Notas de Clase: Modelación Computacional

Juan Gómez
jgomezc1@eafit.edu.co
Nicolás Guarín-Zapata
nguarinz@eafit.edu.co
Edward Villegas-Pulgarin
evillega@eafit.edu.co

Departamento de Ingeniería Civil
Escuela de Ingeniería
Universidad EAFIT
Medellín, Colombia
2019

Tabla de contenidos

Capítulo 1

Métodos Numéricos

Presentación

Una amplia gama de problemas de ingeniería se reduce a la solución de un sistema de ecuaciones diferenciales que gobiernan el comportamiento, a lo largo de una region del espacio de alguna función de interés. Para que exista alguna posibilidad de solución la región de interés debe estar además limitada por una superficie (o frontera) sobre la cual se deben conocer además algunas caracteristicas de la solución. En matemáticas este tipo de formulaciones se conocen como problemas de contorno o de valores en la frontera (PVF), y si además involucran variaciones en el tiempo entonces se denominan problemas de valores iniciales y en la frontera (PVI-F).

Sin embargo en la mayoría de casos reales la solución de los problemas de valores en la frontera es imposible de conseguir por medio de métodos análiticos que produzcan soluciones cerradas, es decir funciones. Como alternativa de solución apearece la posibilidad de recurrir a planteamientos matemáticos que permiten re-escribir los PVF en términos de formulaciones muy amigables para ser resueltas en el computador. En el caso de problemas de ingeniería basados en modelos con una fuerte base mecánica. como por ejemplo, la mecánica de suelos, la mecánica de sólidos, la mecánica de fluidos los PVF asociados suelen resolverse por métodos que forman la base de la

denominada Mecánica Computacional. Por ejemplo los métodos de desplazamientos y fuerzas utilizados para el análisis de edificios y otras estructuras, pertenecen a la mecánica computacional. Similarmente en la solución de problemas mas complejos se usan los métodos de elementos finitos (FEM por sus siglas en ingles), los métodos de diferencias finitas (FD por sus siglas en ingles) y en una mayor proporción el método de elementos de frontera (BEM por sus siglas en ingles).

En este capitulo revisamos, aunque no de manera profunda pero sí muy practica, algunas tecnicas numéricas que son fundamentales en la formulación de métodos típicos de la Mecánica Computacional como los ya descritos. En particular estaremos cubriendo los siguientes métodos:

- Solución de ecuaciones no lineales (determinación de raices)
- Interpolación
- Integración numérica

Al final de este capitulo el estudiante estará en la capacidad de:

- Reconocer el concepto de raíz de una función (o conjunto de funciones) y manejar los métodos básicos para su determinación en funciones escalares de una variable.
- Resolver manualmente y/o con la ayuda de implementaciones en el computador problemas de determinación de raíces.
- Reconocer el problema de interpolación como uno de aproximación de funciones que originalmente se desconocen en forma cerrada.
- Manejar los aspectos matemáticos fundamentales del método de interpolación de Lagrange.
- Resolver manualmente y/o con la ayuda de implementaciones en el computador problemas de aproximación de funciones a través de métodos de interpolación.

- Reconocer los principios matemáticos básicos del problema de integración numérica de funciones sobre dominios en una y dos dimensiones.
- Resolver manualmente y/o con la ayuda de implementaciones en el computador problemas de calculo numérico de integrales de funciones.

1.1. Raíces de Ecuaciones

El problema de determinación de raíces (o de los ceros) de una función se presenta frecuentemente en diferentes aplicaciones de ingeniería y física. Acá discutiremos 2 métodos que pueden considerarse clásicos, como son el método de bisección y el método de Newton-Raphson. El primero tiene valor ya que es bastante intuitivo aunque no siempre eficaz, mientras que el segundo, al desprenderse de la expansión de una función en su serie de Taylor permite generalizarse a problemas de sistemas de ecuaciones.

1.1.1. Planteamiento del problema

Consideremos una función escrita de la forma

$$y = f(x) \tag{1.1}$$

la cual seguramente sabemos interpretar como una regla que transforma valores de la variable independiente x en valores de la variable dependiente y y donde $f(x)$ es la regla de la transformación, ver figura 1.1.

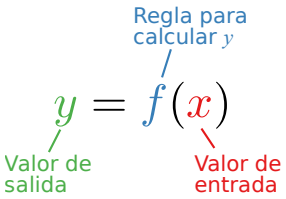


Figura 1.1. Descripción del concepto de función como una regla de transformación de valores (posiblemente) reales en otros valores (posiblemente) reales.

Por ejemplo un caso partícular de una regla de transformación o función puede ser de la forma:

$$f(x) = \ x^3 + 4x^2 - 10\,.$$

El problema de busqueda de raíces de la función consiste en encontrar valores de la variable independiente x que cuando sean transformados por la regla $f(x)$ produzcan valores nulos de y. Matematicamente esta pregunta la podemos plantear como econtrar valores de x que satisfagan la condición:

$$f(x) = 0.0\,.$$

Por ejemplo la figura 1.2 muestra la variación de la función

$$f(x) = \ x^3 + 4x^2 - 10\,,$$

para valores de x en el intervalo $[-4, 2]$. En la gráfica el círculo blanco correspondiente a la intercepción de la línea $y = 0$ y la función corresponde a una raíz de la función. Esta tiene un valor aproximado de $x = 1.4$.

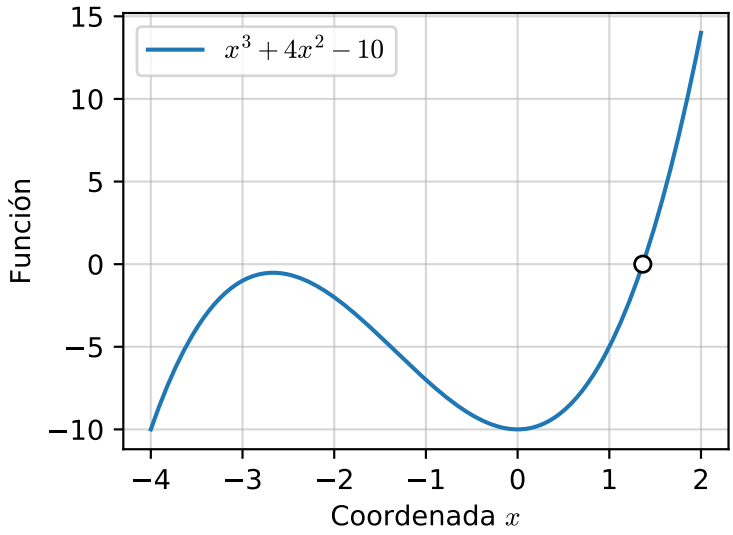


Figura 1.2. Concepto de raíz de una función. La función sobre el intervalo $[-4.0, 2.0]$ se muestra con la línea azul y la raíz con el círculo blanco.

La función puede tener varias raíces reales (positivas o negativas) o inclusive varias raíces complejas.

Ejemplo: Determinación de caudal. Un problema típico en ingeniería hidráulica es el de determinar el caudal Q con el que es necesario alimentar una turbo-máquina de potencia especifica ϕ, resistencia total a fluir R_T la cual está localizada entre 2 niveles con salto ΔH. Usando argumentos de conservación de energía se puede demostrar que este caudal satisface la condición:

$$R_T Q^3 - \Delta H Q + \phi = 0. \tag{1.2}$$

Claramente, se trata de un problema de determinación de raices tras encontrar el caudal Q que satisfaga la condició::

$$f(Q) = 0.0\,,$$

donde

$$f(Q) = R_T Q^3 - \Delta H Q + \phi. \tag{1.3}$$

> En computación se denomina iteración la operación de un procedimiento que se aplica al resultado de una operación previa.

En general, los métodos de determinación de raíces son iterativos y requieren de una estimación inicial de la raíz (o raíces). De acuerdo con dicha estimación se pueden tener los siguientes resultados (ver figura 1.3):

- Falla de convergencia.
- Convergencia a un valor incorrecto.
- Convergencia a un valor correcto.

Antes de iniciar el estudio formal de un par de métodos clásicos para determinación de raíces es importate tener una idea intuitiva de la definición de convergencia y divergencia. Concentremonos en la figura 1.3 en la cual se grafica la variación del error vs el numéro de iteraciones en un algoritmo. Concretamente la convergencia describe la velocidad con la que un cálculo iterativo se aproxima a una solución. La gráfica presenta el porcentaje de error contra el número de iteraciones requeridas para predecir la raíz $x = 1.3652305$ por medio de los 2 métodos que estudiaremos en esta sección a saber el método de bisección y el método de Newton.

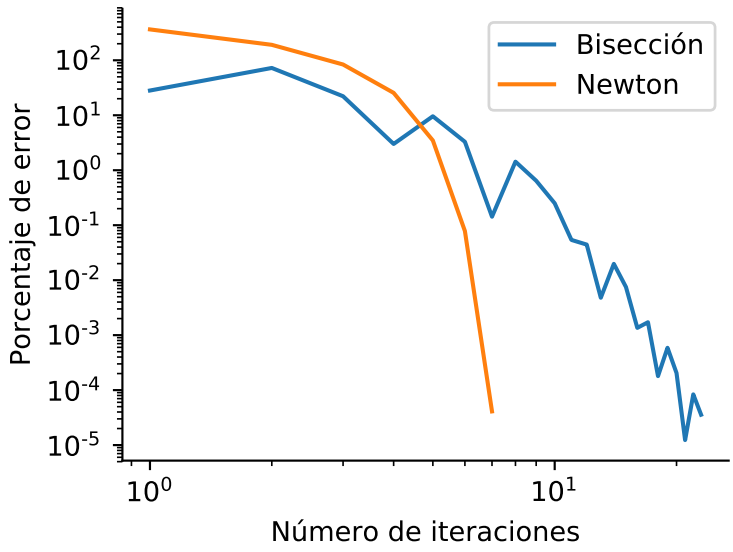


Figura 1.3. Concepto de convergencia o número de iteraciones requeridas para alcanzar una solución.

1.1.2. Localización incremental de raíces

El cálculo de las raíces de una ecuación involucra 2 procesos que se discutirán a continuación. Inicialmente se identifican de manera aproximada las localizaciones de las raíces en un intervalo determinado. Este proceso se denomina **detección** o **bracketing**. El proceso de detección arroja valores de las raíces con baja precisión.

Para mejorar la precisión se ejecuta un segundo paso tras seleccionar una precisión especificada mediante una **tolerancia** o valor de referencia. La tolerancia indica cual es la definiciòn aceptable de cero para determinar si efectivamente se encontró la raíz. Esta es necesaria yá que en el computador solo es posible especificar el cero hasta cierto número de cifras significativas, en otras palabras en el computador no existe el cero matemático. El segundo proceso corresponde entonces al acercamiento a la raíz con una precisión dada y definida en términos de la tolerancia. Este segundo paso se denomina como el de **determinación** de la raíz. En resumen, el proceso de busqueda

de raíces implica 2 pasos:

i. Localización inicial de las raíces o **Detección**.

ii. Acercamiento o mejoramiento en la precisión del valor de la raíz o **Determinación**.

En el proceso de detección de la raíz la idea fundamental consiste en recorrer la función por intervalos de tamaño Δx y con limites x_1 y x_2. Si durante el recorrido se encuentra que los valores $f(x_1)$ y $f(x_2)$ de las funciones tienen signos diferentes, entonces se concluye que existe al menos una raíz en el intervalo. El proceso se resume en los siguientes 2 pasos:

i Recorrido del dominio por intervalos Δx.

ii Evaluación de la función para detectar cambios de signo.

Para explicar el proceso y su eventual solución en el computador tomemos como referencia la figura 1.4 en la que se presenta una función que tiene 2 raíces (circulos negros) en el intervalo $[a, b]$. El proceso de detección se describe además en el algoritmo 1. Los datos de entrada al problema son los extremos del intervalo correspondientes a $x = a$ y $x = b$; el número de subintervalos N en los que se partirá el dominio del problema; y la funciónn $f(x)$ cuya raíz se desea determinar. El algoritmo entregará como resultado las raices encontradas, las cuales almacenará en un vector denominado x_R. En el primer paso del algoritmo calculamos el tamaño del subintervalo Δx y fijamos el contador de detección de raíces *iroot* en cero. Este contador se inceremen tará en 1 cada que el algoritmo detecte una raíz.

La parte principal del algoritmo consiste en un ciclo que recorre los N sub-intervalos en los que se ha partido el problema y detecta cambios de signo hacieno evaluaciones del producto $f(x_1) \times f(x_2)$. Si se da la condición $f(x_1) \times f(x_2) < 0.0$ entonces se concluye que existe una raíz en algún punto entre $x = x_1$ y $x = x_1 + \Delta x$. Este evento queda registrado en el contador *iroot*. Al mismo tiempo se selecciona el inicio del intervalo como el de localización

aproximada de la raíz. Esta se almacena en la posición *irrot* del vector x_R . Este algoritmo sencillo se da como referencia en el apendice de estas notas.

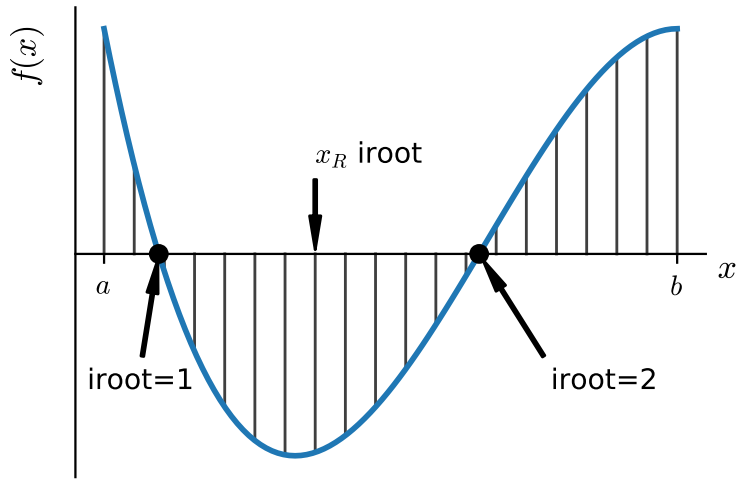


Figura 1.4. Proceso de detección de raíces de una función en un intervalo $[a, b]$.

Algoritmo 1: Detección o *bracketing*.

