+1}.

Método de Euler

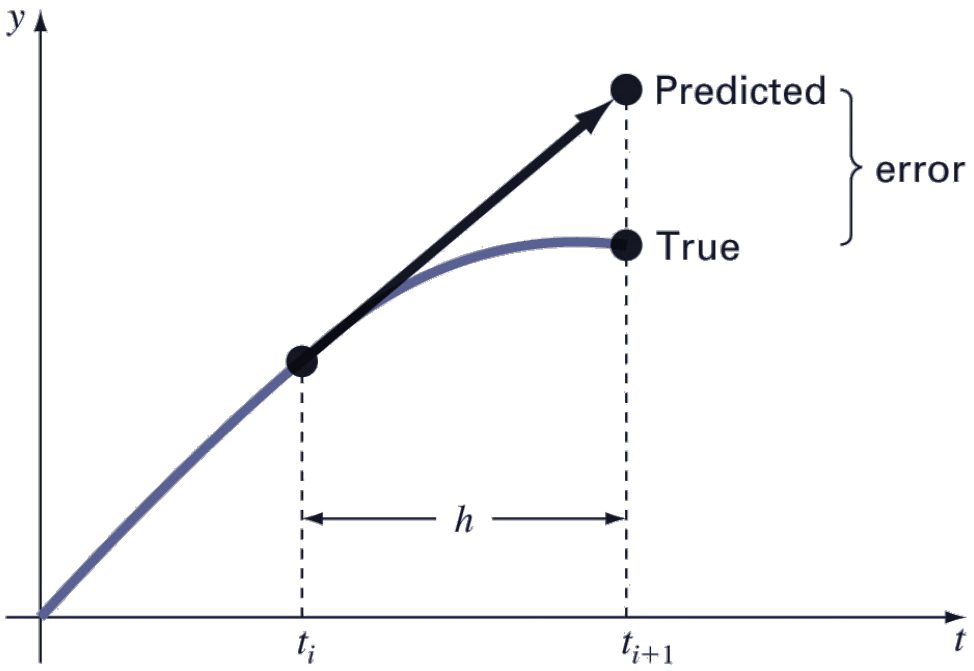
- The first derivative provides a direct estimate of the slope at t_i:

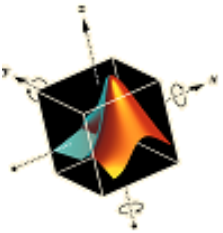
$$\left.\frac{dy}{dt}\right|_{t_i} = f(t_i, y_i)$$

and the Euler method uses that estimate as the increment function:

$$\phi = f(t_i, y_i)$$

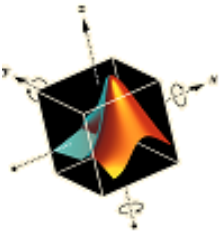
$$y_{i+1} = y_i + f(t_i, y_i)h$$





Análisis de error del método de Euler

- The numerical solution of ODEs involves two types of error:
  - *Truncation errors*, caused by the nature of the techniques employed
  - *Roundoff errors*, caused by the limited numbers of significant digits that can be retained
- The total, or *global* truncation error can be further split into:
  - *local truncation error* that results from an application method in question over a single step, and
  - *propagated truncation error* that results from the approximations produced during previous steps.



Análisis de error del método de Euler

- In general the Euler's method has large errors. This can be illustrated using Taylor series.

$$y_{i+1} = y_i + \left.\frac{dy}{dx}\right|_{x_i,y_i}(x_{i+1}-x_i) + \frac{1}{2!}\left.\frac{d^2y}{dx^2}\right|_{x_i,y_i}(x_{i+1}-x_i)^2 + \frac{1}{3!}\left.\frac{d^3y}{dx^3}\right|_{x_i,y_i}(x_{i+1}-x_i)^3 + \ldots$$

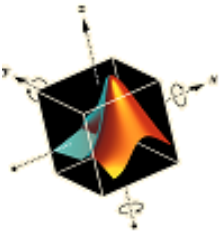
$$y_{i+1} = y_i + f(x_i,y_i)(x_{i+1}-x_i) + \frac{1}{2!}f'(x_i,y_i)(x_{i+1}-x_i)^2 + \frac{1}{3!}f''(x_i,y_i)(x_{i+1}-x_i)^3 + \ldots$$

- The first two terms of the Taylor series are the Euler's method

$$y_{i+1} = y_i + f(x_i,y_i)h$$

- The true error in the approximation is given by

$$E_t = \frac{f'(x_i,y_i)}{2!}h^2 + \frac{f''(x_i,y_i)}{3!}h^3 + \ldots \qquad E_t \propto h^2$$

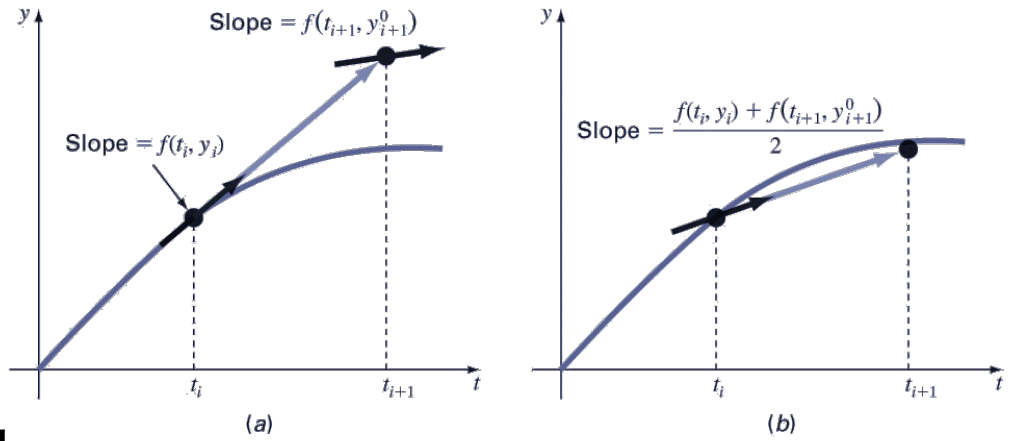


Análisis de error del método de Euler

- The local truncation error for Euler's method is $O(h^2)$ and proportional to the derivative of $f(t,y)$ while the global truncation error is $O(h)$.
- This means:
  - The global error can be reduced by decreasing the step size, and
  - Euler's method will provide error-free predictions if the underlying function is linear.
- Euler's method is *conditionally stable*, depending on the size of h.

Método de Heun

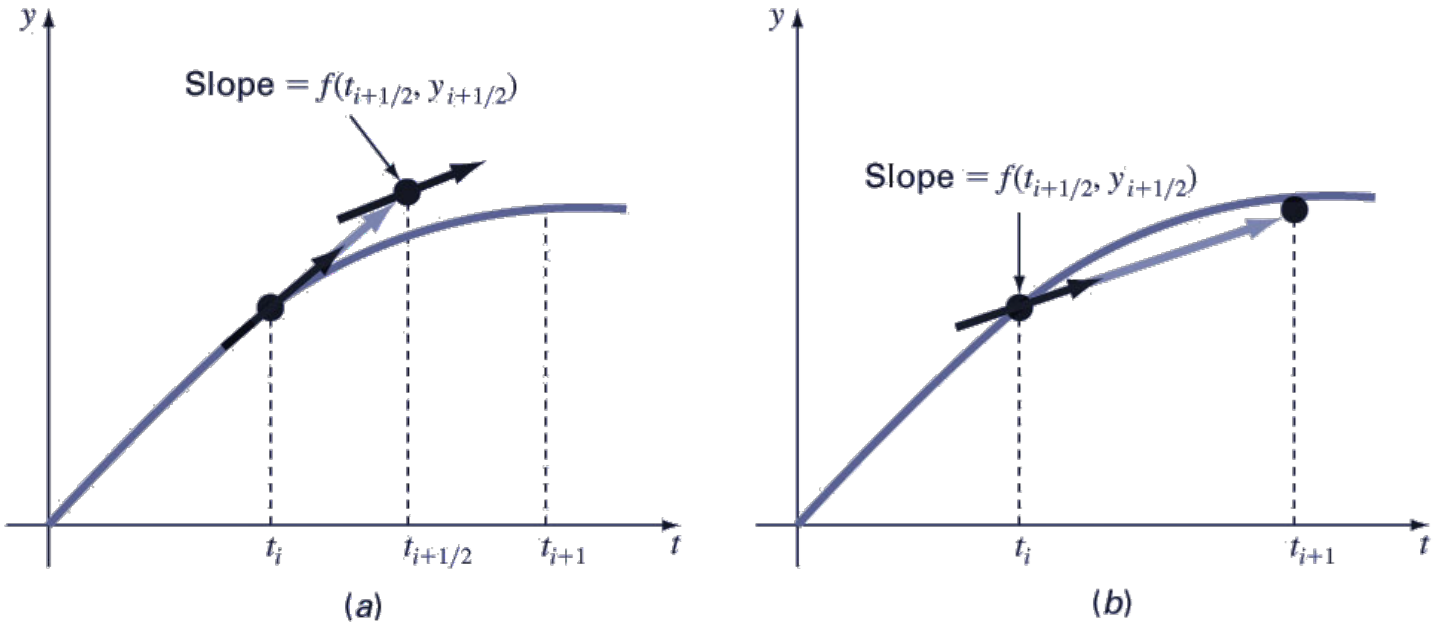
- One method to improve Euler's method is to determine derivatives at the beginning and predicted ending of the interval and average them:
- This process relies on making a prediction of the new value of y, then correcting it based on the slope calculated at that new value.
- This predictor-corrector approach can be iterated to convergence:



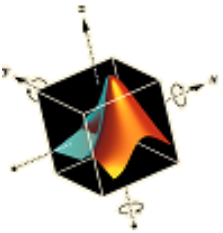
$$y^j_{i+1} \leftarrow y^m_i + \frac{f(t_i, y^m_i) + f(t_{i+1}, y^{j-1}_{i+1})}{2} h$$

Método del punto medio

- Another improvement to Euler's method is similar to Heun's method, but predicts the slope at the midpoint of an interval rather than at the end:



- This method has a local truncation error of $O(h^3)$ and global error of $O(h^2)$



Métodos Runge-Kutta

- Runge-Kutta (RK) methods achieve the accuracy of a Taylor series approach without requiring the calculation of higher derivatives.
- For RK methods, the increment function ϕ can be generally written as: $\phi = a_1 k_1 + a_2 k_2 + \cdots + a_n k_n$ where the *a*'s are constants and the *k*'s are

$$k_1 = f(t_i, y_i)$$
$$k_2 = f(t_i + p_1 h, y_i + q_{11} k_1 h)$$
$$k_3 = f(t_i + p_2 h, y_i + q_{21} k_1 h + q_{22} k_2 h)$$
$$\vdots$$
$$k_n = f(t_i + p_{n-1} h, y_i + q_{n-1,1} k_1 h + q_{n-1,2} k_2 h + \cdots + q_{n-1,n-1} k_{n-1} h)$$

where the *p*'s and *q*'s are constants.

Método clásico Runge-Kutta de cuarto orden

- The most popular RK methods are fourth-order, and the most commonly used form is:

$$y_{i+1} = y_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)h$$

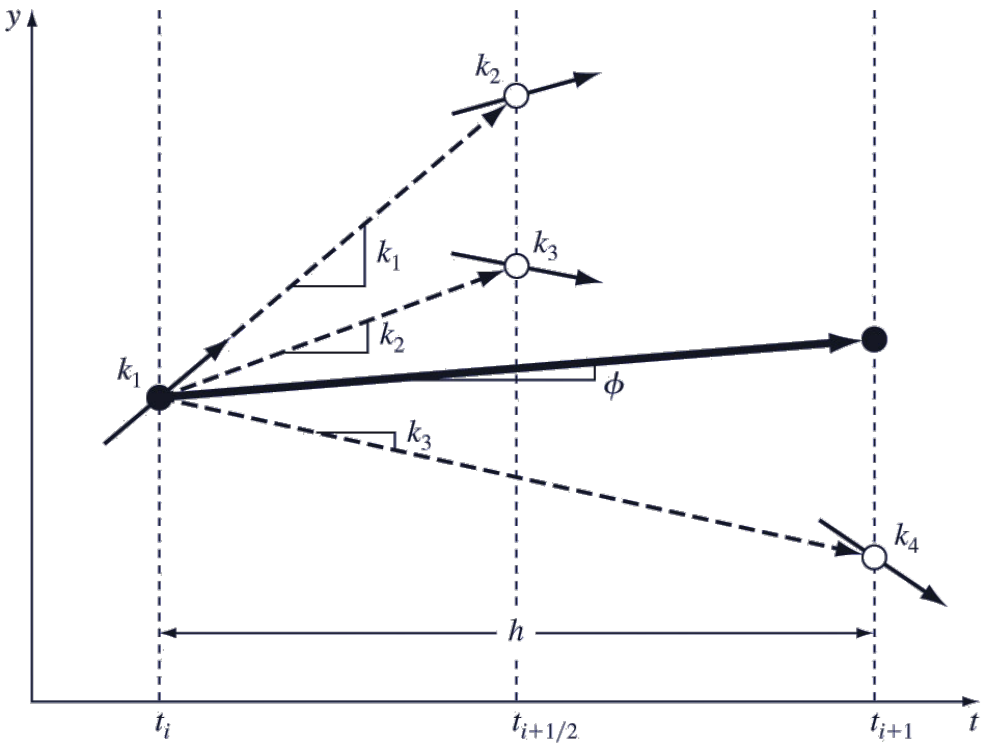
where:

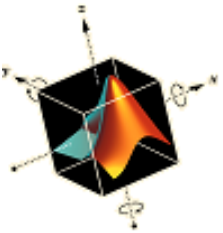
$$k_1 = f(t_i, y_i)$$

$$k_2 = f\left(t_i + \frac{1}{2}h, y_i + \frac{1}{2}k_1 h\right)$$

$$k_3 = f\left(t_i + \frac{1}{2}h, y_i + \frac{1}{2}k_2 h\right)$$

$$k_4 = f(t_i + h, y_i + k_3 h)$$



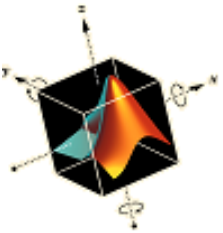


Sistemas de ecuaciones

- Many practical problems require the solution of a *system* of equations:

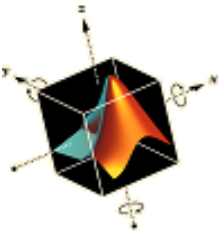
$$\frac{dy_1}{dt} = f_1(t, y_1, y_2, \cdots, y_n)$$
$$\frac{dy_2}{dt} = f_2(t, y_1, y_2, \cdots, y_n)$$
$$\vdots$$
$$\frac{dy_n}{dt} = f_n(t, y_1, y_2, \cdots, y_n)$$

- The solution of such a system requires that *n* initial conditions be known at the starting value of *t*.



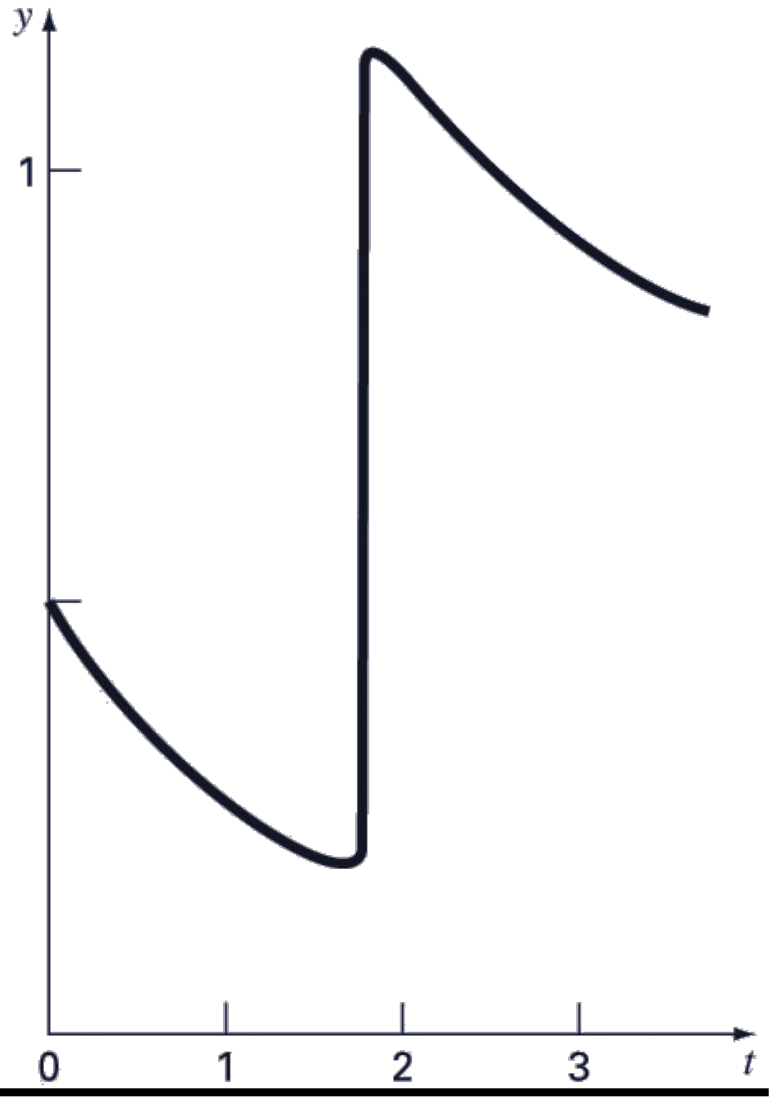
Sistemas de ecuaciones – métodos de solución

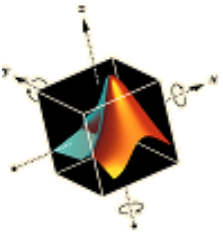
- Single-equation methods can be used to solve systems of ODE's as well; for example, Euler's method can be used on systems of equations - the one-step method is applied for every equation at each step before proceeding to the next step.
- Fourth-order Runge-Kutta methods can also be used, but care must be taken in calculating the *k*'s.



Métodos Runge-Kutta adaptativos

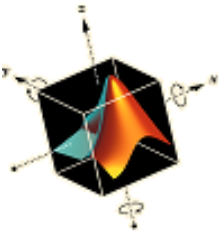
- The solutions to some ODE problems exhibit multiple time scales - for some parts of the solution the variable changes slowly, while for others there are abrupt changes.
- Constant step-size algorithms would have to apply a small step-size to the entire computation, wasting many more calculations on regions of gradual change.
- Adaptive algorithms, on the other hand, can change step-size depending on the region.





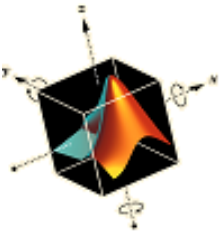
Propuestas de métodos adaptativos

- There are two primary approaches to incorporate adaptive step-size control:
  - *Step halving* - perform the one-step algorithm two different ways, once with a full step and once with two half-steps, and compare the results.
  - *Embedded RK methods* - perform two RK iterations of different orders and compare the results. This is the preferred method.



Funciones Matlab para EDO

- MATLAB's `ode23` function uses second- and third-order RK functions to solve the ODE and adjust step sizes.
- MATLAB's `ode45` function uses fourth- and fifth-order RK functions to solve the ODE and adjust step sizes. This is recommended as the first function to use to solve a problem.
- MATLAB's `ode113` function is a multistep solver useful for computationally intensive ODE functions.