Data: $a,\ b,\ N,\ f$
Result: x_R: Vector con la aproximación de las raíces

$\Delta x \leftarrow \dfrac{b-a}{N-1}$;
$iroot \leftarrow 0$;
$x_2 \leftarrow a$;
for $i \leftarrow 0$ *to* $N-1$ **do**
  $x_1 \leftarrow x_2$;
  $x_2 \leftarrow x_1 + \Delta x$;
  if $f(x_1) \times f(x_2) < 0.0$ **then**
    $iroot \leftarrow iroot + 1$;
    $x_R[i] \leftarrow x_1$;
  end
end

La aplicación del código anterior a la función $f(x) = x^3 + 4x^2 - 10$ en el intervalo comprendido entre $a = -10.0$ y $b = 10.0$ con $\Delta x = 1.0$ produce el vector de raíces x_R

```
A change of sign was found.
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

el cual indica que la función tiene un cero en el intervalo $[1.0, 2.0]$ y localizado aproximadamente en $x = 1.0$. Este valor de la raíz debe ser posteriormente refinado para encontrar el valor consistente con la tolerancia prescrita. En las secciones que se presentan a continuación discutiremos 2 métodos clásicos usados para realizar este refiniamiento de la raíz.

1.1.3. Método de bisección

En este paso se asume que ya se ha encerrado una raíz de $f(x) = 0.0$ en el intervalo (x_1, x_2) mediante un proceso previo de detección. Tal y como se describe en la figura 1.5, la raíz se encuentra en algún punto entre x_1 y x_2. Mas aún, el algoritmo de detección entrega como localización aproximada de la raíz $x = x_1$. El siguiente paso consiste entonces en acercarse a la raíz con una precisión deseada. El método de bisección consiste en la partición por mitades del intervalo Δx en el que se encuentra localizada la raíz alternando entre posiciones de cambio de signo. Cada que se detecta el cambio de signo se redefine el intervalo de la raíz y esta partición continúa hasta que el intervalo se haga lo suficientemente pequeño. El algoritmo localiza inicialmente el punto intermedio $x_3 = \frac{1}{2}(x_1 + x_2)$ ubicado entre los extremos del intervalo x_1 y x_2.

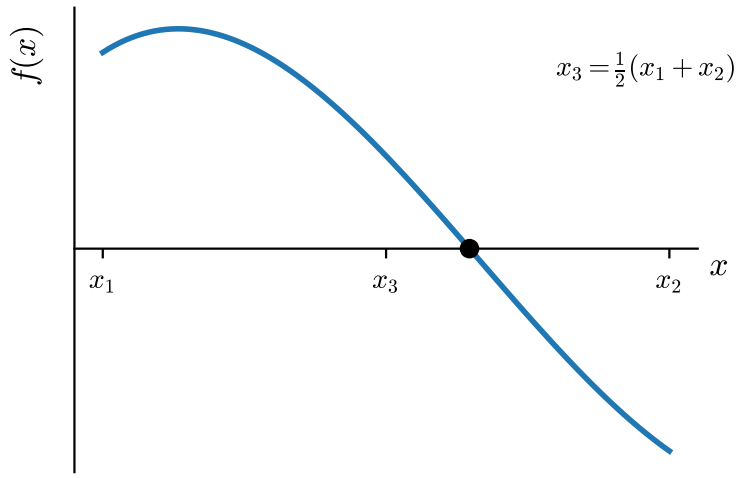


Figura 1.5. Método de bisección para encontrar las raíces.

Si $f(x_1) \times f(x_3) < 0.0$ entonces existe una raíz entre x_1 y x_3 en cuyo caso se hace $x_2 \leftarrow x_3$ y se calcula un nuevo $x_3 = \frac{1}{2}(x_1 + x_2)$ dividiendo a la mitad el intervalo que contiene a la raíz. Si por el contrario se determina que $f(x_1) \times f(x_3) > 0.0$ entonces la raíz se localiza entre x_3 y x_2 en cuyo caso se hace $x_1 \leftarrow x_3$ y se calcula un nuevo $x_3 = \frac{1}{2}(x_1 + x_2)$ dividiendo nuevamente el intervalo. Este proceso se resume en la siguiente tabla.

Si $f(x_1) \times f(x_3) < 0.0$	
	Raíz entre x_1 y x_3
	$x_2 \leftarrow x_3$
	$x_3 \leftarrow \dfrac{1}{2}(x_1 + x_2)$
	Calcula $\Delta x \leftarrow x_3 - x_1$
Si $f(x_1) \times f(x_3) > 0.0$	
	Raíz entre x_3 y x_2
	$x_1 \leftarrow x_3$
	$x_3 \leftarrow \dfrac{1}{2}(x_1 + x_2)$
	Calcula $\Delta x \leftarrow x_2 - x_3$

El algoritmo 2 presenta el pseudocódigo para el método de bisección.

Algoritmo 2: Bisección

Data: $a,\ b,\ tol,\ f$
Result: c: Aproximación de la raíz
Set tol;
$n_{\text{máx}} \leftarrow \lceil \log_2 \left(\frac{b-a}{tol}\right) \rceil$;
for $cont \leftarrow 0$ *to* $n_{\text{máx}}$ **do**
  $c \leftarrow \frac{1}{2}(a+b)$;
  if $f(a)f(b) < 0.0$ **then**
    $b \leftarrow c$;
  else
    $a \leftarrow c$;
  end
end

En el apendice se presenta el código en Python correspondinete al algoritmo de bisección el cual arroja el siguiente resultado tras aplicarlo a la función $f(x) = x^3 + 4x^2 - 10$, sobre el intervalo [-2, 2], usando una tolerancia $tol = 10^{-4}$.

```
n: 0, x: 0.0
n: 1, x: 1.0
n: 2, x: 1.5
n: 3, x: 1.25
n: 4, x: 1.375
n: 5, x: 1.3125
n: 6, x: 1.34375
n: 7, x: 1.359375
n: 8, x: 1.3671875
n: 9, x: 1.36328125
n: 10, x: 1.365234375
n: 11, x: 1.3642578125
n: 12, x: 1.36474609375
n: 13, x: 1.36499023438
```

```
n: 14, x: 1.36511230469
n: 15, x: 1.36517333984
Maximum number of iterations reached.
1.36517333984375
```

1.1.4. Método de Newton-Raphson

Derivación matemática

Sea x_i la aproximación de orden i a la raíz buscada. Por ejemplo cuando $i = 0$ x_0 es el valor encontrado por el algoritmo de detección. Sea p el valor refinado de la raíz, el cual por lo tanto se encuentra en la vecindad de x_i. Escribiendo la función como una serie de Taylor alrededor de p se tiene:

$$f(p) = f(x_i) + f'(x_i)(p - x_i) + (p - x_i)^2 \frac{f''(x_i)}{2!} + ...$$

Luego, usando la condición $f(p) \approx 0$ y linealizando se tiene:

$$0 \approx f(x_i) + f'(x_i)(p - x_i)$$

haciendo $x_{i+1} \equiv p$ y re-escribiendo se tiene la expresión

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

correspondiente a la iteración de Newton-Raphson que permite determinar la raíz aproximada x_{i+1} a partir de la aproximación x_i.

Se denomina linealización a la operación de eliminar los terminos de orden 2 y superiores y retener solo los términos líneales en una expresión. Claramente la iteración de Newton-Raphson corresponde a la serie de Taylor linealizada de la expansión de la función alrededor de la raíz buscada.

Derivación geométrica

Geométricamente el método de Newton-Raphson consiste en extender la recta tangente en el punto actual x_i hasta que cruce el cero para luego hacer la siguiente aproximación en la abscisa de dicho punto de cruce. Este método tiene la desventaja de que requiere, además de la evaluación de la función $f(x)$, también la de su primera derivada $f'(x)$. El proceso se ilustra en la figura 1.6 de donde es evidente que para iniciar las iteraciones se requiere una aproximación inicial (x_0) a la raíz. Este valor incial es el encontrado mediante el algoritmo de detección discutido anteriormente.

Para derivar el método desde una mirada geométrica partiremos del punto x_i, encontraremos la recta tangente que pasa por el mismo y lo extenderemos hasta su cruce con cero, como ya se describió. Usando la ecuación de la pendiente, tenemos

$$m = \frac{0 - f(x_i)}{x_{i+1} - x_i},$$

pero, sabemos que la pendiente es $m = f'(x_i)$, luego

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

y si resolvemos esta ecuación para x_{i+1} obtenemos el punto para el cual la recta toma el valor de cero. Este será el nuevo punto inicial para repetir el proceso y correspondiente a la iteración de Newton-Raphson.

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \tag{1.4}$$

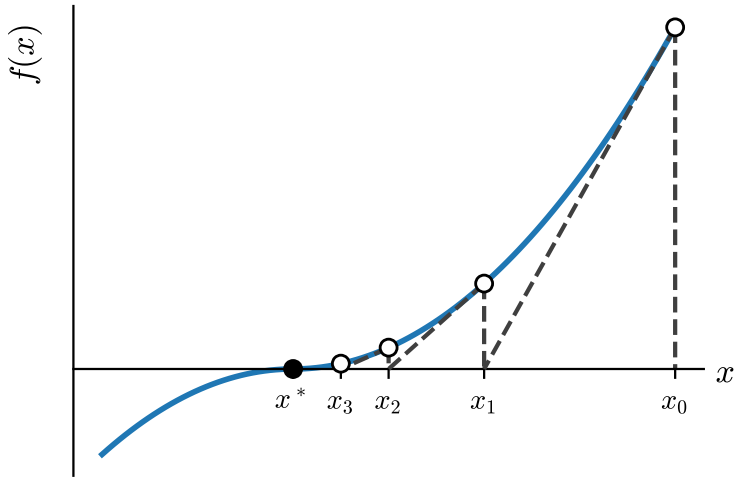


Figura 1.6. Iteración de Newton-Raphson.

El método de Newton-Raphson suele converger más rápido que el método de bisección. Esto se debe a que se incluye más información sobre la función en las iteraciones, a saber, las derivada. Por la forma que toma la iteración la función debe tener un valor diferente de cero para cada aproximación. Como se discutió anteriormente las iteraciones se detiene una vez se alcance el cero dentro de una tolerancia prescrita. El algoritmo 3 presenta el método de Newton para un punto inicial x mientras que su correspondiente código

en Python se presenta en los apendices.

Algoritmo 3: Newton-Raphson

Data: $x,\ tol,\ n_{\text{máx}},\ f,\ f'$
Result: Raíz aproximada
for $cont \leftarrow 0$ *to* $n_{\text{máx}}$ **do**
  if $|f'(x)| < tol$ **then**
    Error, división por número cercano a cero.
  end
  $x \leftarrow x - \frac{f(x)}{f'(x)}$;
  if $|f(x)| < tol$ **then**
    Pare, se llegó al valor deseado.
  end
end

Si se aplica el código correspondiente al algoritmo anterior a la función $f(x) = x^3 + 4x^2 - 10$ usando una tolerancia $tol = 10^{-8}$ y una estimación inicial de la raíz correspondiente a $x_2 = 2.0$ de acuerdo al resultado del algoritmo de detección se obtiene el siguiente resultado tras 4 iteraciones.