Uso de funciones ode

- The functions are generally called in the same way; ode45 is used as an example:

  `[t, y] = ode45(odefun, tspan, y0)`

  - *y*: solution array, where each column represents one of the variables and each row corresponds to a time in the t vector
  - *odefun*: function returning a column vector of the right-hand-sides of the ODEs
  - *tspan*: time over which to solve the system
    - If `tspan` has two entries, the results are reported for those times as well as several intermediate times based on the steps taken by the algorithm
    - If `tspan` has more than two entries, the results are reported only for those specific times
  - *y0*: vector of initial values

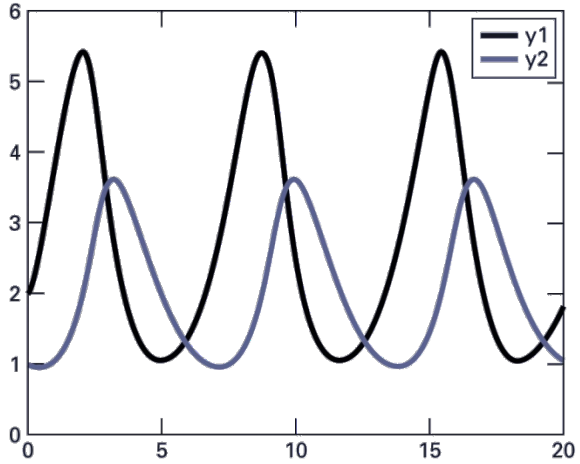
Ejemplo presa-depredador

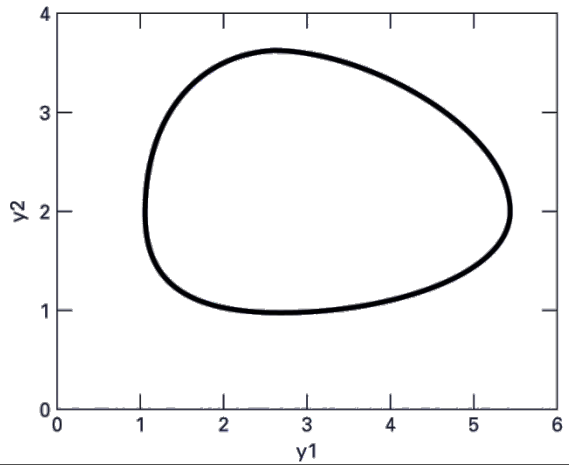
- Solve: $\frac{dy_1}{dt} = 1.2y_1 - 0.6y_1y_2 \quad \frac{dy_2}{dt} = 0.8y_2 + 0.3y_1y_2$

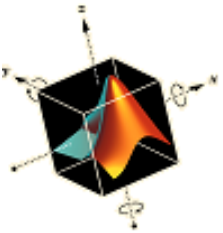
  with $y_1(0)=2$ and $y_2(0)=1$ for 20 seconds
- predprey.m M-file:

```
function yp = predprey(t, y)
yp = [1.2*y(1)-0.6*y(1)*y(2);…
      -0.8*y(2)+0.3*y(1)*y(2)];
```

-
```
tspan = [0 20];
y0 = [2, 1];
[t, y] = ode45(@predprey, tspan, y0);
figure(1); plot(t,y); figure(2); plot(y(:,1),y(:,2));
```







Opciones del solver ODE

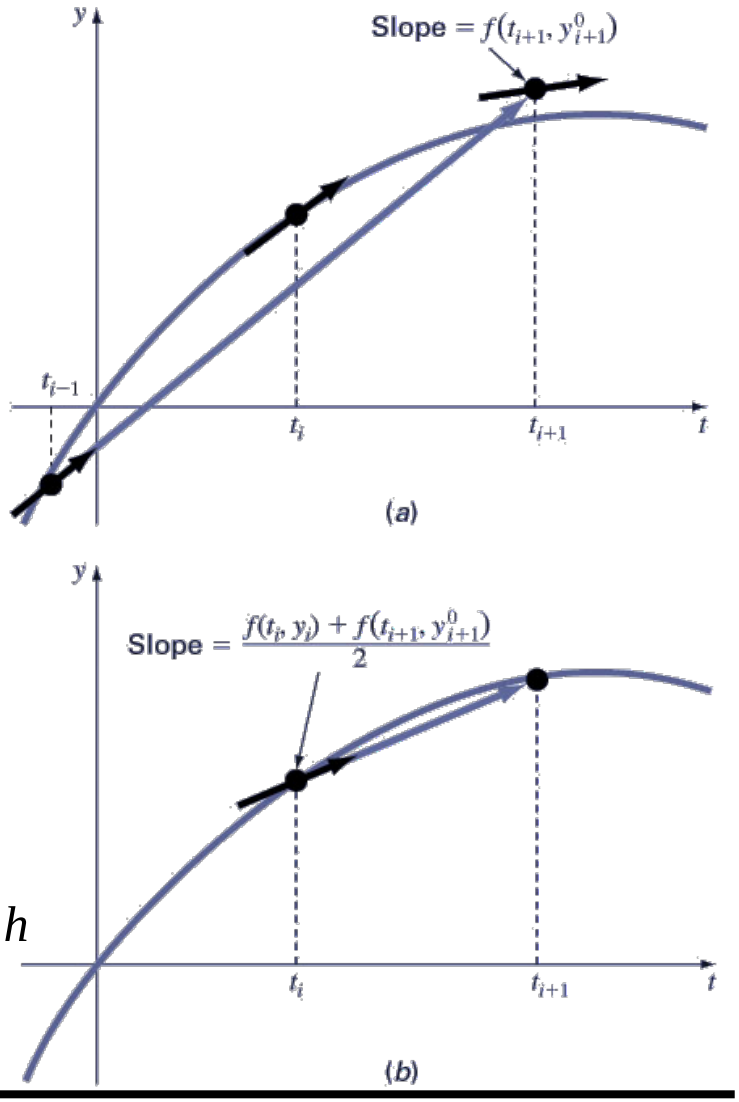
- Options to ODE solvers may be passed as an optional fourth argument, and are generally created using the `odeset` function:
  `options=odeset('par1', 'val1', 'par2', 'val2',…)`
- Commonly used parameters are:
  - `'RelTol'`: adjusts relative tolerance
  - `'AbsTol'`: adjusts absolute tolerance
  - `'InitialStep'`: sets initial step size
  - `'MaxStep'`: sets maximum step size (default: one tenth of `tspan` interval)

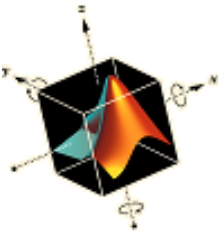
Métodos multipaso

- *Multistep methods* are based on the insight that, once the computation has begun, valuable information from the previous points exists.
- One example is the non-self-starting Heun's Method, which has the following predictor and corrector equations:

(a) Predictor $$y_{i+1}^0 = y_{i-1}^m + f\left(t_i, y_i^m\right)2h$$

(b) Corrector $$y_{i+1}^j = y_i^m + \frac{f\left(t_i, y_i^m\right) + f\left(t_{i+1}, y_{i+1}^{j-1}\right)}{2}h$$



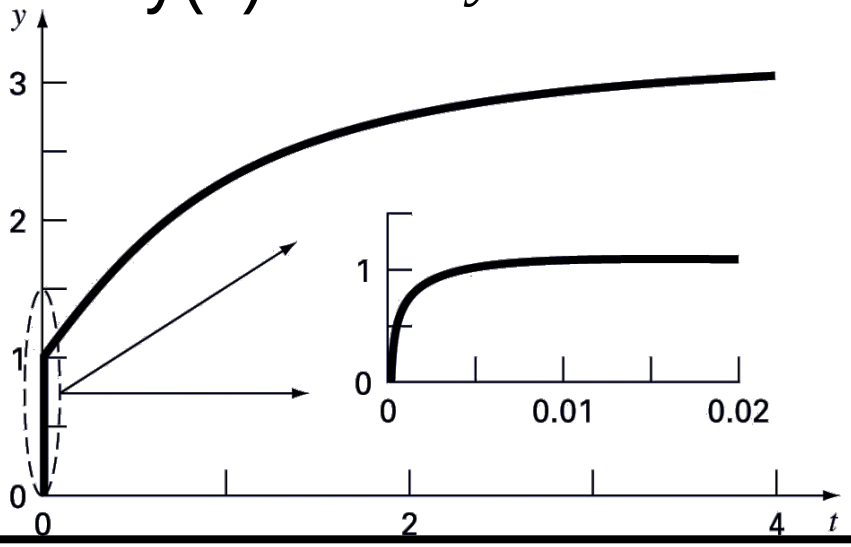


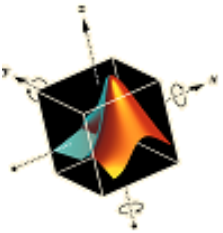
Rigidez (Stiffness)

- A *stiff system* is one involving rapidly changing components together with slowly changing ones.
- An example of a single stiff ODE is:

$$\frac{dy}{dt} = -1000y + 3000 - 2000e^{-t}$$

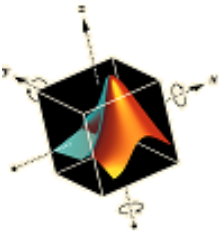
whose solution if y(0)=0 is: $y = 3 - 0.998e^{-1000t} - 2.002e^{-t}$





Funciones Matlab para sistemas Stiff

- MATLAB has a number of built-in functions for solving stiff systems of ODEs, including `ode15s`, `ode23`, `ode23t`, and `ode23tb`.
- The arguments for the stiff solvers are the same as those for previous solvers.