```
n: 0, x: 2.0
n: 1, x: 1.5
n: 2, x: 1.37333333333
n: 3, x: 1.36526201487
(1.3652300139161466, 'Root found with desired accuracy.')
```

Ejercicios

1. Detectar la localización de las raíces de la función $f(x) = x - \tan x$ (ver figura 1.7) en el intervalo $[-10.0, 10.0]$.

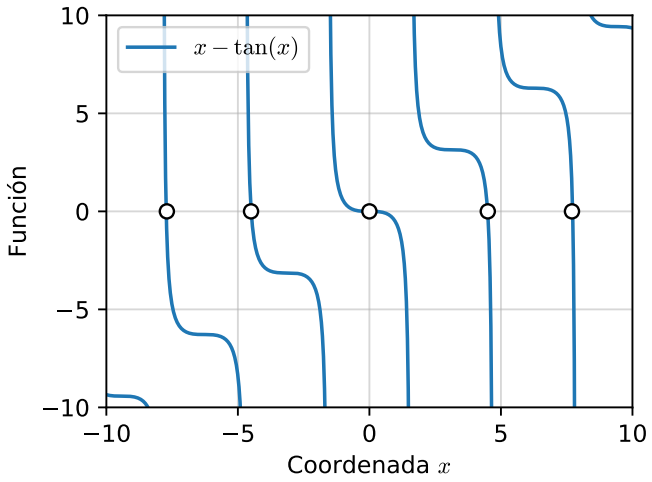


Figura 1.7. Función $f(x) = x - \tan x$.

2. Se desea determinar el caudal Q con el que es necesario alimentar una turbomáquina que inicialmente está localizada sobre una conducción con una resistencia al flujo $R_T = 149\ {}^{\mathrm{s}^2}/_{\mathrm{m}^5}$, un salto o diferencia de niveles $\Delta H = 500$ m y una potencia específica $\phi = 2548\ {}^{\mathrm{m}^4}/_{\mathrm{s}}$. Ajustar los parámetros de manera que el problema tenga solución física.

3. En un problema de diseño estructural de cimentaciones se debe determinar la dimensión en planta de una zapata cuadrada de forma tal que no se exceda el esfuerzo admisible sobre la masa de suelo en la cual se va a apoyar. Empleando la teoría de mecánica de sólidos, se puede demostrar que la dimensión de la zapata cuadrada, que no sobrepasa el esfuerzo admisible (σ_{adm}) satisface la siguiente ecuación:

$$H^3\sigma_{\mathrm{adm}} - HP \pm 6(M_1 + M_2) = 0.0 \tag{1.5}$$

Dónde:

- H : Dimensión en planta de la zapata cuadrada.
- σ_{adm} : Esfuerzo admisible.
- P : Carga vertical sobre la zapata.
- $M_1,\ M_2$: Momentos flectores al rededor de dos ejes ortogonales.

Se desea determinar la dimensión H de la zapata necesaria para no exceder el esfuerzo admisible $\sigma_{\text{adm}} = 18\ \text{tf/m}^2$, si la zapata se encuentra sometida a la carga $P = 32$ tf y los momentos $M_1 = 18\ \text{tf} \cdot \text{m}$ y $M_2 = 23\ \text{tf} \cdot \text{m}$.

1.2. Interpolación

El problema de aproximación de funciones a través de técnicas de interpolación aparece de manera recurrente en diferentes aplicaciones de ingeniería. Por ejemplo en el ajuste de datos de laboratorio cuando estos son relativamente estables, en la visualización de funciones y en el desarrollo de métodos de solución de problemas de valores en la frontera. La teoría de interpolación es de especial interés en el método de los elementos finitos, en el cual se obtienen valores de los desplazamientos en una serie de puntos y estos se usan, conjuntamente con métodos de interpolación para determinar deformaciones y tensiones. En esta sección se estudiarán los aspectos fundamentales del problema de interpolación de funciones a partir de polinomios de Lagrange.

1.2.1. Planteamiento del problema

Sea $f(x)$ una función desconocida analiticamente pero con valores conocidos de manera discreta en n puntos $x_1, x_2, \cdots, x_n$. Queremos conocer (o interpolar) el valor de $f(x)$ en un punto arbitrario $x \in [x_1, x_n]$ y que es diferente a uno de los $n-$puntos.

El problema de interpolación consiste precisamente en determinar el valor desconocido de $f(x)$ usando los valores conocidos $\{f_1, f_2, ..., f_n\}$. Por ejemplo,

tomemos los valores discretos de una función $f(x)$ que se tiene en la tabla 1.1 y que se muestran mediante circulos negros en la figura 1.8.

x	$f(x)$
-1.0	-7.000
-0.5	-9.125
0.00	-10.00
0.50	-8.875
1.00	-5.000

Tabla 1.1. Valores conocidos de la función $f(x)$.

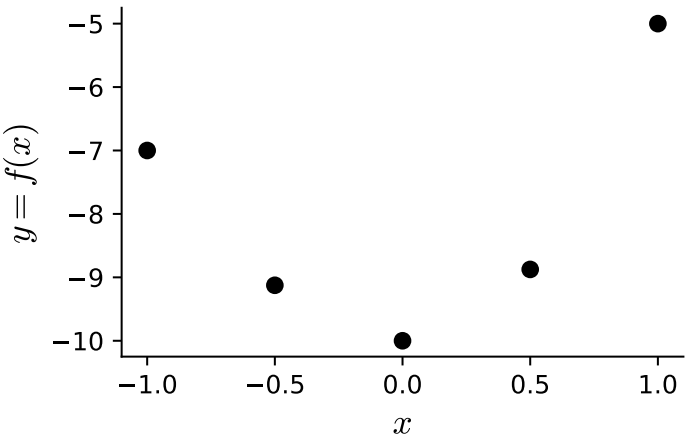


Figura 1.8. Puntos correspondientes a valores conocidos de una función. Se desea determinar el valor de la función para puntos diferentes a los de medición.

Para determinar los valores desconocidos de la función $f(x)$ usando los valores dados $\{f_1, f_2, ..., f_n\}$ el problema se resuelve en dos pasos:

1. Se propone una función de interpolación que pase por los n valores conocidos. Este paso se ilustra mediante la línea punteada de la figura 1.9. En nuestro caso, propondremos funciones polinomiales.

2. Una vez se disponga de la función de interpolación esta puede ser usada para predecir el valor en puntos arbitrarios. Si se dispone de n datos o valores conocidos de la función es posible proponer un polinomio de interpolación de orden $n-1$. Sin embargo, si se dispone de un número alto de valores conocidos de la función, puede resultar contraproducente el realizar la aproximación usando un polinomio de alto orden.

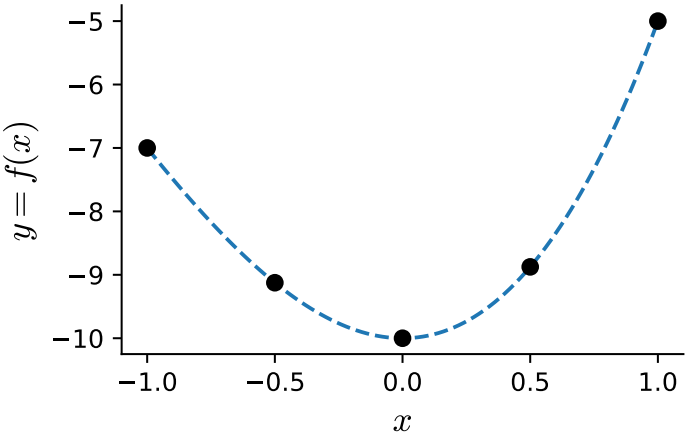


Figura 1.9. Polinomio de interpolación usando los n-datos. El polinomio resultante es de orden $n-1$.

Alternativamente, es posible dividir el dominio del problema en subdominios y proceder con interpolaciones locales. Esta alternativa se muestra en la figura 1.10 en la cual se realiza una interpolación lineal entre pares de puntos.

Ejemplo: Una aplicación en ingeniería civil. En este ejemplo se presenta a manera de pregunta un problema típico de interpolación en Ingeniería Civil. En este caso la interpolación requerida es para una fución en el plano. Como veremos de manera formal mas adelante, la solución de este tipo de problemas usa como ingrediente fundamental la solución al problema de interpolación de funciones uni-dimensionales.

La figura 1.11 muestra la distribución de los equipos de registro sísmico que conforman la red acelerográfica de Medellín. Tras la ocurrencia de un

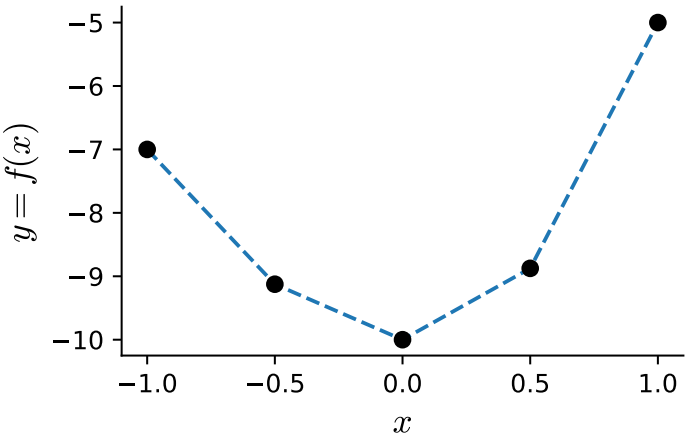


Figura 1.10. Función de interpolación definida por tramos usando pares de puntos para producir polinomios locales de orden 1.

evento sísmico cada uno de estos equipos registra el movimiento del terreno en el sitio particular. Dichos movimientos son usados posteriormente para calcular espectros de respuesta para uso en análisis y diseño de estructuras sismo-resistentes.

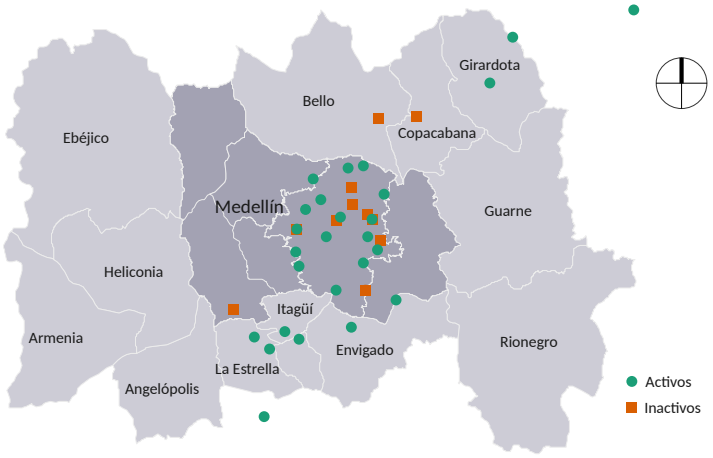


Figura 1.11. Red acelerográfica de Medellín. Los círculos verdes representan estaciones activas. Mientras que los cuadrados naranja son estaciones inactivas. Datos de julio de 2017, tomados de SIATA [**?**]

Como se aprecia en la figura, la cobertura espacial es limitada, lo que dificulta el problema de análisis estructural en zonas de la ciudad donde no se disponga de instrumentación. Se requiere entonces usar una alternativa numérica que permita determinar los espectros de respuesta en sitios carentes de instrumentación de manera consistente con los resultados instrumentales. En términos matemáticos el problema corresponde a uno de interpolación o cálculo de valores de una función en un punto arbitrario usando valores conocidos de la función en un número limitado de puntos. En esta sección se estudiarán los aspectos teóricos fundamentales para resolver problemas de interpolación sobre dominios unidimensionales (1D) y bidimensionales (2D). Inicialmente se estudiará el método clásico de interpolación a partir de los polinomios de Lagrange y posteriormente, se discutirá la generalización de este esquema a dominios 2D (como en el problema de la red acelerográfica de Medellín). Posteriormente se discutirán algunas patologías o inconvenientes numéricos propios del problema.

1.2.2. Teorema de interpolación de Lagrange

Dado un conjunto de n-puntos $\{(x_1, y_1), \cdots, (x_n, y_n)\}$, donde $y_n \equiv f(x_n)$ entonces: existe un único polinomio $p(x)$ de orden a lo sumo $(n-1)$ tal que $p(x_i) = f(x_i)$ para $i = 1, 2, \cdots, n$.

El polinomio está dado por

$$p(x) = L_1(x)f(x_1) + L_2(x)f(x_2) + ... + L_n(x)f(x_n)\,, \tag{1.6}$$

para $i = 1, 2, \cdots, n$ donde

$$L_i(x) = \prod_{\substack{j=1 \\ i \neq j}}^{n} \frac{(x - x_j)}{(x_i - x_j)}\,, \tag{1.7}$$

y donde se debe notar que

$$L_i(x_j) = \begin{cases} 1, & \text{si } i = j \\ 0, & \text{si } i \neq j \,. \end{cases}$$

En el lenguaje matemático es comun denominar la función $p(x)$ con la que se aproxima la función $f(x)$ como **polinomio de interpoalción** mientras que cada uno de los polinomios de Lagrange del tipo $L_i(x)$ se denominan **polinomios interpolantes** aunque también es posible encontrarlos como polinomios de interpolación. Será fácil reconocer quien es quien de acuerdo al contexto. De otro lado, en el lenguaje del método de los elementos finitos donde es frecuente el uso de técnicas de interpolación es común denominar a los polinomios interpolantes como funciones de forma o funciones base y a los puntos donde la función es conocida como nodos.

Ejemplo: Interpolación de una fución usando 3 valores conocidos. El tabla 1.2 contiene los valores exactos de la función $f(x) = x^3 + 4x^2 - 10$ en los puntos $x^1 = -1.0$, $x^2 = +1.0$ y $x^3 = 0.0$. Usando polinomios de interpolación de Lagrange proponer una función de interpolación que permita conocer el valor de la función y su primera derivada en $x = 0.7$.

x	$f(x)$
-1.0	-7.000
0.00	-10.00
1.00	-5.000

Tabla 1.2. Valores conocidos de la función $f(x) = x^3 + 4x^2 - 10$.

Inicialmente determinemos los polinomios interpolantes correspondientes a los puntos $x_1 = -1.0$, $x_2 = 1.0$ y $x_3 = 0.0$. Usando la ecuación (1.7), obtenemos

$$L_1(x) = \frac{(x-x_2)(x-x_3)}{(x_1-x_2)(x_1-x_3)} = -\frac{1}{2}(1-x)x$$
$$L_2(x) = \frac{(x-x_1)(x-x_3)}{(x_2-x_1)(x_2-x_3)} = \frac{1}{2}(1+x)x$$
$$L_3(x) = \frac{(x-x_1)(x-x_2)}{(x_3-x_1)(x_3-x_2)} = 1-x^2$$

Los polinomios $L_1(x)$, $L_2(x)$ y $L_3(x)$ se muestran en la figura 1.12. En esta es posible identificar que cada polinomio interpolante toma un valor unitario en su nodo correspondiente y un valor nulo en los demás nodos.

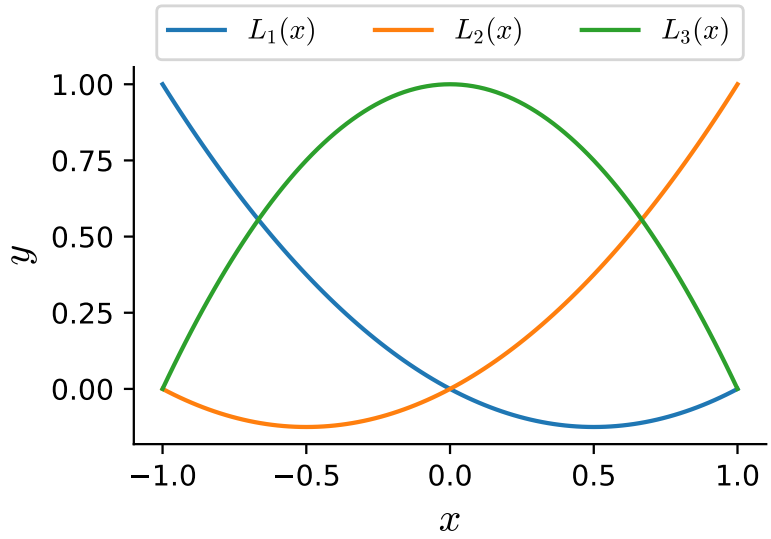


Figura 1.12. Polinomios interpolantes de Lagrange correspondientes a los puntos $x_1 = -1.0$, $x_2 = 1.0$ y $x_3 = 0.0$

La función o polinomio de interpolación resultante tras realizar la combinación lineal de estos polinomios de acuerdo con

$$p(x) = L_1(x)f_1 + L_2(x)f_2 + L_3(x)f_3\,,$$

es

$$\begin{aligned} p(x) &= 10x^2 + \frac{7}{2}(1-x)x - \frac{5}{2}(1+x)x - 10\,. \\ &= 4x^2 + x - 10 \end{aligned}$$

La función resultante $p(x)$, se compara con la función original $f(x)$ en la figura 1.13. Podemos ver que $p(x)$ (polinomio de orden 2) y $f(x)$ (polinomio de orden 3) no son iguales a largo del dominio. Sin embargo, estas funciones coinciden en los puntos donde $f(x)$ se conocía originalmente.

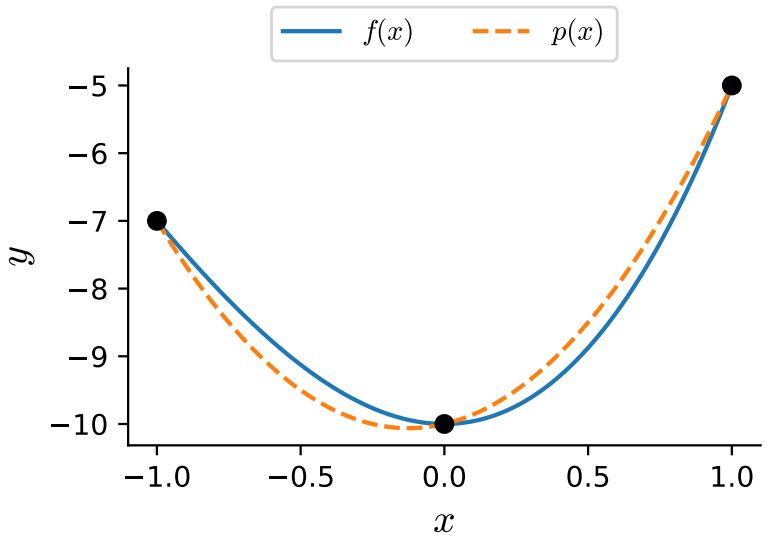


Figura 1.13. Función de interpolación (en línea punteada) resultante tras aplicar la superposición de la ecuación (1.7) conjuntamente con los datos de la tabla 1.2 (puntos negros) y correspondientes a la función $f(x) = x^3 + 4x^2 - 10$ (línea continua) $x_1 = -1.0$, $x_2 = 1.0$ y $x_3 = 0.0$

Supongamos que además se desea calcular una aproximación a la primera derivada de la función. Para calcular dichas aproximaciones derivamos la

ecuación (1.7) para obtener

$$\frac{\mathrm{d}p(x)}{\mathrm{d}x} = \frac{\mathrm{d}L_1(x)}{\mathrm{d}x} f_1 + \frac{\mathrm{d}L_2(x)}{\mathrm{d}x} f_2 + \frac{\mathrm{d}L_3(x)}{\mathrm{d}x} f_3 \tag{1.8}$$

La aproximación resulta ser una combinación lineal de productos entre polinomios interpolantes (que en este caso resultan ser derivadas de los polinomios de Lagrange) y valores conocidos de la función. La derivada obtenida de esta manera se compara con la derivada exacta en la figura 1.14.

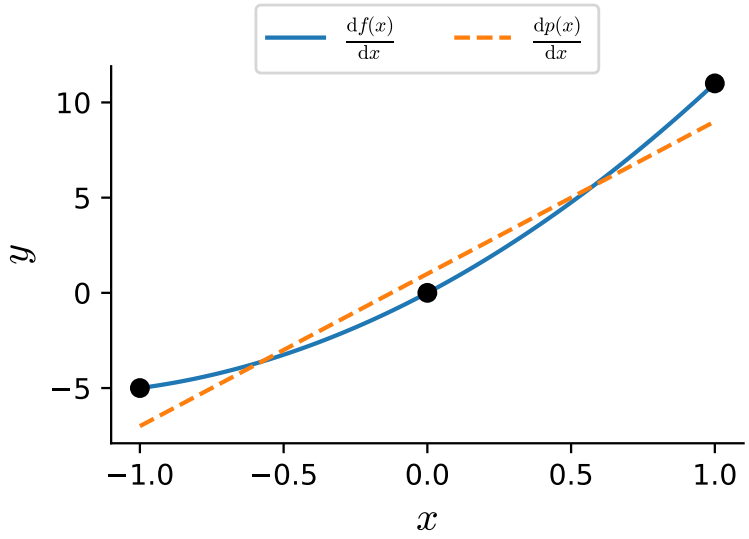


Figura 1.14. Comparación entre la derivada exacta de $f(x)$ y la derivada obtenida a partir de la aproximación con polinomios de Lagrange usando la ecuación (1.8).

Se puede observar como ahora la aproximación a la derivada, aunque se acerca a los valores reales obtenidos analíticamente, no coincide con estos últimos. Esta pérdida de precisión se debe a que en la superposición dada por la ecuación (1.8) se usan los valores conocidos de f y no los de $\frac{\mathrm{d}f(x)}{\mathrm{d}x}$ y se usan como polinomios $\frac{\mathrm{d}L_i(x)}{\mathrm{d}x}$, los cuales no satisfacen la propiedad de

interpolación, es decir

$$\frac{\mathrm{d}L_i(x_j)}{\mathrm{d}x} \neq \begin{cases} 1, & \text{si } i = j\,, \\ 0, & \text{si } i \neq j\,. \end{cases}$$

Con el propósito de asimilar la variación de la solución con los diferentes esquemas de interpolación, la figura 1.15 compara la interpolación obtenida por medio de polinomios de orden 1, 2 y 4 para la función $f(x) = 2e^x + \sin(3x)$. La columna izquierda muestra los polinomios interpolantes y la columna derecha los polinomios base $L_i(x)$.

Si en un problema de ingeniería se aproxima, por medio de métodos de interpolación, la función de desplazamientos entonces también es posible determinar la aproximación a la función de deformaciones. Sin embargo esta última sería una función de mas bajo nivel de aproximación.

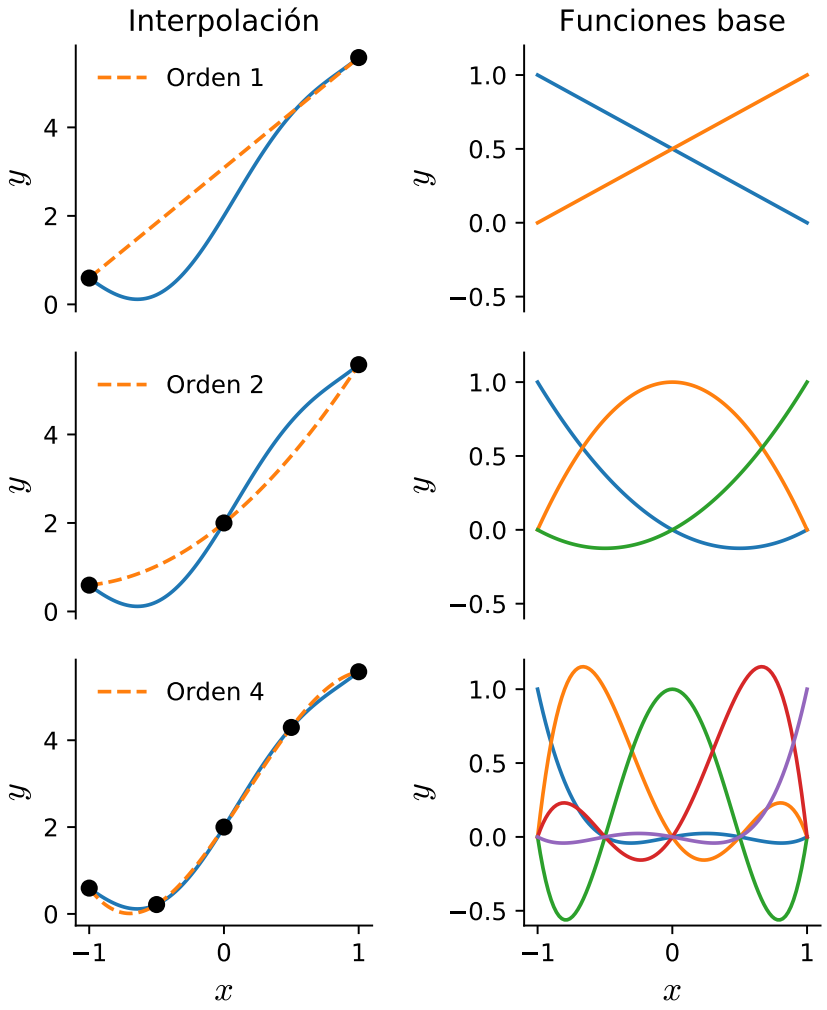


Figura 1.15. Interpolación para la función $f(x) = 2e^x + \sin(3x)$ mediante polinomios de diferente orden.

Finalmente, la figura 1.16 muestra las primeras derivadas de los polinomios de interpolación de orden 4 así como la aproximación a la primera derivada usando la expresión

$$\frac{\mathrm{d}p(x)}{\mathrm{d}x} = \sum_i \frac{\mathrm{d}L_I(x)}{\mathrm{d}x} f_i \,.$$

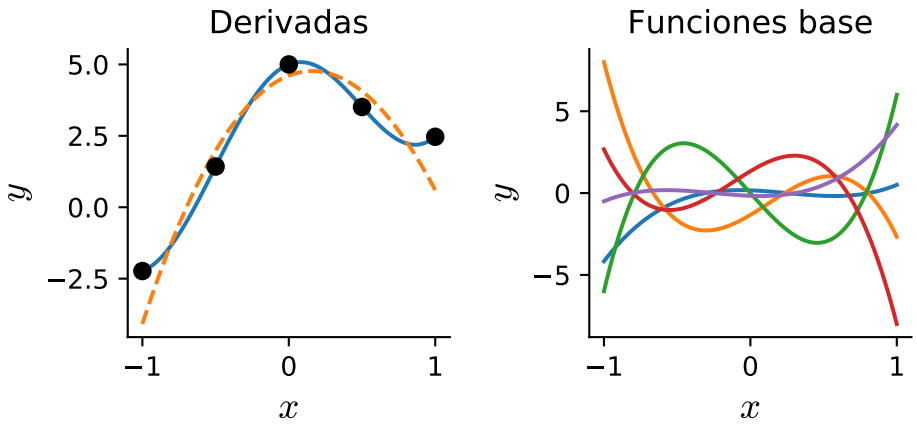


Figura 1.16. Derivadads de los polinomios de interpolación de orden 4 y aproximación a la primera derivada de la función $f(x) = 2e^x + \sin(3x)$.

1.2.3. Distribución de los puntos de muestreo

Considere la función

$$f(x) = \frac{1}{1 + 25x^2} \,,$$

mostrada en la figura 1.17 y en la cual los 11 puntos negros corresponden a nodos o puntos de muestreo donde esta se supone conocida.

Los puntos están separados por una distancia constante $\Delta x = 0.2$. Se desea aproximar la función mediante polinomios de Lagrange usando los 11 puntos de muestreo.

La figura 1.18 muestra los 11 polinomios de Lagrange de orden 10 asociados con los nodos del dominio así como el polinomio de interpolación resultante para aproximar la función. La aproximación es imprecisa en los

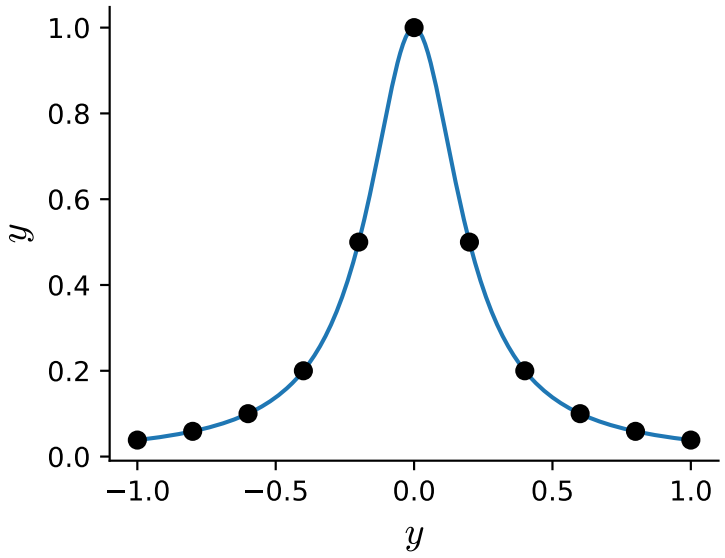


Figura 1.17. Función de Runge $f(x) = \frac{1}{1+25x^2}$.

extremos del intervalo donde se presentan unas fuertes oscilaciones debidas a la distribución equidistante de los nudos.

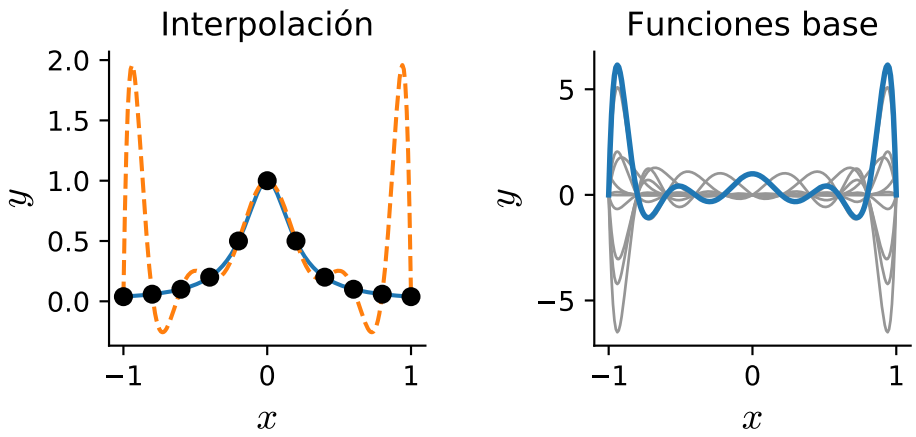


Figura 1.18. Función de interpolación para aproximar la función de Runge usando los puntos de muestreo mostrados en figura 1.17. A la derecha se muestran las funciones base para esta interpolación, se resalta la función de interpolación para el nodo de la mitad.

La aproximación se puede mejorar variando el espaciamiento de los puntos de muestreo como se muestra en la figura 1.19 donde los nodos tienen una mayor densidad en los extremos del dominio de solución.

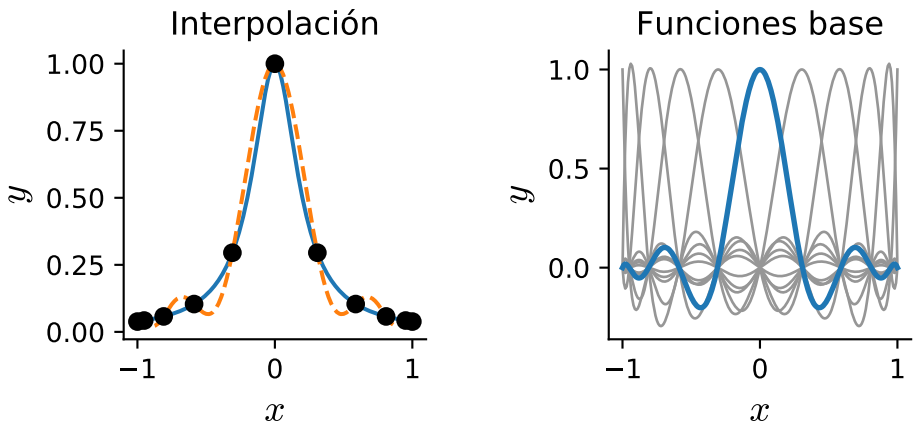


Figura 1.19. Función de interpolación para aproximar la función de Runge usando los puntos de muestreo no equidistantes. A la derecha se muestran las funciones base para esta interpolación, se resalta la función de interpolación para el nodo de la mitad. Note que las funciones base toman valores mucho más pequeños que en el caso equidistante.

Para entender esta patología numérica, relacionada con la distribución de los puntos de muestreo, consideremos los polinomios asociados con el punto central y extremo de la distribución equidistante (figura 1.18) que se muestran en la parte (a) de la figura 1.20. La traza verde corresponde al polinomio asociado con el nudo central, mientras que la traza azul corresponde al asociado con el nudo extremo. Claramente el polinomio del nudo central introduce la fuerte variación concentrada sobre los extremos del intervalo, mientras que el polinomio del nudo extremo presenta una variación relativamente suave. De manera analoga, la parte (b) de la figura muestra los polinomios central (traza verde) y extremo (traza azul) asociados con la distribución variable de nodos usada para la interpolación de mayor precisión. En este caso ambos polinomios corresponden a una función suave sin concentrar la fuerte oscilación sobre el extremo del intervalo.