Ecuaciones Diferenciales Ordinarias

Problemas de valor de frontera

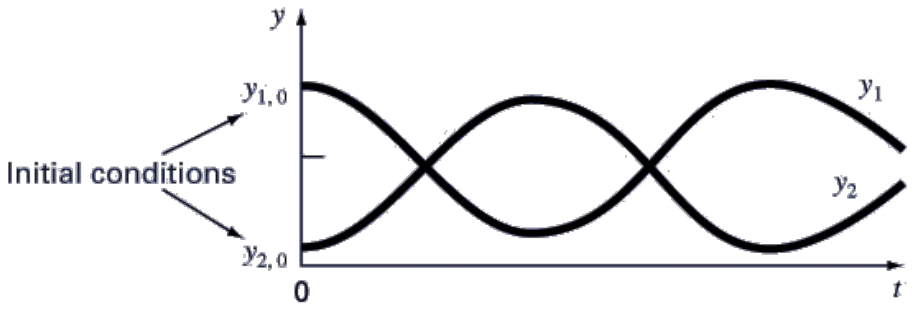
Problemas de valor frontera

- *Boundary-value problems* are those where conditions are not known at a single point but rather are given at different values of the independent variable.
- Boundary conditions may include values for the variable or values for derivatives of the variable.

$$\frac{dy_1}{dt} = f_1(t, y_1, y_2)$$

$$\frac{dy_2}{dt} = f_2(t, y_1, y_2)$$

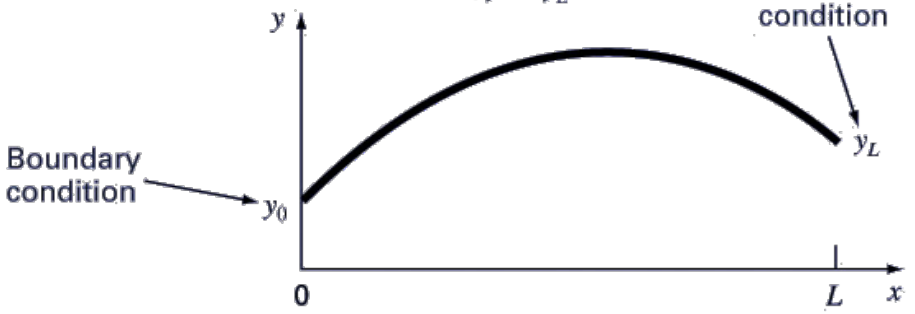
where at $t = 0, y_1 = y_{1,0}$ and $y_2 = y_{2,0}$



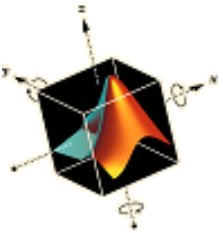
(a)

$$\frac{d^2y}{dx^2} = f(x, y)$$

where at $x = 0, y = y_0$
$x = L, y = y_L$



(b)

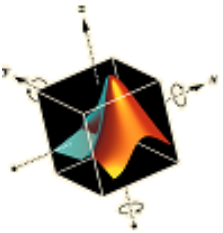


Sistemas de orden superior

- MATLAB's ODE solvers are based on solving first-order differential equations only.
- To solve an n^{th} order system (n>1), the system must be written as n first-order equations:

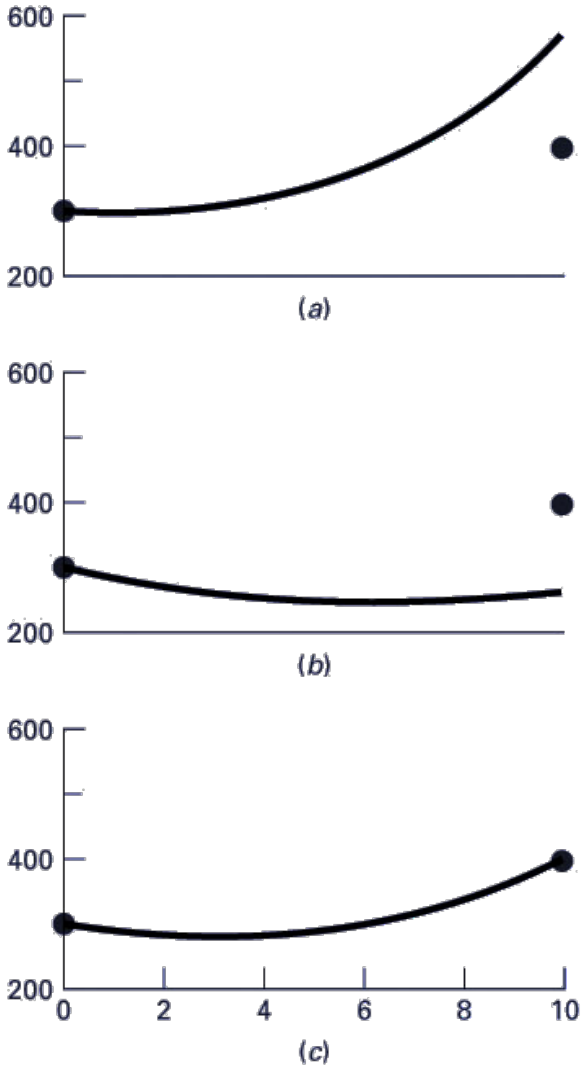
$$\frac{d^2T}{dx^2} + h'\left(T_\infty - T\right) = 0 \Rightarrow \begin{cases} \dfrac{dT}{dx} = z \\ \dfrac{dT}{dz} = -h'\left(T_\infty - T\right) \end{cases}$$

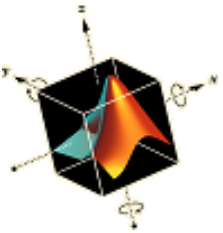
- Each first-order equation needs an initial value or boundary value to solve.



Método de disparo

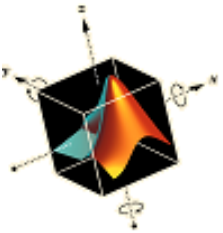
- One method for solving boundary-value problems - the shooting method - is based on converting the boundary-value problem into an equivalent initial-value problem.
- Generally, the equivalent system will not have sufficient initial conditions and so a guess is made for any undefined values.
- The guesses are changed until the final solution satisfies all the boundary conditions.
- For linear ODEs, only two “shots” are required - the proper initial condition can be obtained as a linear interpolation of the two guesses.





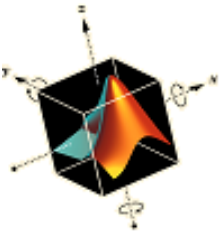
Condiciones de contorno

- *Dirichlet boundary conditions* are those where a fixed value of a variable is known at a particular location.
- *Neumann boundary conditions* are those where a derivative is known at a particular location.
- Shooting methods can be used for either kind of boundary condition.



Método de disparo para EDOs no lineales

- For nonlinear ODEs, interpolation between two guesses will not necessarily result in an accurate estimate of the required boundary condition.
- Instead, the boundary condition can be used to write a roots problem with the estimate as a variable.



Método de disparo - ejemplo

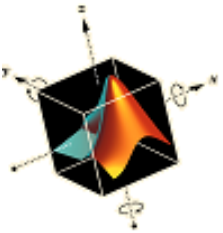
- Solve

$$\frac{d^2T}{dx^2} + h'\left(T_\infty - T\right) + \sigma'\left(T_\infty^4 - T^4\right) = 0$$

with σ'=2.7x10^{-9} K^{-3} m^{-2}, L=10 m, h'=0.05 m-2, T_∞=200 K, $T(0)$ = 300 K, and $T(10)$ = 400 K.

- First - break into two equations:

$$\frac{d^2T}{dx^2} + h'\left(T_\infty - T\right) + \sigma'\left(T_\infty^4 - T^4\right) = 0 \Rightarrow \begin{cases} \dfrac{dT}{dx} = z \\ \dfrac{dT}{dz} = -0.05\left(200 - T\right) - 2.7\times10^{-9}\left(1.6\times10^{9} - T\right) \end{cases}$$



Método de disparo – ejemplo código

- Code for derivatives:

```
function dy=dydxn(x,y)
dy=[y(2);…
     -0.05*(200-y(1))-2.7e-9*(1.6e9-y(1)^4)];
```

- Code for residual:

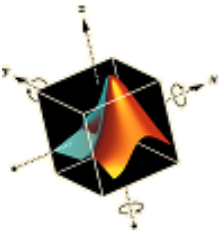
```
function r=res(za)
[x,y]=ode45(@dydxn, [0 10], [300 za]);
r=y(length(x),1)-400;
```

- Code for finding root of residual:

```
fzero(@res, -50)
```

- Code for solving system:

```
[x,y]=ode45(@dydxn,[0 10],[300 fzero(@res,-50)]);
```



Métodos de diferencias finitas

- The most common alternatives to the shooting method are finite-difference approaches.
- In these techniques, finite differences are substituted for the derivatives in the original equation, transforming a linear differential equation into a set of simultaneous algebraic equations.

Diferencias finitas - ejemplo

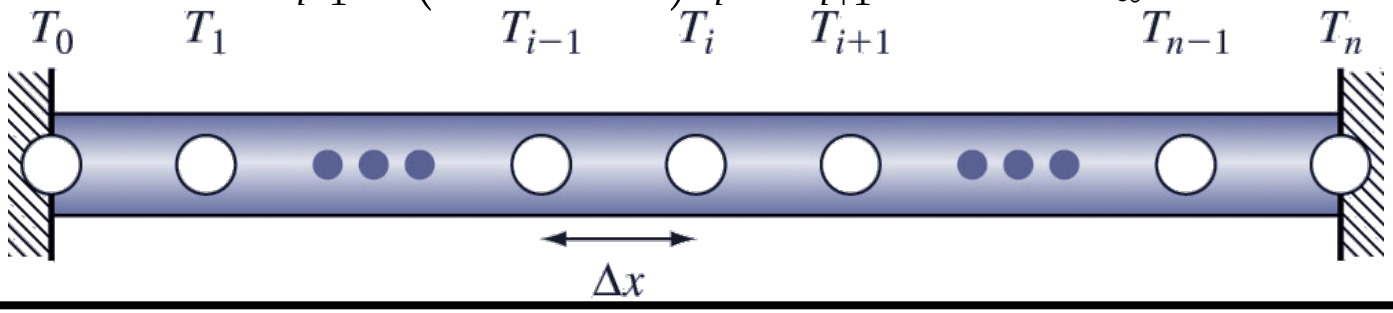
- Convert: $$\frac{d^2T}{dx^2} + h'\left(T_\infty - T\right) = 0$$

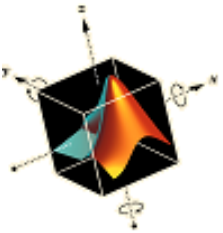
  into *n*-1 simultaneous equations at each interior point using centered difference equations:

$$\frac{d^2T}{dx^2} = \frac{T_{i-1} - 2T_i + T_{i+1}}{\Delta x^2}$$

$$\frac{T_{i-1} - 2T_i + T_{i+1}}{\Delta x^2} + h'\left(T_\infty - T_i\right) = 0$$

$$-T_{i-1} + \left(2 + h'\Delta x^2\right)T_i - T_{i+1} = h'\Delta x^2 T_\infty$$

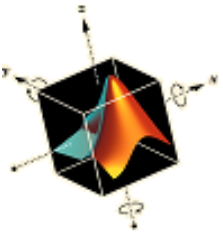




Diferencias finitas - ejemplo

- Since T_0 and T_n are known, they will be on the right-hand-side of the linear algebra system (in this case, in the first and last entries, respectively):

$$\begin{bmatrix} 2+h'\Delta x^2 & -1 & & & \\ -1 & 2+h'\Delta x^2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2+h'\Delta x^2 & \end{bmatrix} \begin{Bmatrix} T_1 \\ T_2 \\ \vdots \\ T_{n-1} \end{Bmatrix} = \begin{Bmatrix} h'\Delta x^2 T_\infty + T_0 \\ h'\Delta x^2 T_\infty \\ \vdots \\ h'\Delta x^2 T_\infty + T_n \end{Bmatrix}$$