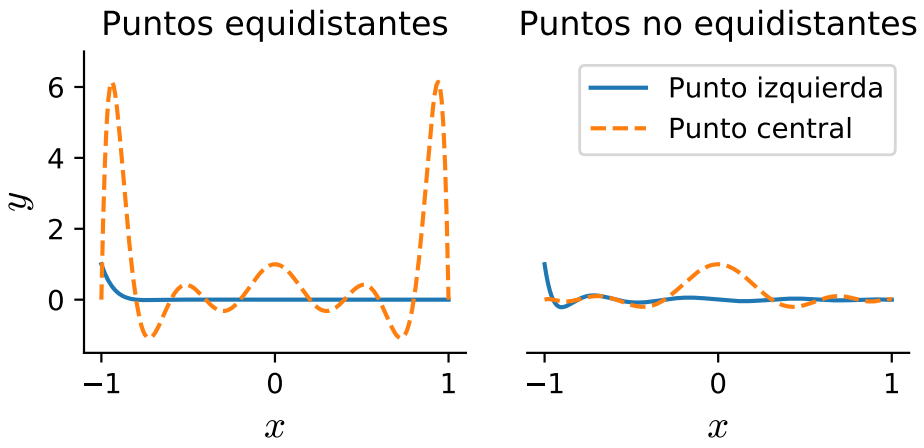


Figura 1.20. Polinomios de Lagrange de orden 10 asociados con el punto inicial y el punto medio para diferentes muestreos.

1.2.4. Interpolación local usando una función a tramos

Como alternativa a la estrategia de usar todos los puntos de muestreo para realizar la interpolación, también existe la posibilidad de subdividir el intervalo de solución $[x_1, x_n]$, en sub-dominios y realizar la interpolación localmente en cada uno de estos. Por ejemplo la tabla 1.3 muestra la partición del intervalo $[-1.0, 1.0]$ en sub-dominios conformados por pares de puntos de muestreo consecutivos. En la misma tabla también se muestran los valores de la función $f(x) = x^3 + 4x^2 - 10$ en cada uno de los extremos de estos sub-intervalos.

En este caso particular cada subdominio está conformado por un par de puntos y la interpolación se reduce a la determinación de la recta (polinomio de orden 1) que pasa por dichos puntos. Otras alternativas de sub-dominio son posibles, por ejemplo definidos por tres puntos permitiendo una interpolación local de orden 2.

Subdominio	Intervalo	Valores de $f(x)$
1	$[-1.0, -0.5]$	$[-7.000, -9.125]$
2	$[-0.5, 0.00]$	$[-9.125, -10.00]$
3	$[+0.0, +0.5]$	$[-10.00, -8.875]$
4	$[+0.5, +1.0]$	$[-8.875, -5.000]$

Tabla 1.3. División del intervalo $[-1.0, 1.0]$ en sub-intervalos o subdominios

La estrategia de interpolación local o por tramos implica el uso de un único conjunto de polinomios, como se muestra en la figura 1.21 para el caso de la interpolación lineal en consideración. Nótese que cada par de polinomios interpolantes de grado 1 se repiten en cada uno de los subintervalos.

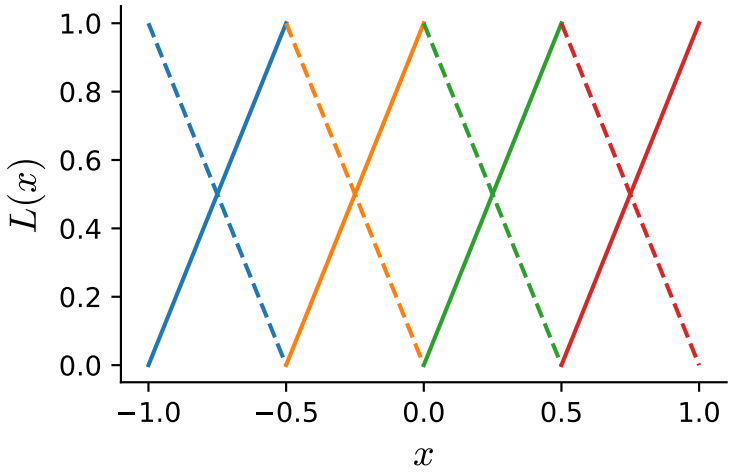


Figura 1.21. Polinomios de interpolación local. Cada dominio tiene dos líneas rectas como base. La interpolación en cada uno de ellos es la combinación lineal estas rectas.

Esta estrategia resulta en una interpolación de la función como la mostrada en la figura 1.22 en la cual se aprecia cómo el polinomio de interpolación es ahora una función definida por tramos. Como resultado de la estrategia de interpolación local la primera derivada de la función presenta discontinuidades en los límites de los subdominios, lo cual se aprecia en la parte derecha

de esta misma figura.

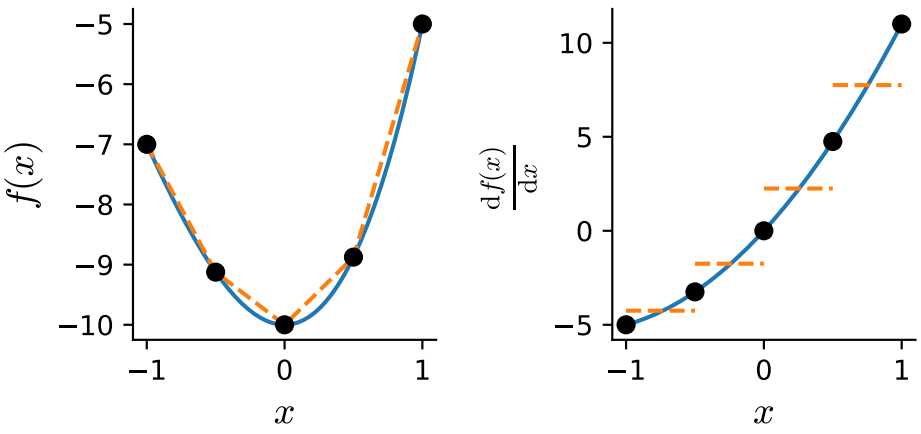


Figura 1.22. Interpolación lineal por tramos de la función $f(x) = x^3 + 4x^2 - 10$.

1.2.5. Generalización a dominios bi-dimensionales

Asumamos que estamos interesados en interpolar una función definida en un dominio plano (ver figura 1.23) en el que cada punto se encuentra especificado por un vector posición de la forma $\mathbf{x} = (x, y)$. Por medio del problema de interpolación deseamos conocer el valor de la función f para un punto $\mathbf{x}$ suponiendo que conocemos el valor de la función en n-puntos de la forma $\{(\mathbf{x}_1, f_1), \cdots, (\mathbf{x}_n, f_n)\}$. Este problema es la base para resolver el presentado al principio de la sección relativo a la Red Acelerográfica de Medellín.

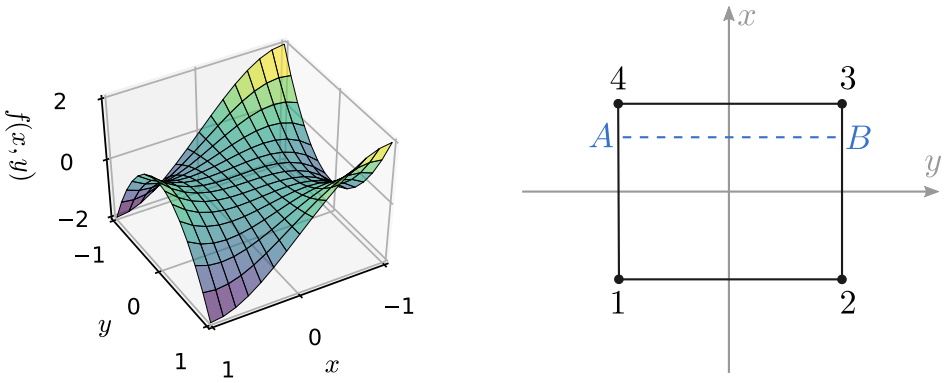


Figura 1.23. Función $f(x, y)$ sobre un dominio cuadrado con puntos de muestreo rotulados como 1, 2, 3 y 4

Para extender el esquema de interpolación planteado para el caso 1D al actual dominio 2D, fijaremos primero $x = x_A$ y realizaremos interpolación unidimensional a lo largo de la dirección y. Es decir, si consideramos la variación de la función en la dirección 1-4, equivalente a $x = x_A$, estaremos interesados en saber cual es el valor de la función para un punto arbitrario sobre esta línea y con coordenadas (x_A, y) como se ilustra en la figura 1.24.

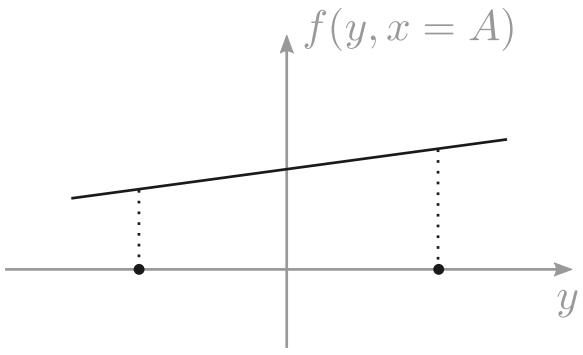


Figura 1.24. Interpolación en la dirección y sobre la línea 1-4.

Usando una expresión análoga a la ecuación (1.6) se tiene la siguiente solución

$$f(x_A, y) = L_1(y)f_1 + L_4(y)f_4\,.$$

Procediendo de manera similar a lo largo de la línea 2-3 se obtiene

$$f(x_B, y) = L_2(y)f_2 + L_3(y)f_3\,.$$

Ahora nos encontramos en la posición de hacer la interpolación en la dirección x usando las aproximaciones $f(x_A, y)$ y $f(x_B, y)$ previamente calculadas, escribiendo

$$\begin{aligned} f(x,y) &= L_A(x)f(x_A,y) + L_B(x)f(x_B,y) \\ &= L_A(x)[L_1(y)f_1 + L_4(y)f_4] + L_B(x)[L_2(y)f_2 + L_3(y)f_3] \\ &= L_A(x)L_1(y)f_1 + L_A(x)L_4(y)f_4 + L_B(x)L_2(y)f_2 + L_B(x)L_3(y)f_3\,, \end{aligned}$$

donde

$$\begin{aligned} L_A(x) &\equiv L_1(x)\,, & L_B(x) &\equiv L_2(x)\,, \\ L_1(y) &\equiv L_1(y)\,, & L_2(y) &\equiv L_1(y)\,, \\ L_3(y) &\equiv L_2(y)\,, & L_4(y) &\equiv L_2(x)\,. \end{aligned}$$

Los polinomios interpolantes que capturen simultáneamente la contribución en x y en y de cada valor conocido de la función se forman entonces a partir de productos de polinomios de interpolación unidimensionales dando lugar al siguiente esquema de interpolación

$$f(x,y) = N_1(x,y)f_1 + N_2(x,y)f_2 + N_3(x,y)f_3 + N_4(x,y)f_4\,,$$

en el cual los polinomios de interpolación $N_i(x, y)$ se definen como

$$\begin{aligned} N_1(x,y) &= L_1(x)L_1(y)\,, & N_2(x,y) &= L_2(x)L_1(y)\,, \\ N_3(x,y) &= L_2(x)L_2(y)\,, & N_4(x,y) &= L_1(x)L_2(y)\,, \end{aligned}$$

o, de forma explícita,

$$N_1(x,y) = \frac{1}{4}(1-x)(1-y)\,,$$
$$N_2(x,y) = \frac{1}{4}(1+x)(1-y)\,,$$
$$N_3(x,y) = \frac{1}{4}(1+x)(1+y)\,,$$
$$N_4(x,y) = \frac{1}{4}(1-x)(1+y)\,.$$

La figura 1.25 muestra los 4 polinomios resultantes que satisfacen la propiedad

$$N_i(x^j, y^j) = \begin{cases} 1, & \text{si } i = j\,, \\ 0, & \text{si } i \neq j\,. \end{cases}$$

En el contexto del método de los elementos finitos para resolver problemas de elasticidad un elemento esta representado por un grupo de nudos, un conjunto de funciones de interpolación del tipo $N_i(x,y)$ correspondientes a cada nudo y estas se usan para aproximar el vector de desplazamientos en cualquier punto del elemento.

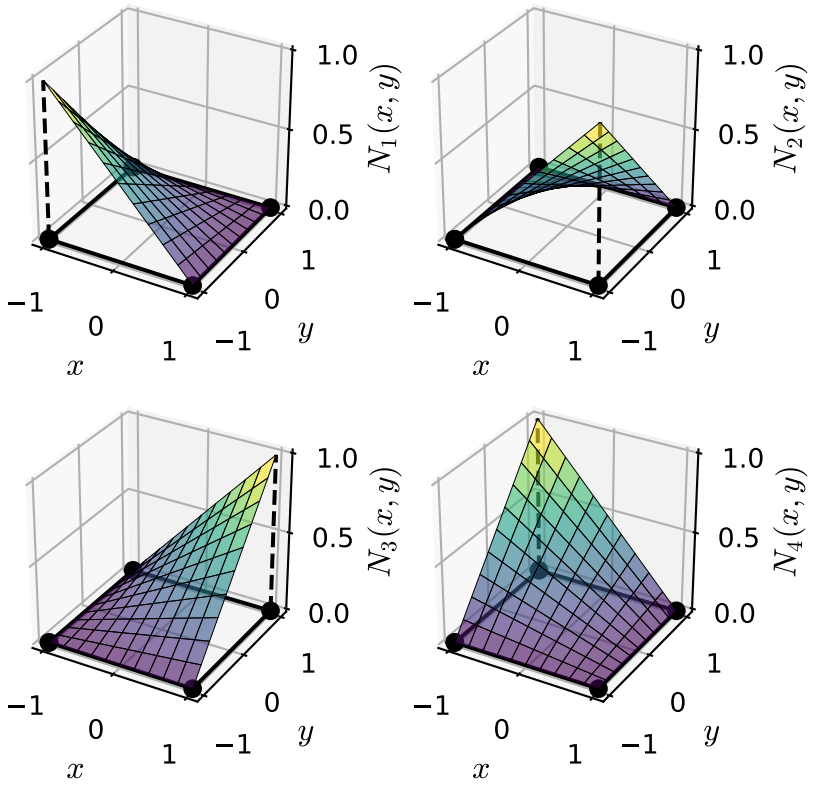


Figura 1.25. Polinomios de interpolación para un dominio bidimensional con 4 puntos de muestreo.

Ejercicios

1. Considere la función $f(x) = x^5 + 4x^3 - 8$ y realice una interpolación de orden 2. Introduzca ahora puntos adicionales de muestreo para tratar de mejorar la aproximación a la función. Compare los resultados gráficamente mostrando la función exacta, los valores en los puntos de muestreo y el polinomio de aproximación resultante. Realice también la comparación para la primera derivada de la función.

2. Considere nuevamente la función $f(x) = x^3 + 4x^2 - 10$. Genere una tabla análoga a la tabla 1.3 pero esta vez con al menos 4 sub-dominios cada uno de 3 puntos de muestreo para el intervalo $[-1.0, 1.0]$. Usando lo puntos generados resolver el problema por medio de interpolación local de orden 2. Presentar gráficamente los polinomios interpolantes en los sub-dominios así como la correspondiente función de interpolación. Comparar esta ultima con la definición exacta de la función. Adicionalmente, calcular la primera derivada a partir de la definición exacta de la función y del polinomio de interpolación.

3. Aproximar la función de Runge dada por:

$$f(x) = \frac{1}{1 + 25x^2}$$

mediante un esquema local o por tramos usando sub-dominios definidos por pares de puntos. Realizar tramos de longitud constante $\Delta x = 0.2$ y tramos de longitud variable con tamaños menores concentrados en los extremos y aumentando hacia el centro.

1.3. Integración numérica

En diferentes aplicaciones de ingeniería y física es frecuente encontrarse con la necesidad de calcular numéricamente la integral de una función, ya sea que esta se tiene definida de manera explícita y su antiderivada no lo es, o en el caso de datos experimentales, cuando la función se encuentra definida de manera discreta. En esta sección revisaremos algunas técnicas simples para abordar el problema de integración numérica de funciones. Inicialmente se

revisará (a manera de repaso) la regla extendida del trapecio, la cual es útil en el caso de funciones definidas. Posteriormente se considerará el caso de una función definida en términos de datos experimentales. Ambos problemas se abordarán para el caso de funciones de una sola variable y posteriormente estos métodos se extenderán al caso de dominios de integración bidimensionales. En la última parte de la sección se abordarán las denominadas formulas Gaussianas, las cuales resultan ser altamente poderosas y versátiles. Estas últimas son de uso común en métodos de elementos finitos y de elementos de frontera.

1.3.1. Planteamiento del problema

En el caso mas general estamos interesados en calcular numericamente integrales de la forma general

$$I = \iiint f(x, y, z) dV \tag{1.9}$$

donde la integral triple representa integración sobre un volumen determiando. De manera analoga al problema de interpolación, el problema de integración numérica también se resuelve a partir de la solución fundamental de integrar una función uni-dimensional. En cualquier caso, el problema de integración numérica se resuelve mediante una aproximación de la integral en términos de una suma ponderada de la forma:

$$\int_a^b f(x)\,\mathrm{d}x \approx \sum_{i=1}^{n} w_i f(x_i)\,, \tag{1.10}$$

A estas formulas se les denomina **cuadraturas**. En general en una cuadratura (o esquema de integración numérica) se evalúa la función $f(x)$ en n puntos de coordenadas x_i y cada valor de la función es ponderado por un factor w_i.

Por ejemplo la tabla 1.4 muestra las abscisas (puntos de evaluación) y factores de ponderación de una cuadratura en particular.

x_i	w_i
-0.86113	0.34785
-0.33998	0.65214
$+0.33998$	0.65214
$+0.86113$	0.34785

Tabla 1.4. Abscisas y factores de ponderación para calcular $\int_{-1}^{+1} f(x)\,\mathrm{d}x$.

Usando dicha cuadratura evaluaremos la integral

$$I = \int_{-1}^{+1} (x^3 + 4x^2 - 10)dx$$

La evaluación numérica de la integral se reduce entonces a calcular la siguiente suma ponderada:

$$\int_{-1}^{+1} (x^3 + 4x^2 - 10)\,\mathrm{d}x \approx 0.34785 \cdot f(-0.86113) + 0.65214 \cdot f(-0.33998)$$
$$+0.34785 \cdot f(0.86113) + 0.65214 \cdot f(0.33998) = -17.3333$$

Ejemplo: Una aplicación en ingeniería civil. Supongamos que queremos calcular el área superficial del Valle de Aburrá. Aunque no conocemos una función que describa la topografía del valle de forma analítica, podemos representar la misma como una serie de puntos con coordenadas (latitud, longitud, altitud) como se presenta en la siguiente figura.

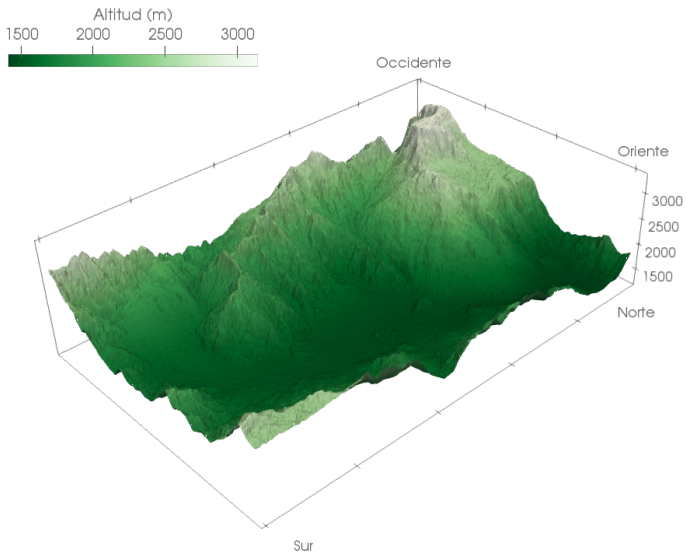


Figura 1.26. Topografía del Valle de Aburrá.

Podemos utilizar las técnicas de interpolación vistas anteriormente para calcular la integral de superficie en el valle. Con la siguiente fórmula podríamos calcular el área superficial

$$A = \int\limits_{\text{Dominio}} \sqrt{\left(\frac{\partial f(x,y)}{\partial x}\right)^2 + \left(\frac{\partial f(x,y)}{\partial y}\right)^2 + 1}\ \mathrm{d}x\,\mathrm{d}y\ ,$$

en donde las coordenadas (x, y) se corresponden con (longitud, latitud), la función $f(x, y)$ es la altitud para cada uno de los puntos, y la integral se hace sobre el dominio sobre el que está definida la superficie.

1.3.2. Integración numérica mediante polinomios de interpolación

La integración numérica se basa fundamentalmente en el ajuste de un polinomio de interpolación $p(x)$ a la función dada $f(x)$ a través de n puntos en los cuales se conoce o calcula el valor de la función, para luego usar $\int_a^b p(x)\,\mathrm{d}x$ como una aproximación a la integral de $f(x)$. El número de evaluaciones de $f(x)$, así como las posiciones de los puntos de evaluación determinan que tan buena es la aproximación de $p(x)$ a $f(x)$ y por ende el error en la aproximación de la integral.

Concretamente, en los algoritmos de integración numérica se hace la aproximación:

$$\int_a^b f(x)\,\mathrm{d}x \approx \int_a^b p(x)\,\mathrm{d}x \ ,$$

donde

$$p(x) = \sum_{i=1}^{n} L_i(x) f(x_i)$$

siendo $L_i(x)$ el polinomio interpolante de Lagrange asociado al punto x_i.

Aproximando la función mediante un polinomio de interpolación se tiene que:

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

la cual es equivalente a la ecuación (1.10) y en la cual se identifica que los factores de ponderación están dados por:

$$w_i \equiv \int_a^b L_i(x)\,\mathrm{d}x \ . \tag{1.11}$$

Este tipo de esquemas se divide en métodos de Newton-Cotes y en cuadraturas Gaussianas. En el primer caso el intervalo de integración se divide

en $n-1$ sub-intervalos iguales (delimitados por abscisas equidistantes) de tamaño $(b-a)/(n-1)$. En el segundo caso se dejan como parámetro de ajuste tanto la localización de los puntos de evaluación como los factores de ponderación, resultando en un esquema más versátil. A continuación se discuten ambas estrategias.

1.3.3. Regla del trapecio

Considere el caso particular en el que se tienen 2 puntos (1 intervalo de integración). El tamaño del intervalo en este caso corresponde a $h = b - a$ y el polinomio de interpolación de $f(x)$ está dado por:

$$p(x) = L_1(x)f_1 + L_2(x)f_2 \equiv L_1(x)f(a) + L_2(x)f(b)\,.$$

Los polinomios interpolantes en este caso son:

$$L_1(x) = \frac{(x-x_2)}{(x_1-x_2)} \equiv -\frac{1}{h}(x-b)$$
$$L_2(x) = \frac{(x-x_0)}{(x_2-x_1)} \equiv \frac{1}{h}(x-a)\,.$$

Reemplazando en (1.11) se tiene que:

$$w_1 = -\frac{1}{h}\int_a^b (x-b)\,\mathrm{d}x \equiv \frac{h}{2}\,,$$
$$w_2 = +\frac{1}{h}\int_a^b (x-a)\,\mathrm{d}x \equiv \frac{h}{2}\,,$$

de donde:

$$I = w_1 f_1 + w_2 f_2 \equiv \frac{h}{2}\left[f(a) + f(b)\right]\,.$$

Y, finalmente,

$$\int_a^b f(x)\,\mathrm{d}x \approx \frac{h}{2}[f(a) + f(b)]\,. \tag{1.12}$$

Ejemplo

Calcular

$$I = \int_{-1}^{+1} (x^3 + 4x^2 - 10)\,\mathrm{d}x \ ,$$

usando la regla del trapecio.

En este caso $h = 2.0$, luego:

$$I = f(-1) + f(1) \equiv -7 - 5 = -12$$

Considere nuevamente la regla del trapecio dada en la ecuación (1.12). Supongamos un intervalo de integración con límites $x_1 = a$ y $x_n = b$ y en el cual se conocen en total n valores de una función $f(x)$ (ver figura 1.27)

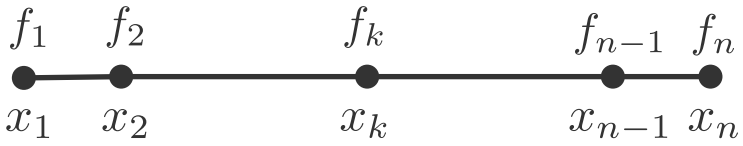


Figura 1.27. Intervalo de integración extendido con puntos de muestreo entre x_1 y x_n.

Aplicando la ecuación (1.12) $n-1$ veces para hacer la integración en los sub-intervalos (x_1, x_2), $(x_2, x_3), \cdots$, (x_{n-1}, x_n) y sumando los resultados se obtiene:

$$\int_a^b f(x)\,\mathrm{d}x = h\left[\frac{1}{2}f_1 + f_2 + f_3 + f_4 + \cdots + f_{n-1} + \frac{1}{2}f_n\right] + O\left[\frac{(b-a)^3 f''}{n^2}\right] \tag{1.13}$$

la cual es válida para calcular la integral usando los n puntos $x_1,\ x_2, \cdots, x_{n-1}, x_n$. En esta expresión h es la separación entre puntos.

En el apendice se describe la función **trapz** (correspondiente a la ecuación (1.13)) la cual integra una función $f(x)$ entre los puntos a y b. La rutina

realiza la transformación del rango $[a, b]$ al ficticio dado por $[-1, +1]$. Usando esta función en el calculo de la integral del ejemplo antrior se tiene la siguiente salida:

```
Approximation for 1 subdivisions: -12.000000
Approximation for 2 subdivisions: -16.000000
Approximation for 3 subdivisions: -16.740741
Approximation for 4 subdivisions: -17.000000
Approximation for 5 subdivisions: -17.120000
Approximation for 6 subdivisions: -17.185185
Approximation for 7 subdivisions: -17.224490
Approximation for 8 subdivisions: -17.250000
Approximation for 9 subdivisions: -17.267490
Analytic integral: -17.333333
```

En las cuadraturas de Newton-Cotes, como la regla del trapecio, los puntos de evaluación de la función son equidistantes. En las cuadraturas Gaussianas los puntos de evaluación no son equidistantes sino que estan localizados en ciertas posiciones de manera que la cuadratura sea lo mas eficiente posible.

1.3.4. Cuadraturas Gaussianas

En la cuadratura correspondiente a la regla extendida del trapecio escrita de la forma

$$\int_a^b f(x)\,\mathrm{d}x \approx \sum_{i=1}^{n} w_i f(x_i)\,, \tag{1.14}$$

los puntos de evaluación se encuentran espaciados de manera equidistante. En las cuadraturas Gaussianas se dejan como parámetros por ajustar los factores de ponderación w_i y también la localización de los puntos de evaluación x_i. Como resultado, ahora se dispone de $2n$ parámetros para ajustar y así resolver el problema de calcular la integral de $f(x)$ entre $x = a$ y $x = b$

con la máxima precisión y el mínimo número de operaciones. Este tipo de cuadraturas proveen mayor precisión que las del tipo Newton-Cotes (como las del trapecio) cuando la función a integrar es apropiadamente representable mediante un polinomio. Considerando lo anterior se tiene que por medio de una cuadratura Gaussiana es posible integrar funciones expresables como:

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

La factorización $w(x)f(x)$ es útil ya que permite expresar una función como el producto de un polinomio $f(x)$ por una función conocida $w(x)$. Esta última puede seleccionarse para remover singularidades integrables de la integral.

Las diferentes cuadraturas Gaussianas se encuentran especificadas en términos de una tabla de valores de abscisas de los puntos de integración (o de evaluación de la función a integrar) y sus correspondientes factores de ponderación (también denominados pesos). Por ejemplo la tabla 1.5 presenta las abscisas y factores de ponderación para una cuadratura de 4 puntos.

x_i	w_i
-0.86113	0.34785
-0.33998	0.65214
$+0.33998$	0.65214
$+0.86113$	0.34785

Tabla 1.5. Abscisas y factores de ponderación para calcular $\int_{-1}^{+1} f(x)\,\mathrm{d}x$

Para facilitar la codificación de las cuadraturas y permitir cálculos generales, es común considerar el intervalo de integración canónico $[-1.0, +1.0]$, por lo que es necesario transformar la integral (incluyendo la función, los límites y el diferencial) a dicho intervalo como se discute en la sección 1.3.5. La figura 1.28 esquematiza este intervalo y los correspondientes puntos de integración (denotados por los círculos blancos). En la siguiente sección se desarrolla un ejemplo completo de evaluación de una integral por medio de

una cuadratura Gaussiana tras realizar la transformación al dominio canónico.

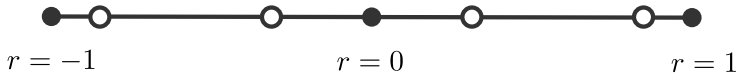


Figura 1.28. Esquematización de una cuadratura Gaussiana en el intervalo canónico $[-1.0, 1.0]$.

En las cuadraturas Gaussianas los puntos de evaluación se encuentran especificados en el intervalo $[-1.0, 1.0]$ el cual se denomina el espacio canonico. Es común denotar el espacio canonico mediante la variable r, de manera que para integrar una función en el intervalo $x \in [a, b]$ primero es necesario transformas la función y el dominio de integración al intervalo canonico $r \in [-1.0, 1.0]$.

Ejemplo de derivación de una cuadratura Gaussiana. Revisemos el proceso de derivación de una cuadratura Gaussiana. Para esto supongamos que queremos formular una cuadratura correspondiente a $n = 2$, o equivalentemente una cuadratura de 2 puntos , asumiendo que el intervalo de integración es $[a, b] = [-1, +1]$ de tal forma que coincida con el del intervalo canonico. Queremos entonces determinar los valores de los factores de ponderación w_1 y w_2 así como la localización de los puntos de evaluación x_1 y x_2 de manera que la cuadratura podamos integrar de manera exacta una función correspondiente a un polinomio de grado 3 que tiene la forma general:

$$f(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 \, .$$

En otras palabras, queremos encontrar valores apropiados de w_1, w_2, x_1 y x_2 tales que se satisfaga:

$$I = \int_{-1}^{+1} f(x)\,\mathrm{d}x \equiv w_1 f(x_1) + w_2 f(x_2)\,.$$

Usando $f(x)$ en I y planteando la integral para cada termino se obtiene:

$$I = \int_{-1}^{+1} a_0\,\mathrm{d}x + \int_{-1}^{+1} a_1 x\,\mathrm{d}x + \int_{-1}^{+1} a_2 x^2\,\mathrm{d}x + \int_{-1}^{+1} a_3 x^3\,\mathrm{d}x$$

donde identificamos un término constante, un término de orden 1, un término cuadratico y un término de orden3. Para que la cuadratura sea exacta cada uno de estos términos debe poder ser integrado de manera exacta. Usando esta condición se tiene:

$$\begin{aligned}
&\int_{-1}^{+1} \mathrm{d}x = 2 = w_1 \cdot 1 + w_2 \cdot 1\,,\\
&\int_{-1}^{+1} x\,\mathrm{d}x = 0 = w_1 \cdot x_1 + w_2 \cdot x_2\,,\\
&\int_{-1}^{+1} x^2\,\mathrm{d}x = \frac{2}{3} = w_1 \cdot x_1^2 + w_2 \cdot x_2^2\,,\\
&\int_{-1}^{+1} x^3\,\mathrm{d}x = 0 = w_1 \cdot x_1^3 + w_2 \cdot x_2^3\,.
\end{aligned}$$

Se tiene un sistema de ecuaciones en 4 incognitas correspondientes a los factores de ponderación w_1, w_2 y los puntos de evaluación x_1 y x_2. Resolviendo el sistema se tiene que $w_1 = 1$, $w_2 = 1$, $x_1 = -\sqrt{3}/3$ y $x_2 = +\sqrt{3}/3$. Usando este resultado en la expresón de la cuadratura I se tiene:

$$I = \int_{-1}^{+1} f(x)\,\mathrm{d}x \approx 1.0 \cdot f(-\sqrt{3}/3) + 1.0 \cdot f(+\sqrt{3}/3).$$

Claramente esta cuadratura es exacta si se trata de integrar funciones polinómicas de orden 3 o menor. Notese que la posibilidad de ajustar 4 paramétros (w_1, w_2, x_1, x_2) permitió integrar de manera exacta una función de orden 3. Mas adelante se demostrará que una cuadratura Gaussiana de orden n permite integrar de manera exacta funciones de orden $2n-1$. Notese también que la localización de los puntos de integración x_1 y x_2 es simetrica con respecto a $x = 0$: Mas aún, en este caso se satisface que $x_1 = -x_2$.

La idea de la cuadratura Gaussiana es extendible a polinomios de grado mayor, pero se requiere de un método efectivo para determinar los factores de ponderación y abscisas de evaluación. En la siguiente sección se presentará un método aplicable a polinomios de orden $2n$, en el que se saca provecho de la propiedad de ortogonalidad de ciertos polinomios especiales.

Polinomios ortogonales Dos polinomios $P(x)$ y $Q(x)$ donde $P(x) \neq Q(x)$ son ortogonales si:

$$\int_a^b P(x)Q(x)dx = 0.$$

Particularmente, los polinomios de Legendre definidos por

$$P_n(x) = \frac{1}{2^n n!}\frac{\mathrm{d}^n}{\mathrm{d}x^n}[(x^2-1)^n]$$

y los cuales son solución a la ecuación:

$$(1-x^2)y'' - 2xy' + n(n+1)y = 0$$

en el intervalo $[-1, +1]$ satisfacen la siguiente condición de ortogonalidad

$$\int_{-1}^{+1} Q_i(x)P_j(x)\,\mathrm{d}x = 0\,,$$

donde $Q_i(x)$ es cualquier función polinomial de grado i menor que j.

Además de la propiedad de ortogonalidad los polinomios de Legendre tiene raíces en el intervalo $(-1.0, 1.0)$ las cuales son diferentes y simétricas con respecto a cero. Como se demuestra en el siguiente teorema esta ultima propiedad hace que estas raíces puedan ser útiles para producir una cuadratura que sea exacta para integrar cualquier función polinomial de grado menor que $2n$. Por ejemplo, el segundo polinomio de Legendre dado por:

$$P_2(x) = x^2 - \frac{1}{3}$$

tiene como raíces $x_1 = -\frac{\sqrt{3}}{3}$ y $x_2 = +\frac{\sqrt{3}}{3}$ los cuales corresponden a los puntos de integración para la cuadratura exacta de grado 3 encontrada en el ejemplo anterior.

Teorema Sean $\{x_1, x_2, \cdots, x_n\}$ las raíces del polinomio de Legendre $P_n(x)$ de grado n; sea

$$w_i = \int_{-1}^{+1} \prod_{j=1}^{n} \frac{x - x_j}{x_i - x_j} \, \mathrm{d}x \ ,$$

y sea $f(x)$ una función polinomial cualquiera de grado menor que $2n$, entonces:

$$I = \int_{-1}^{+1} f(x) \, \mathrm{d}x = \sum_{i=1}^{n} w_i f(x_i) \, . \tag{1.15}$$

Demostración

1. Si $f(x)$ es de grado menor que n entonces claramente es representable en términos de polinomios de Lagrange con lo cual se satisface de manera automática la condición ecuación (1.15).

2. Si $f(x)$ es de grado menor que $2n$ entonces es representable como:

$$f(x) = Q(x)P_n(x) + R(x)$$

donde $Q(x)$ es el cociente de $f(x)/P_n(x)$ y de grado $n-1$ (o menor) y $R(x)$ es el residuo y de grado menor que n. Integrando esta representación de $f(x)$ se tiene:

$$\int_{-1}^{+1} Q(x)P_n(x)\,\mathrm{d}x + \int_{-1}^{+1} R(x)\,\mathrm{d}x \ ,$$

la cual se reduce a:

$$I = \int_{-1}^{+1} f(x)\,\mathrm{d}x = \int_{-1}^{+1} R(x)\ \,\mathrm{d}x \ ,$$

tras usar la propiedad de ortogonalidad entre $Q(x)$ y $P_n(x)$. Ahora, retomando la expresión:

$$f(x) = Q(x)P_n(x) + R(x)$$

si esta es evaluada en las raíces de los polinomios de Legendre se tiene que:

$$f(x_i) = R(x_i)$$

con lo cual queda completa la demostración.

1.3.5. Transformación del dominio de solución

La construcción de tablas con las coordenadas y factores de ponderación de las diferentes cuadraturas y su programación en el computador se facilita si estas se especifican para un rango fijo. Por diversas conveniencias matemáticas, es común usar como intervalo general el dado por $[-1, 1]$ denominado previamente como el intervalo canonico y descrito mediante una variable independiente, comunmente denotada como r. Se tiene entonces que el espacio de la variable independiente correspondiente a $x \in [a, b]$ se transforma el espacio canonico dado por $r \in [-1.0, 1.0]$. Note que esta transformación también implica una transformación de la función $f(x)$ en el espacio de x a su respresentación en el espacio de r.

Es necesario entonces reescribir la integral de $f(x)$ en el rango comprendido entre $x = a$ y $x = b$ a una integral en el espacio canonico de acuerdo con:

$$\int_{a}^{b} f(x)\,\mathrm{d}x \equiv \int_{-1}^{+1} F(r)\,\mathrm{d}r \ . \tag{1.16}$$

La transformación 1.16 se describe en la figura 1.29 y en la cual la flecha curva esquematiza la transformación entre los 2 espacios a saber, el espacio representado mediante la variable independiente x y comprendido entre $x = a$ y $x = b$ y el espacio canonico descrito mediante la variable r y comprendido entre $r = -1$ y $r = 1$.

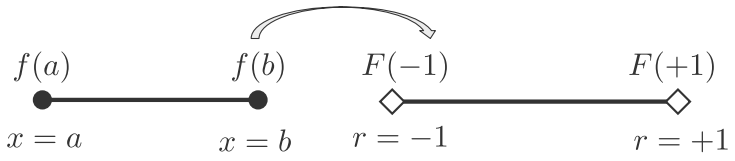


Figura 1.29. Transformación del rango de integración $[a, b]$ al intervalo canónico $[-1.0, +1.0]$.

La transformación espacial puede indicarse matemáticamente como:

$$x = x(r)\,, r = r(x)\,,$$

y en la cual $x(r)$ y $r(x)$ denotan relaciones funcionales entre los 2 espacios. Es evidente que independiente de la forma de la relación funcional esta minimamente debe satisfacer las condiciones $x(-1.0) = a$ y $x(+1) = b$. Por ejemplo, una relación relación funcional válida puede encontrarse tras asumir que ambos espacios están relacionados mediante un polinomio de interpolación de Lagrange de primer grado construido como:

$$x(r) = L_1(r)x(r_1) + L_2(r)x(r_2)\,.$$

Usando los correspondientes polinomios interpolantes:

$$L_1(r) = \frac{r - r_2}{r_1 - r_2} \equiv \frac{1}{2}(1 - r)\,,$$
$$L_2(r) = \frac{r - r_1}{r_2 - r_1} \equiv \frac{1}{2}(1 + r)\,,$$

en la expresión para $x(r)$ se tiene que:

$$x(r) = \frac{1}{2}(a + b) + \frac{r}{2}(b - a).$$

Continuando con la transformación es claro que también es necesario reescribir $f(x)$ en términos de r de acuerdo con:

$$f = f(x) \equiv f[x(r)] = \hat{F}(r)\,.$$

Finalmente, para completar la transformación es necesario transformar el diferencial de .^espacio físico", es decir dx a su correspondente diferencial en el espacio canonico dr. Procediendo directamente desde la transformación en términos de los polinomios de Lagrange se tiene que:

$$\frac{\mathrm{d}x}{\mathrm{d}r} = \frac{\mathrm{d}L_1(r)}{\mathrm{d}r}x(r_1) + \frac{\mathrm{d}L_2(r)}{\mathrm{d}r}x(r_2) \equiv \frac{1}{2}(b - a)\,,$$

y por lo tanto

$$\frac{\mathrm{d}x}{\mathrm{d}r} = \frac{1}{2}(b - a)$$

lo cual permite escribir finalmente la integral en el dominio ficticio comprendido entre $[-1, +1]$ de acuerdo con:

$$\int_a^b f(x)\,\mathrm{d}x \equiv \int_{-1}^{+1} \hat{F}(r)\frac{h}{2}\,\mathrm{d}r \equiv \int_{-1}^{+1} F(r)\,\mathrm{d}r \ ,$$

y donde $h = \frac{1}{2}(b-a)$.

Ejemplo

Usar una cuadratura Gaussiana de 2 puntos (ver tabla 1.6) para evaluar la integral:

$$I = \int_0^3 (2^x - x)\,\mathrm{d}x \ .$$

x^I	w^I
-0.5773503	1.000000
$+0.5773503$	1.000000

Tabla 1.6. Abscisas y factores de ponderación para calcular $\int_{-1}^{+1} f(r)\,\mathrm{d}r$

Para realizar la integración usando la cuadratura Gaussiana de 2 puntos dada en la tabla 1.6 es necesario transformar el rango de integración y el integrando de la función al espacio correspondiente al intervalo $[-1.0, +1.0]$. Esta transformación está dada por:

$$x(r) = \frac{3}{2} + \frac{3}{2}r$$

mientras que los elementos diferenciales satisfacen

$$\mathrm{d}x = \frac{3}{2}\,\mathrm{d}r \ .$$

Para transformar la función usamos

$$\hat{f}(r) = f[x(r)] \equiv f\left(\frac{3}{2} + \frac{3}{2}r\right) \ ,$$

de donde se tiene que

$$I = \int_0^3 (2^x - x)\,\mathrm{d}x = \int_{-1.0}^{+1.0} \left[2^{\frac{3}{2}(1+r)} - \frac{3}{2}(1+r)\right]\frac{3}{2}\,\mathrm{d}r \ ,$$

y evaluando,

$$\begin{aligned} I &= \sum_{i=1}^{2} w_i \left[2^{\frac{3}{2}(1+r_i)} - \frac{3}{2}(1+r_i)\right]\frac{3}{2} \\ &= 1.0 \cdot (1.37678967978) + 1.0 \cdot (4.18374583924) \\ &= 5.56053551 \end{aligned}$$

En el apendice se presenta a manera de función la implementación en Python de la cuadratura Gaussiana de 2 puntos. La rutina hace uso de la función **gauss1d** para integrar la función definida en $f(x)$. La rutina realiza la transformación de $[a, b]$ a $[-1, +1]$.

Si ejecutamos la rutina, obtenemos el siguiente resultado:

```
Analytic integral: -17.333333
Gauss quadrature: -17.333333
```

1.3.6. Extensión a dominios en 2D

En varias aplicaciones de ingeniería es necesario el calculo de integrales sobre dominios bi-dimensionales, posiblemente con geométrias arbitrarias. En esta sección se generaliza la idea de la cuadratura numérica presentada en los numerales anteriores para funciones de 1 sola variable independiente al caso de funciones de varias variables. En particular, acá ausmimos que las variables independientes representan coordenadas x, y de un espacio cartesiano.

En la figura 1.30 se esquematiza un dominio en 2-dimensiones (línea negra continua) denotado como R y sobre el cual se desea calcular una integral de

la forma

$$I = \iint_R f(x, y)\, \mathrm{d}A \ .$$

Para proceder con el cálculo el dominio se ha dividido en N subdominios rectangulares (líneas negras punteadas) de manera que un subdominio típico tiene dimensiones $\Delta x_i \times \Delta y_i$.

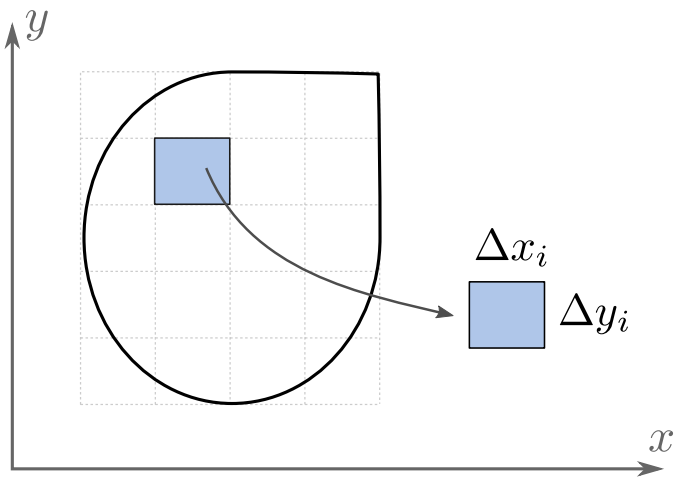


Figura 1.30. Partición de Riemman para un dominio plano.

Definiendo

$$p = \text{máx}\{\Delta x_i, \Delta y_i\}\,,$$

como la norma de la partición, se tiene, de acuerdo a la definición de una integral como una suma de Riemann, que:

$$I = \iint_R f(x, y)\, \mathrm{d}A \equiv \lim_{p \to 0} \sum_{j=1}^{N} f(x_j, y_j) \Delta x_j \Delta y_j \,.$$

Ahora, tomando cada uno de los límites en la dirección x y y por separado permite identificar 2 procesos de integración de tal forma que la integral sobre R se reduce a la integral doble dada por:

$$I = \iint f(x, y)\, \mathrm{d}x\, \mathrm{d}y \ .$$

Para identificar los límites de integración considere la figura 1.31. En esta se muestra un dominio de integración rectangular (en sombreado azul) cuyo lado mayor es paralelo a la dirección x y con altura media correspondiente a un valor constante y. Los lados menores del rectángulo tienen abscisas $x_1(y)$ y $x_2(y)$ respectivamente.

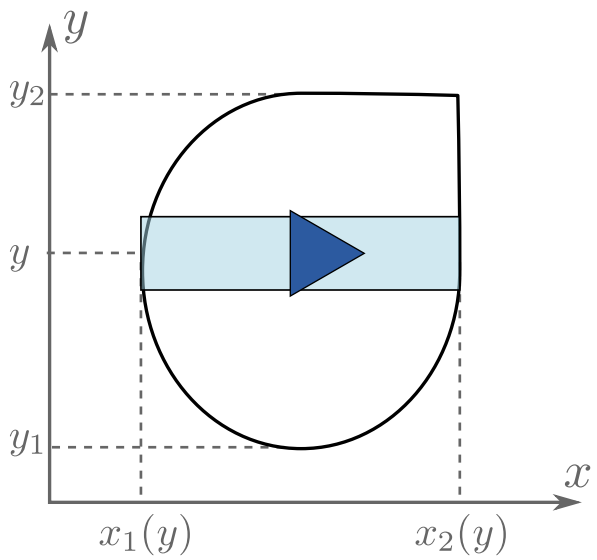


Figura 1.31. Integración en la dirección x.

Claramente se tiene que para el valor de y constante la contribución a la integral I sobre la región R del rectángulo acotado por $x_1(y)$ y $x_2(y)$ esta dada por:

$$\int_{x_1(y)}^{x_2(y)} f(x,y)\,\mathrm{d}x \ ,$$

de forma que el cálculo de la integral sobre la totalidad de la region R se completa repitiendo el proceso para valores de y constantes variando entre y_1 y y_2 y dando la integral total como:

$$I = \int_{y_1}^{y_2} \left\{ \int_{x_1(y)}^{x_2(y)} f(x,y)\,\mathrm{d}x \right\} \mathrm{d}y \tag{1.17}$$

Para dar mayor claridad a la ecuación (1.17), nótese que la integral interna puede escribirse como una función de y

$$F(y) = \int_{x_1(y)}^{x_2(y)} f(x,y)\,\mathrm{d}x \ ,$$

y la integral externa como

$$I = \int_{y_1}^{y_2} F(y)\,\mathrm{d}y \ .$$

Notese que el calculo de I se ha reducido a 2 integrales uni-dimensionales y en consecuencia el proceso puede resolverse por medio de las cuadraturas uni-dimensionales ya discutidas. En este caso la función mas interna $F(y)$ resulta de integrar solo la variación en x y posteriormente la integral I resulta de integrar la variación en y. Alternativamente (ver figura 1.32) también es posible definir

$$H(x) = \int_{y_1(x)}^{y_2(x)} f(x,y)\,\mathrm{d}y$$

de manera que la integral completa I queda definida por

$$I = \int_{x_1}^{x_2} H(x)\,\mathrm{d}x\,.$$

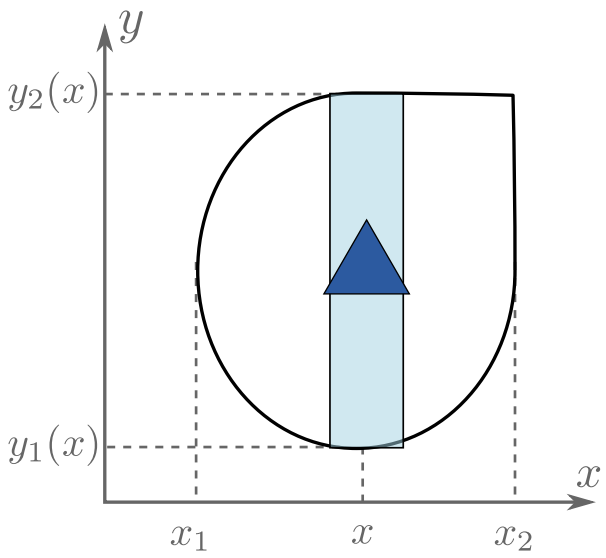


Figura 1.32. Integración en la dirección y.

Ejemplo

Considere la región (o dominio) triangular de la figura 1.33.

Utilice el concepto de integrales iteradas discutido anteriormente para determinar la integral de la función

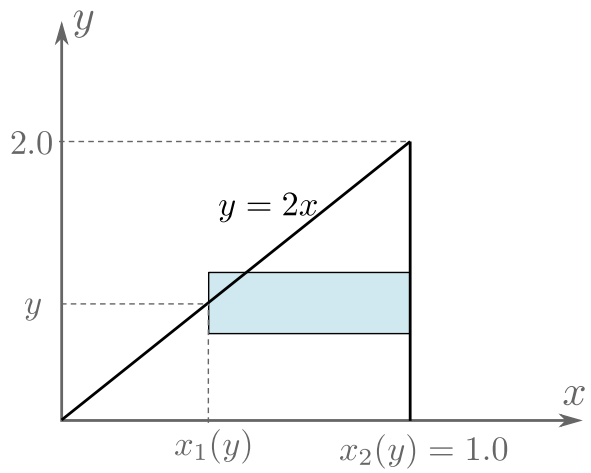


Figura 1.33. Integración en la dirección x sobre la región triangular R.

$$f(x, y) = xy^2$$

sobre la región R de la figura.

La integral sobre R esta dada por

$$I = \iint f(x, y)\,\mathrm{d}A$$

donde ddA representa un elemento diferencial de superficie. Supongamos que tomaremos rectangulos paralelos al eje x de manera que para valores constantes de y se tiene:

$$x_1(y) = \frac{y}{2}$$

y

$$x_2(y) = 1$$

y por lo tanto se tiene que:

$$F(y) = \int_{x_1(y)}^{1.0} xy^2 \, \mathrm{d}x \equiv \int_{y/2}^{1.0} xy^2 \, \mathrm{d}x \ ,$$

luego

$$F(y) = \left[\frac{1}{2} x^2 y^2 \right]_{y/2}^{1.0} \equiv \frac{1}{2} y^2 - \frac{1}{8} y^4 \, .$$

Identificando ahora los limites inferior y superior en la dirección y como $y_1 = 0$ y $y_2 = 2$ es posible integrar la dependencia en y de acuerdo con:

$$I = \int_0^{2.0} F(y) \, \mathrm{d}y \equiv \int_0^{2.0} \left(\frac{1}{2} y^2 - \frac{1}{8} y^4 \right) \mathrm{d}y \equiv \frac{8}{15} \, .$$

Procediendo de manera alternativa (ver figura 1.34) es posible escribir:

$$H(x) = \int_{y_1(x)}^{y_2(x)} xy^2 \, \mathrm{d}y$$

y

$$I = \int_{x_1}^{x_2} H(x) \, \mathrm{d}x \ .$$

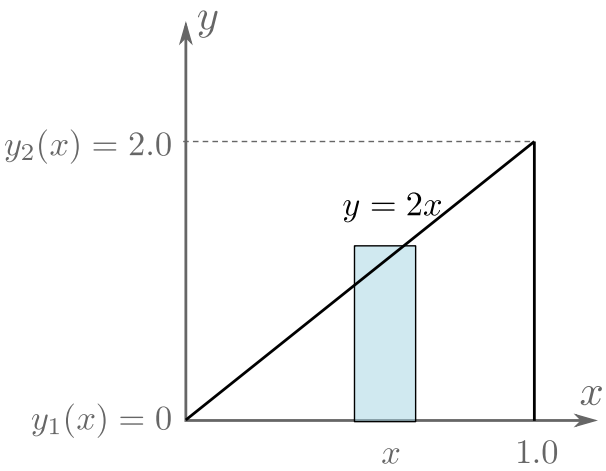


Figura 1.34. Integración en la dirección y sobre la región triangular R.

donde:

$$y_1(x) = 0$$
$$y_2(x) = 2x$$

luego

$$H(x) = \int_0^{2x} xy^2 dy \equiv \frac{1}{3}xy^3\Big|_0^{2x} \equiv \frac{8}{3}x^4$$

de manera que la integral se reduce a:

$$I = \int_0^{1.0} \frac{8}{3}x^4 dx \equiv \frac{8}{15}.$$

En el apendice se presenta una rutina con la correspondiente implementación en Python del esquema de integrales iteradas. Esta permite calcular integrales sobre regiones rectangulares. Por ejemplo si se calcula la siguiente integral

$$\int_0^2 \int_0^2 [3xy^2 - x^3]\,\mathrm{d}x\,\mathrm{d}y = 8\,,$$

usando la rutina código se obtiene el siguiente resultado:

```
Gauss quadrature: 8.000000
```

Ejercicios

1. Calcule la integral de la función

$$f(x, y) = 4x^2 + 3xy + y^2$$

sobre los dominios planos mostrados en las figuras

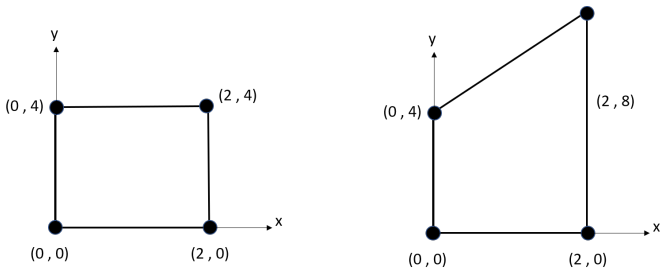


Figura 1.35. Dominios de integración para el problema 1.

2. Calcule la integral de la función

$$f(r, s) = rs$$

sobre el dominio triangular mostrado en la figura

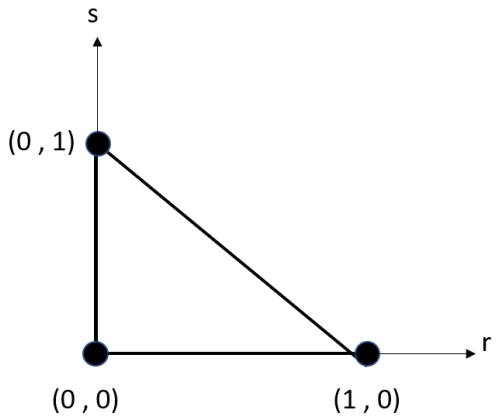


Figura 1.36. Dominios de integración para el problema 2.

3. Calcular las siguientes integrales usando una cuadratura de Gauss apropiada:

$$I = \int_{1.0}^{1.5} x^2 \ln x \, \mathrm{d}\, x$$

$$I = \int_{0}^{\frac{\pi}{4}} e^{3x} Sin2x \, \mathrm{d}\, x$$

$$I = \int_{0}^{\frac{\pi}{4}} Cos^2 x \, \mathrm{d}\, x$$

Apéndice A

Algoritmos

A.1. Ejemplos

A.1.1. Ordenamiento de una lista de valores en orden ascendente-Algoritmo de la burbuja

Supongamos que se tiene el vector

$$\begin{bmatrix} A_1 & A_2 & A_3 & \cdots & A_N \end{bmatrix}$$

de orden N y se desea organizarlo en orden ascendente. En el denominado algoritmo de la burbuja[1] se hacen recorridos del vector en los cuales se comparan pares de valores consecutivos y se hacen las permutaciones que sean necesarias de acuerdo con el resultado de la comparación. El ordenamiento termina cuando se haga una pasada por el vector sin la necesidad de realizar

[1] Ver https://en.wikipedia.org/wiki/Bubble_sort.

ninguna permutación.

Algoritmo 4: Algoritmo de la burbuja