Condiciones de contorno - derivadas

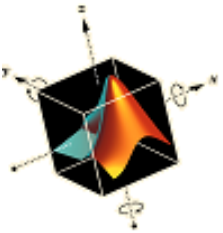
- Neumann boundary conditions are resolved by solving the centered difference equation at the point and rewriting the system equation accordingly.
- For example, if there is a Neumann condition at the T_0 point,

$$\left.\frac{dT}{dx}\right|_0 = \frac{T_1 - T_{-1}}{2\Delta x} \Rightarrow T_{-1} = T_1 - 2\Delta x\left(\left.\frac{dT}{dx}\right|_0\right)$$

$$-T_{-1} + \left(2 + h'\Delta x^2\right)T_0 - T_1 = h'\Delta x^2 T_\infty$$

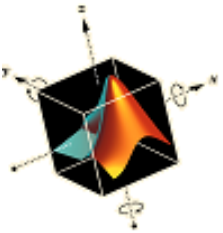
$$-\left[T_1 - 2\Delta x\left(\left.\frac{dT}{dx}\right|_0\right)\right] + \left(2 + h'\Delta x^2\right)T_0 - T_1 = h'\Delta x^2 T_\infty$$

$$\left(2 + h'\Delta x^2\right)T_0 - 2T_1 = h'\Delta x^2 T_\infty - 2\Delta x\left(\left.\frac{dT}{dx}\right|_0\right)$$



Método de diferencias finitas para EDOs no lineales

- Root location methods for systems of equations may be used to solve nonlinear ODEs.
- Another method is to adapt a successive substitution algorithm to calculate the values of the interior points.



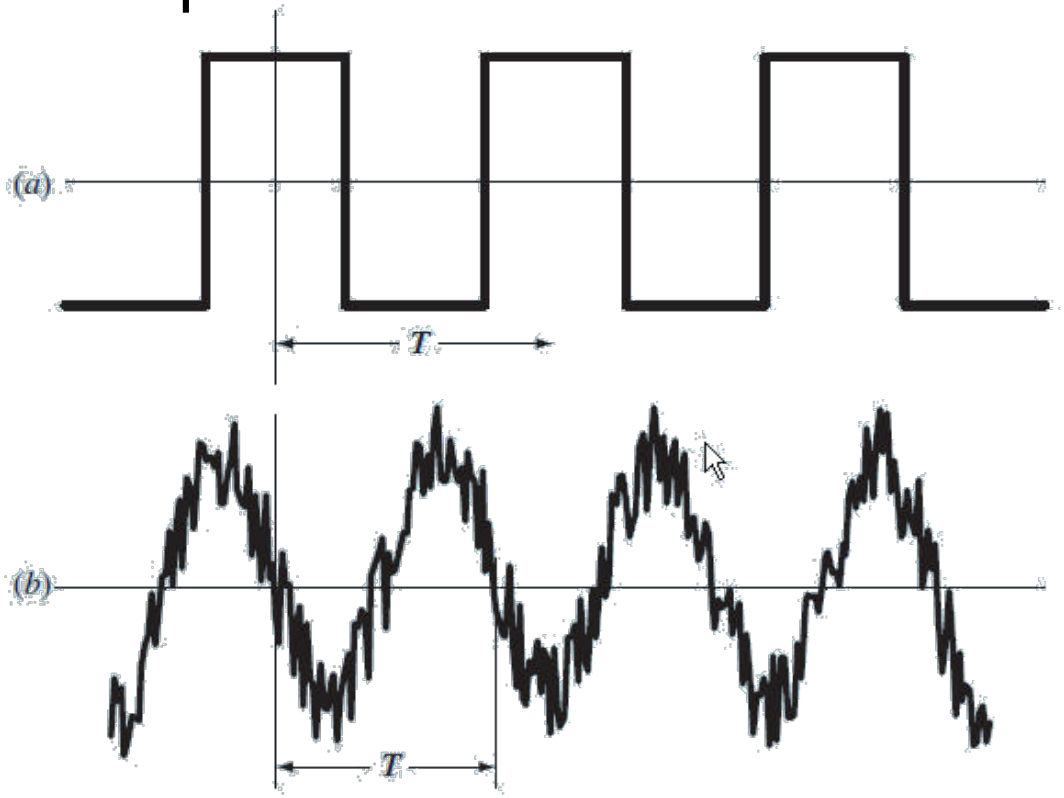
Análisis de Fourier

Funciones periódicas

- A periodic function is one for which

$$f(t) = f(t + T)$$

where T = the period



Sinusoides

$$f(t) = A_0 + C_1\cos(\omega_0 t + \theta)$$

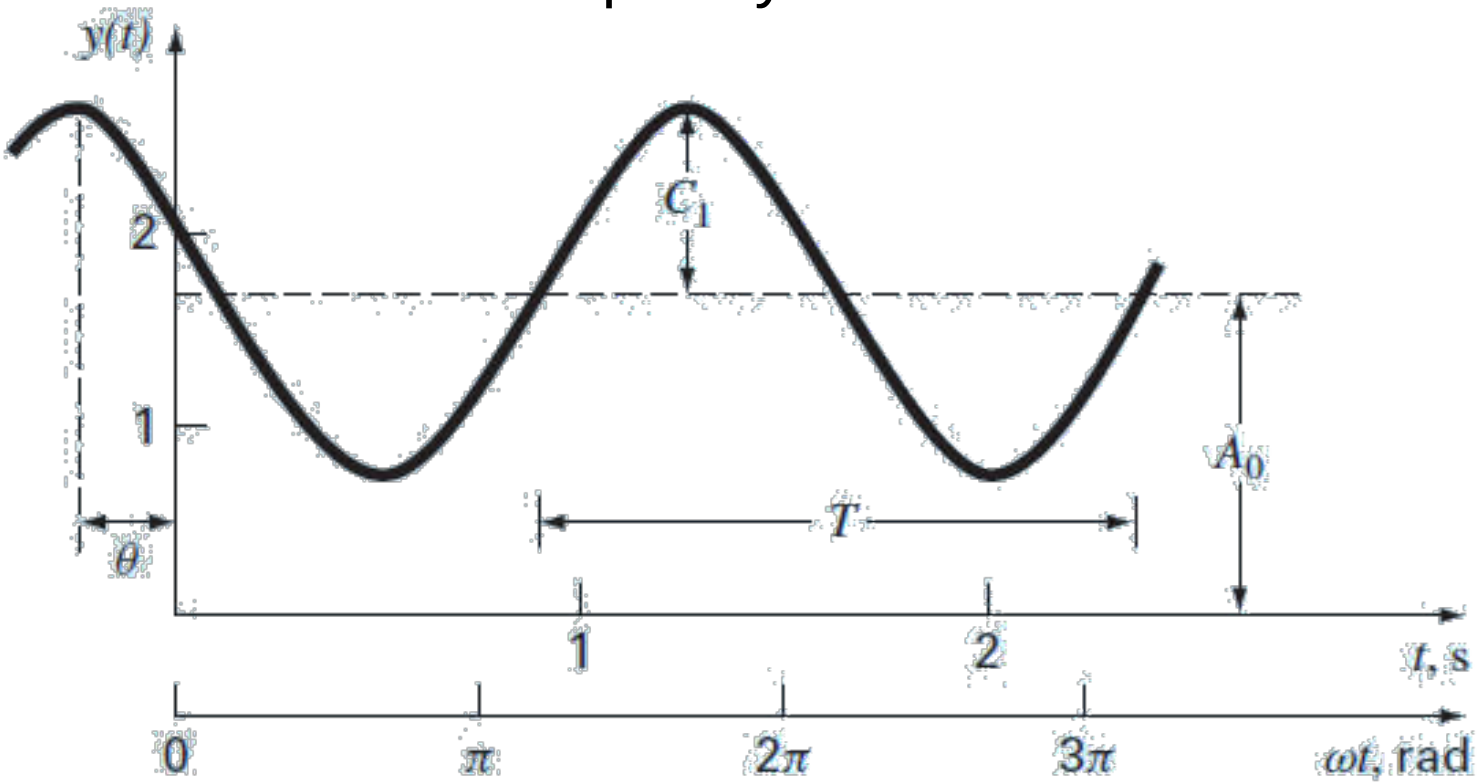
Mean value

Amplitude

Angular frequency

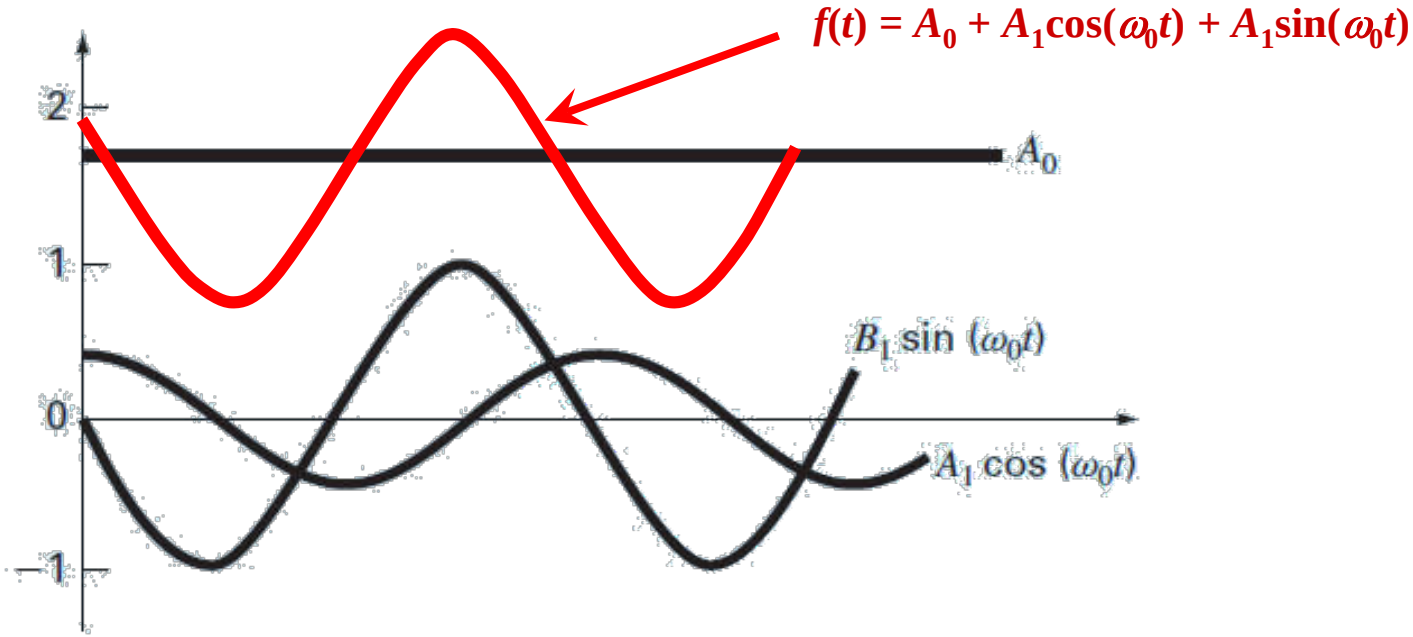
Phase shift

$$\omega_0 = 2\pi f = \frac{2\pi}{T}$$



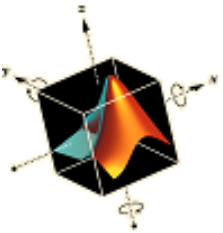
Representación alternativa

$$f(t) = A_0 + A_1\cos(\omega_0 t) + B_1\sin(\omega_0 t)$$



- The two forms are related by

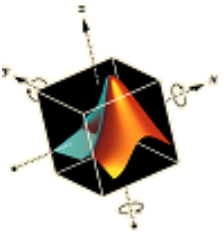
$$C_1 = \sqrt{A_1^2 + B_1^2} \qquad \theta = \arctan(-B_1/A_1)$$



Ajuste de curvas mediante sinusoides

- You will frequently have occasions to estimate intermediate values between precise data points.
- The function you use to interpolate must pass through the actual data points - this makes interpolation more restrictive than fitting.
- The most common method for this purpose is polynomial interpolation, where an (n-1)th order polynomial is solved that passes through n data points: $f(x) = a_1 + a_2 x + a_3 x^2 + \cdots + a_n x^{n-1}$

$$\text{MATLAB version}: \quad f(x) = p_1 x^{n-1} + p_2 x^{n-2} + \cdots + p_{n-1} x + p_n$$



Serie de Fourier continua

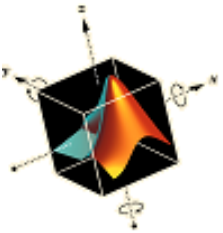
- A Fourier series of a periodic function f (with period 2π) defined as an expansion of f in a series of sines and cosines such as

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} (a_n \cos nx + b_n \sin nx)$$

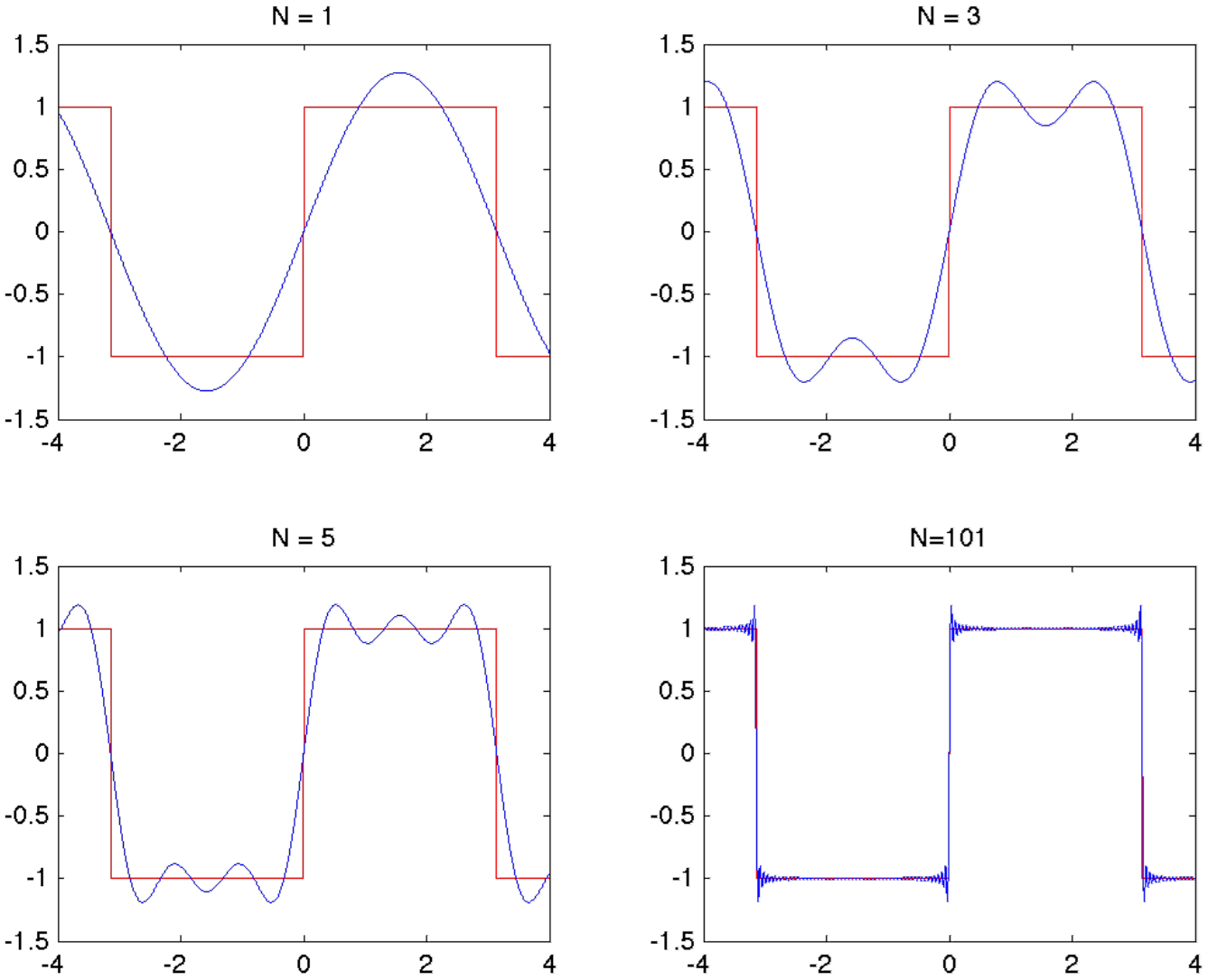
- Computing the coefficients a and b:

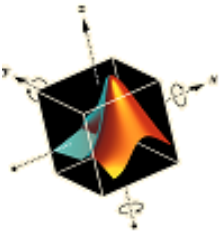
$$a_n = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \cos(nx) dx$$

$$b_n = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \sin(nx) dx \qquad a_0 = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) dx$$

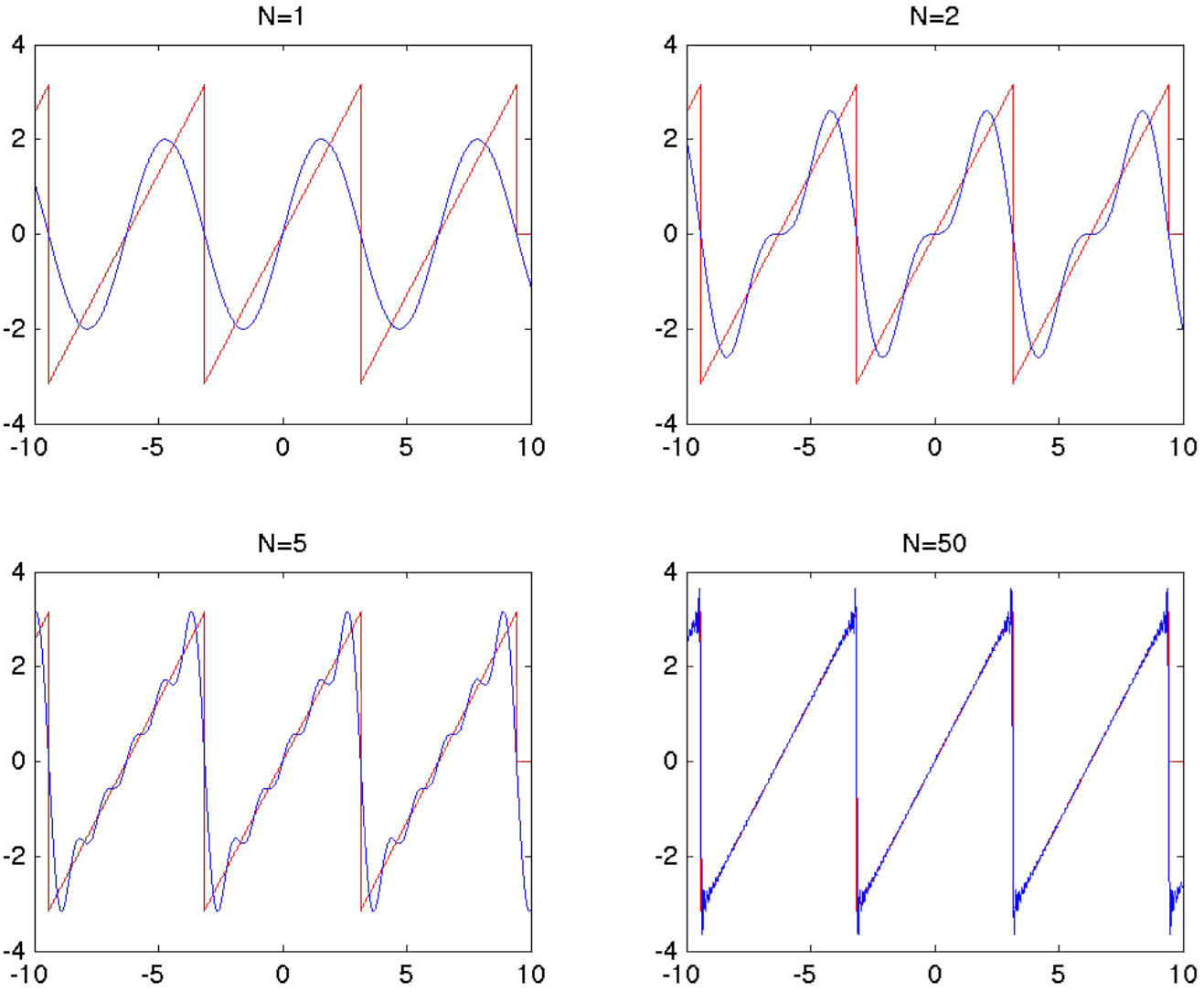


Serie de Fourier continua - ejemplo

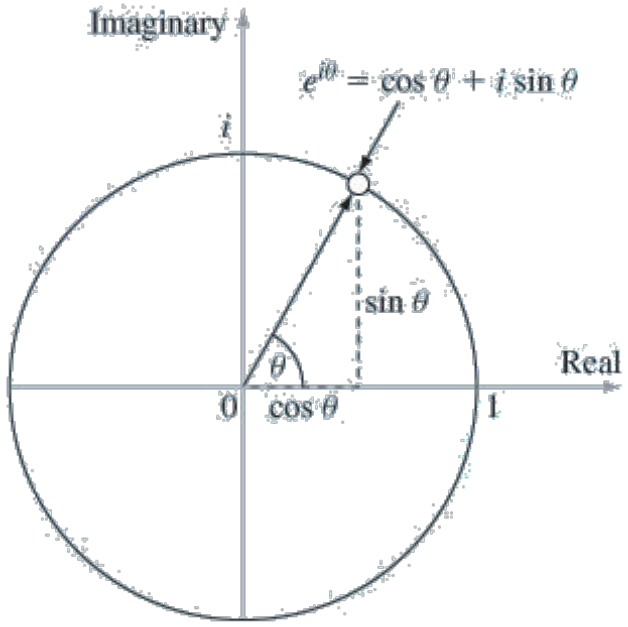




Serie de Fourier continua - ejemplo



Serie de Fourier – forma de Euler



$$e^{i\theta} = \cos\theta + i\sin\theta \qquad \text{(a unit vector)}$$

$$g(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty}(a_n \cos nx + b_n \sin nx)$$

If we express *cos nx* and *sin nx* in exponential form,

$$\cos nx = \frac{1}{2}\left(e^{inx} + e^{-inx}\right)$$

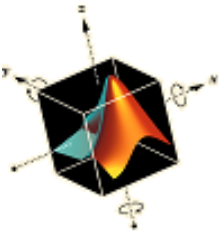
$$\sin nx = \frac{1}{2i}\left(e^{inx} - e^{-inx}\right)$$

we may rewrite this equation as

$$g(x) = \sum_{n=-\infty}^{\infty} G(n)e^{inx}$$

$$G(n) = \frac{1}{2}(a_n - ib_n) \qquad G(0) = \frac{1}{2}a_0.$$

$$G(-n) = \frac{1}{2}(a_n + ib_n) \qquad n > 0$$



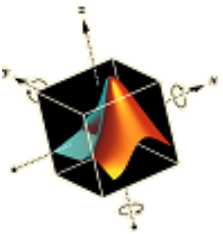
Serie de Fourier

- For a function g with period T:

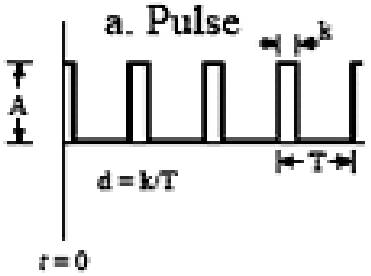
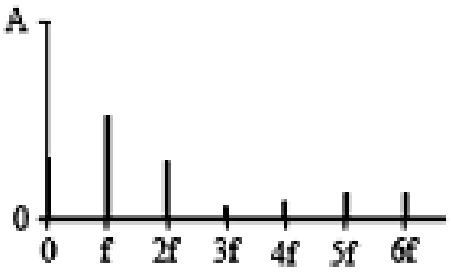
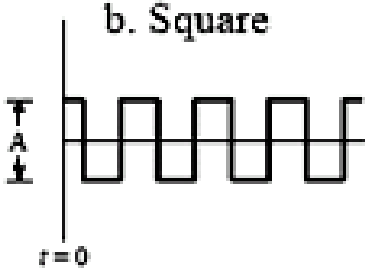
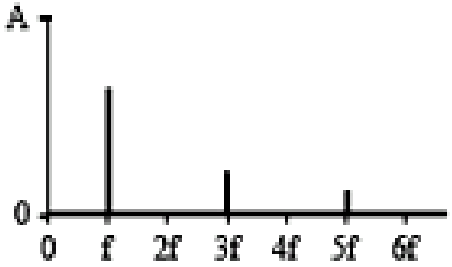
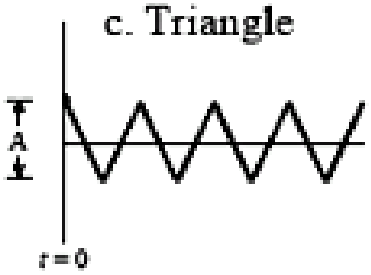
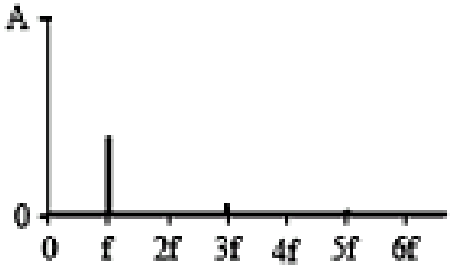
$$g(x) = \sum_{n=-\infty}^{\infty} G(n) e^{i 2\pi \frac{n}{T} x} = \sum_{n=-\infty}^{\infty} G(n) e^{i 2\pi n f_0 x}$$

- where $f_0 = 1/T$ is the fundamental frequency for the function g.
- In this formula, G(n) can be written as:

$$G(n) = \frac{1}{T} \int_0^T g(x) e^{-i 2\pi n f_0 x} dx$$

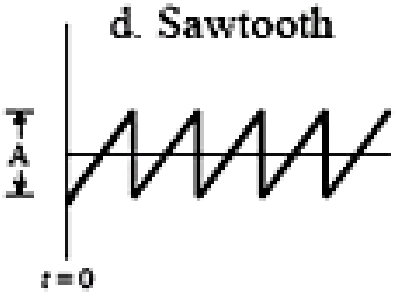


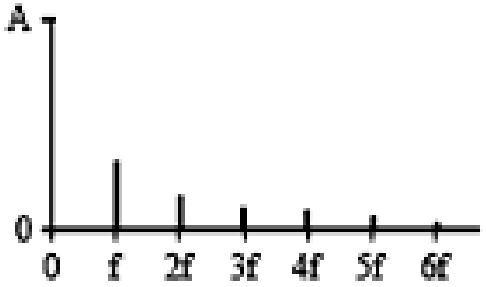
Serie de Fourier

Time Domain	Frequency Domain	
a. Pulse 		$a_0 = A\,d$ $a_n = \frac{2A}{n\pi}\sin(n\pi d)$ $b_n = 0$ (d = 0.27 in this example)
b. Square 		$a_0 = 0$ $a_n = \frac{2A}{n\pi}\sin\left(\frac{n\pi}{2}\right)$ $b_n = 0$ (all even harmonics are zero)
c. Triangle 		$a_0 = 0$ $a_n = \frac{4A}{(n\pi)^2}$ $b_n = 0$ (all even harmonics are zero)

Serie de Fourier

d. Sawtooth



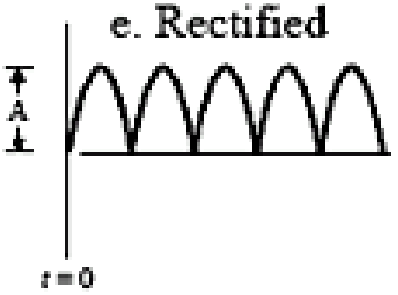


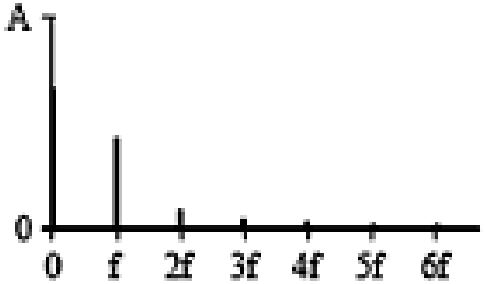
$$a_0 = 0$$

$$a_n = 0$$

$$b_n = \frac{A}{n\pi}$$

e. Rectified



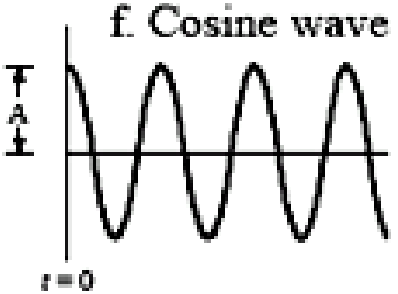


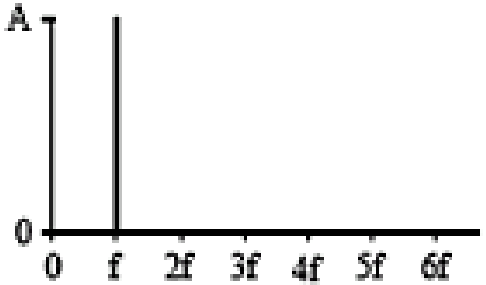
$$a_0 = 2A/\pi$$

$$a_n = \frac{-4A}{\pi(4n^2 - 1)}$$

$$b_n = 0$$

f. Cosine wave

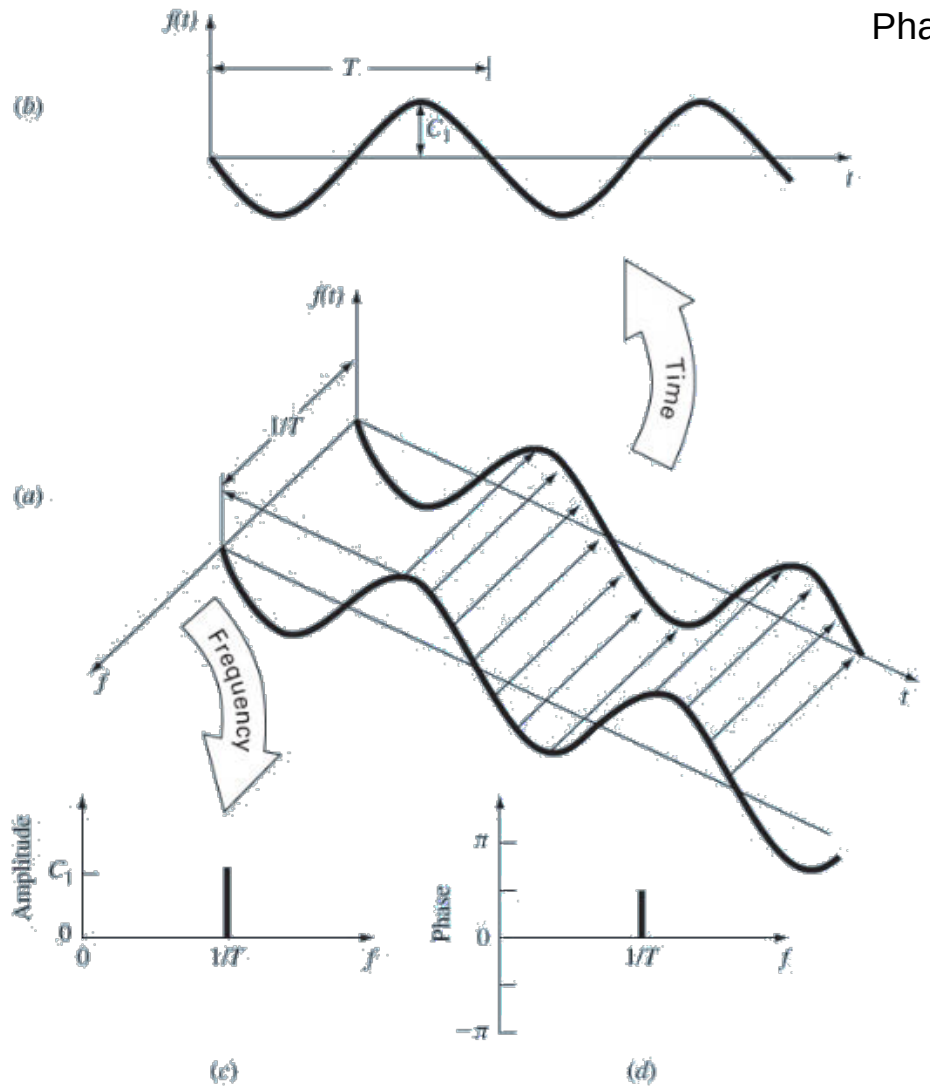




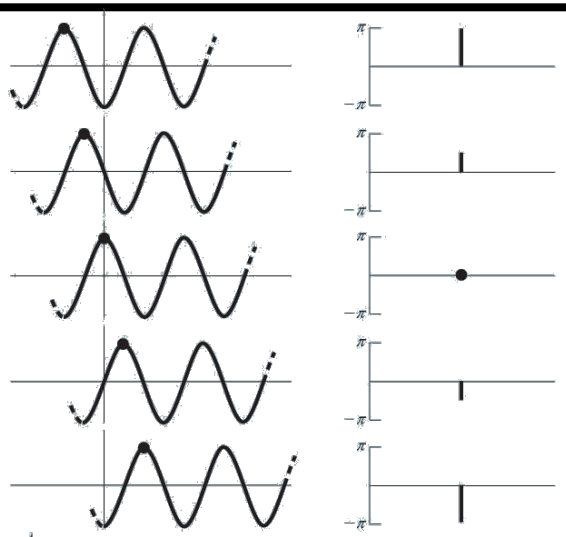
$$a_1 = A$$

(all other coefficients are zero)

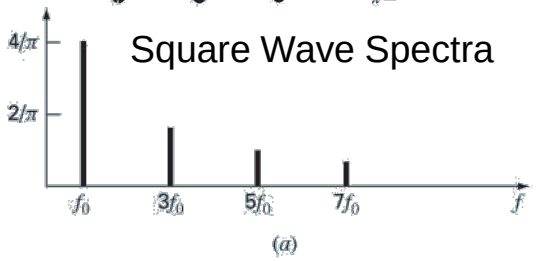
Dominios tiempo - frecuencia

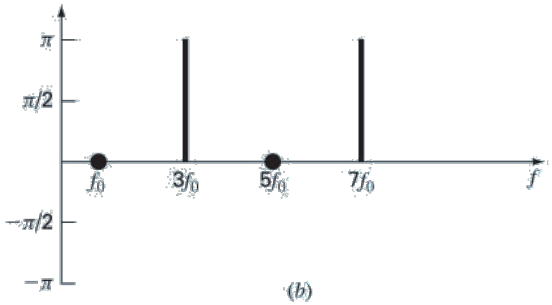


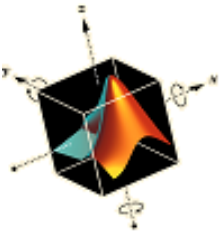
Phase Spectra



Square Wave Spectra







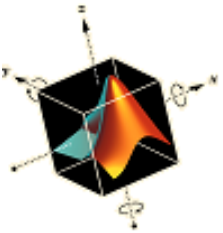
Transformada de Fourier

- For a periodic function f with period T, the Fourier coefficients F(n) are computed at multiples nf_0 of a fundamental frequency $f_0 = 1/T$
- For a non periodic function g(t), the Fourier coefficients become a continuous function of the frequencies f:

$$G(f) = \int_{-\infty}^{+\infty} g(t) e^{i 2\pi f t} dt$$ Fourier transform

- g(t) can then be reconstructed according to:

$$g(t) = \int_{-\infty}^{+\infty} G(f) e^{-i 2\pi f t} df$$ inverse Fourier transform



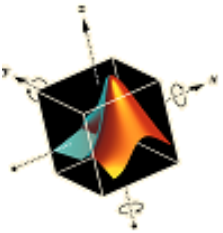
Transformada de Fourier discreta

- Given a discrete set of values x(n), with n integer; the discrete Time Fourier transform of x is:

$$X(f) = \sum_{n=-\infty}^{n=+\infty} x(n) e^{i 2\pi f n}$$

- Notice that X(f) is periodic:

$$X(f+k) = \sum_{n=-\infty}^{n=+\infty} x(n) e^{i 2\pi (f+k) n} = \sum_{n=-\infty}^{n=+\infty} x(n) e^{i 2\pi f n} e^{i 2\pi n} = X(f)$$



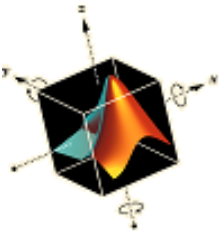
Transformada de Fourier discreta

- The sequence of numbers $x_0,\ldots x_{N-1}$ is transformed into a new series of numbers $X_0,\ldots.X_{N-1}$ according to the digital Fourier transform (DFT) formula:

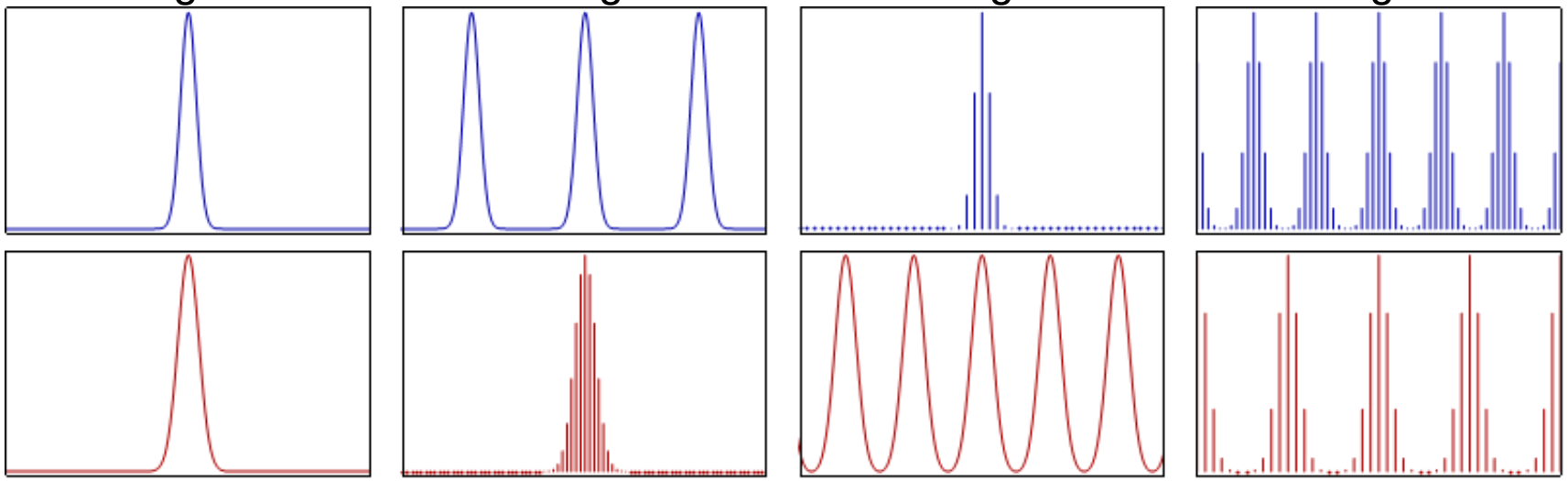
$$X(k) = \sum_{n=0}^{N-1} x(n) e^{i 2\pi \frac{kn}{N}}$$

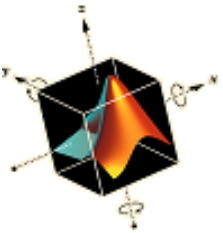
- The inverse DFT is given by: $x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) e^{-i 2\pi \frac{kn}{N}}$

- If n is a power of 2, X(k) can be computed using the Fast Fourier Transform (FFT). The command in Matlab is: X = fft(x) x(n) can be real or complex. X(k) is always complex.



Análisis de Fourier





Referencias

- Holistic Numerical Methods, Autar Kaw
- Applied Numerical Methods with Matlab for Engineers and Scientists, Chapra