```
Data: Vector A de orden N
Result: Vector original con elementos en orden ascendente
READ N;
READ A_i;
repeat
    i ← 1 ;
    NSW ← 0 ;
    while i < N do
        if A_i > A_{i+1} then
            SWAP(A_i, A_{i+1});
            NSW ← NSW + 1;
        end
        i ← i + 1;
    end
until NSW = 0;
```

Nótese que en el algoritmo de la burbuja hemos usado la función SWAP(A,B) la cual se encarga de realizar el intercambio de los valores almacenados en las variables A y B. La función SWAP se define en el siguiente algoritmo.

Algoritmo 5: Función SWAP(A,B)

```
Data: Numbers to swap A, B
Result: A and B with changed values
temp ← A;
A ← B;
B ← temp;
```

A.1.2. Ordenamiento de una lista de valores en orden ascendente-Algoritmo de selección

En este algoritmo se recorre el vector original N veces y en cada pasada se extrae el valor mínimo y la posición del valor mínimo. Esto se ilustra en la siguiente estructura matricial en la cual la primera fila corresponde al vector original (con los valores en desorden). El valor mínimo en cada fila se muestra resaltado en negrilla. En cualquier caso la fila $i+1$ corresponde a la fila i tras haber extraído el valor mínimo.

$$\begin{bmatrix} \mathbf{1} & 5 & 3 & 2 & 7 & 4 \\ 5 & 3 & \mathbf{2} & 7 & 4 & 0 \\ 5 & \mathbf{3} & 7 & 4 & 0 & 0 \\ 5 & 7 & \mathbf{4} & 0 & 0 & 0 \\ \mathbf{5} & 7 & 0 & 0 & 0 & 0 \\ 7 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Posteriormente el valor mínimo se acomoda en el vector solución. En la siguiente secuencia se muestra la forma del vector solución tras cada iteración.

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$
$$\begin{bmatrix} 1 & 2 & 0 & 0 & 0 & 0 \end{bmatrix}$$
$$\begin{bmatrix} 1 & 2 & 3 & 0 & 0 & 0 \end{bmatrix}$$
$$\begin{bmatrix} 1 & 2 & 3 & 4 & 0 & 0 \end{bmatrix}$$
$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 0 \end{bmatrix}$$
$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 7 \end{bmatrix}$$

A continuación se presenta el algoritmo de inserción. En este haremos uso del concepto de función. En programación una función no es otra cosa que un programa, que procesa datos de entrada para producir resultados o datos de salida, pero que esperamos usar de manera repetida en varios programas diferentes. En la definición (o declaración) de una función es importante distinguir cuales son los datos de entrada y cuales son los datos de salida.

Para esto adoptaremos la siguiente convención

Algoritmo 6: function_name($Par-1$,..,$Par-n$;$Res-1$,...,$Res-k$)

Data: n input parameters
Result: k output results
Instr-1;
Instr-2;
...;
Instr-n;

En esta `function_name` corresponde al nombre de la función y entre paréntesis se declaran o definen los parámetros o datos de entrada y los resultados o datos de salida. En la definición de la función estos parámetros se separan por un punto y coma (;).

En el algoritmo de selección definiremos 2 funciones. La primera llamada MINOR se encargará de extraer el menor valor de un vector y su posición, mientras que la segunda se encarga de copiar la fila i en la fila $i+1$ tras haber extraído el valor mínimo. La definición de estas funciones se deja como tarea. El algoritmo de inserción tiene entonces la siguiente forma

Algoritmo 7: Selection algorithm

for $j \leftarrow i$ **to** N **do**
    MINOR(A_{ij}; min, $ipos$);
    $R_i \leftarrow min$;
    EXTRACT(A_{ij}, $ipos$; A_{i+1j});
end

A.1.3. Organización de un vector fila en una matriz

Suponga que se tiene un total de $N \times M$ datos correspondientes a la aceleración del terreno y registrada mediante acelerógrafos localizados en N sitios diferentes de la ciudad. Cada registro contiene un total de M datos de aceleración para M datos de tiempo. Sin embargo la totalidad de los datos se

encuentra almacenada en un arreglo unidimensional (o vector) de orden $N \times M$ donde cada grupo de N datos corresponde a las aceleraciones en un mismo instante de tiempo para las N estaciones. Con el fin de procesar el registro para cada estación se requiere extraer la información y almacenarla en una matriz de orden $N \times M$ en la cual cada fila corresponde a las aceleraciones registradas en una misma zona de la ciudad.

El siguiente algoritmo presenta un algoritmo para almacenar el vector en una matriz. En este caso, se almacena una fila de la matriz en cada iteración.

Algoritmo 8: Escritura de un vector en una matriz. Opción 1

Data: Vector V of order $N \times M$
Result: Matrix of order $[N, M]$
READ N;
READ M;
READ V;
for $i \leftarrow 1$ *to* N **do**
  $Mat\,[i,:] \leftarrow V\,[1 + N\,(i-1)\,, Ni]$;
end

El siguiente algoritmo también presenta un algoritmo para almacenar el vector en una matriz. En este caso, se almacena una posición de la matriz en cada iteración. Note los dos ciclos anidados, mientras en el caso anterior se usó un único ciclo.

Algoritmo 9: Escritura de un vector en una matriz. Opción 2

Data: Vector V of order NxM
Result: Matrix of order $[N, M]$
READ N;
READ M;
READ V;
for $i \leftarrow 1$ *to* N **do**
  for $j \leftarrow 1$ *to* M **do**
    $Mat\,[i,j] \leftarrow V\,[i + N\,(j-1)]$;
  end
end

A.1.4. Numeros primos

En esta sección presentamos algunos algoritmos relacionados con números primos. El primer algoritmo verifica si un número entero dado es primo o no y devuelve un mensaje con esta información.

Algoritmo 10: Prime number verification

Data: number N
Result: Prime or no-prime number
READ N;
$iflag \leftarrow 0$;
for $i \leftarrow 2$ **to** $N - 1$ **do**
  if n mód $i = 0$ **then**
    $iflag \leftarrow 1$;
  end
end
if $iflag = 0$ **then**
  WRITE "Prime number";
else
  WRITE "No-prime number";
end

Una vez contemos con un algoritmo para verificar si un número es primo o no podemo realizar la descomposición en factores primos para cualquier número entero mayor a 2. El siguiente algoritmo presenta cómo hacerlo.

Algoritmo 11: Prime factors of a number

```
Data: number N
Result: Prime factors of a number N
READ N;
for i ← 2 to N do
    res ← n  mód i ;
    if res = 0 then
        Detect if i is prime (iflag = 0);
        if iflag = 0 then
            WRITE "Is prime";
        end
    end
end
```

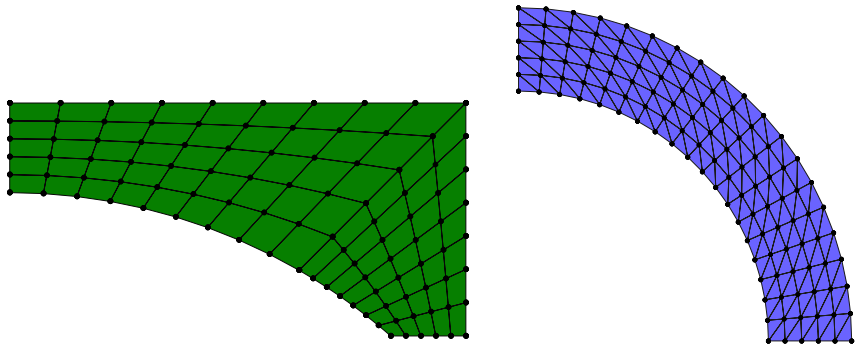
El siguiente algoritmo presenta el pseudocódigo para encontrar la suma de todos los primos menores que cierto número N dado.

Algoritmo 12: Sum of the prime numbers up to a given value

```
Data: number N
Result: Sum of all the primes up to N
READ N;
sum ← 2 ;
for i ← 3 to N do
    Detect if i is prime (iflag = 0);
    if iflag = 0 then
        sum ← sum + i;
    end
end
```

A.1.5. Mallador de dominios rectangulares

En una gran cantidad de métodos numéricos como diferencias finitas, elementos finitos, elementos de frontera se requiere de la partición o división del dominio del problema en sub-dominios o elementos. Algunos ejemplos se muestran en la figura A.1.



(a) **(b)**

Figura A.1. Mallas para el análisis de un puente y una tubería por medio del método de los elementos finitos. En el primer caso el dominio del problema esta dividido en subdominios de 4 lados mientras que la tubería se dividió en elementos triangulares.

Este tipo de particiones se denominan mallas. Generalmente una malla corresponde a una serie de sub-dominios o elementos definidos por un determinado numero de puntos (o nudos) y por los puntos mismos. Por ejemplo, las mallas de los problemas ilustrados en la figura A.1 están conformadas por elementos de 4 y de 3 lados que resultan de unir 4 y 3 puntos respectivamente.

En los programas de Mecánica Computacional un elemento se define mediante la declaración de un número entero, correspondiente al identificador del elemento, y el listado de puntos o nodos que conforman el elemento.

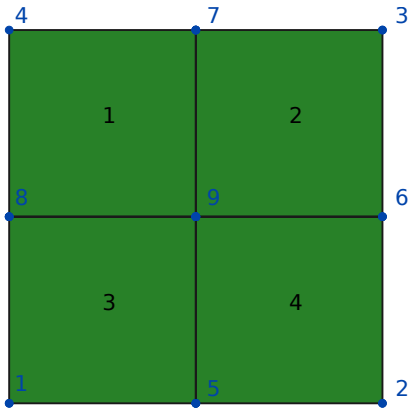


Figura A.2. Malla de un dominio cuadrado dividido en 4 subdominios cuadrados.

Por ejemplo, considere el elemento 1 de la malla mostrada en la figura A.2. Este se define como se ilustra en la figura A.3. En esta, el primer entero corresponde al identificador del elemento, mientras que los 4 enteros subsiguientes corresponden a los identificadores de los 4 nudos que conforman el elemento.

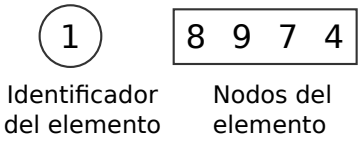


Figura A.3. Definición de un elemento cuadrado correspondiente al identificador del elemento y 4 nudos.

En el proceso de creación de una malla es necesario definir el orden que deben seguir los puntos que conforman un elemento. Dicha definición se hace en términos de un elemento típico o generalizado. Por ejemplo, el ordenamiento de los nudos del elemento 1 corresponde a un ordenamiento de un elemento típico como el que se muestra en la figura A.4. Nótese que este elemento se encuentra definido por sus 4 nodos y el sentido en el que estos están relacionados.

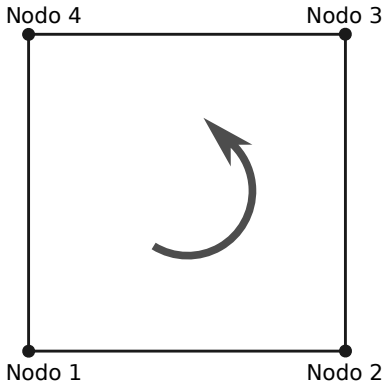


Figura A.4. Definición de un elemento típico en el que se declara el ordenamiento de los nodos.

A.1.6. Ejemplo: Algoritmo para mallado de dominios rectangulares

Considere el dominio rectangular de ancho W y altura H dividido en 18 subdominios cada uno de ellos correspondiente a un elemento de 4 lados definidos por los 4 nudos que se muestran en **??**. Definiendo como Δx y Δy los tamaños característicos en las direcciones X y Y respectivamente diseñar un algoritmo que genere la malla para un dominio rectangular como el mostrado en la figura.

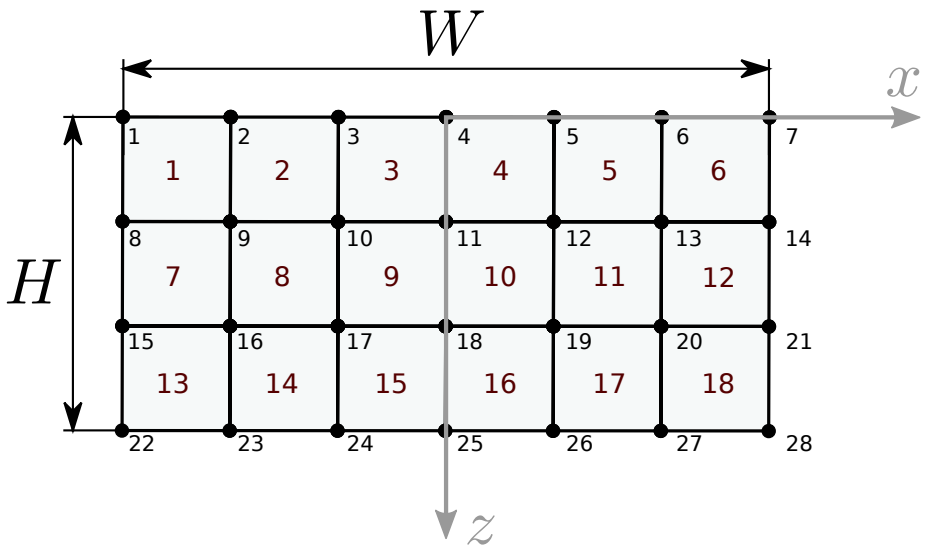


Figura A.5. Dominio rectangular.

El archivo de datos correspondiente al caso particular del dominio mostrado en la figura A.5 se muestra a continuación.

archivo_puntos.txt

```
1  -3.0   0.0
2  -2.0   0.0
3  -1.0   0.0
4   0.0   0.0
5   1.0   0.0
6   2.0   0.0
7   3.0   0.0
8  -3.0   1.0
9  -2.0   1.0
10 -1.0   1.0
11  0.0   1.0
12  1.0   1.0
13  2.0   1.0
14  3.0   1.0
15 -3.0   2.0
16 -2.0   2.0
17 -1.0   2.0
18  0.0   2.0
19  1.0   2.0
20  2.0   2.0
21  3.0   2.0
22 -3.0   3.0
23 -2.0   3.0
24 -1.0   3.0
25  0.0   3.0
26  1.0   3.0
27  2.0   3.0
28  3.0   3.0
```

archivo_elementos.txt

```
1    1   2   9   8
2    2   3  10   9
3    3   4  11  10
4    4   5  12  11
5    5   6  13  12
6    6   7  14  13
7    8   9  16  15
8    9  10  17  16
9   10  11  18  17
10  11  12  19  18
11  12  13  20  19
12  13  14  21  20
13  15  16  23  22
14  16  17  24  23
15  17  18  25  24
16  18  19  26  25
17  19  20  27  26
18  20  21  28  27
```

Algoritmo 13: Mesher for simple rectangular domains

Data: Domain dimensions W and H and characteristic element sides Δx and Δy along the x and y directions; Horizontal coordinate of the left-most point x_L

Result: Text files containing the nodal point coordinates and element connectivities

READ W, H, Δx, Δy, x_L ;
$Mx \leftarrow W/\Delta x$ (number of elements in x);
$My \leftarrow H/\Delta y$ (number of elements in y);
$Nx \leftarrow Mx + 1$ (number of points in x);
$Ny \leftarrow My + 1$ (number of points in y);
$l \leftarrow 0$;
for $j \leftarrow 1$ *to* Ny **do**
    for $i \leftarrow 1$ *to* Nx **do**
        $l \leftarrow l + 1$;
        $x \leftarrow x_L + (i-1) \cdot \Delta x$;
        $y \leftarrow (j-1) \cdot \Delta y$;
        WRITE to point file: l, x, y
    end
end
$l \leftarrow 0$ (for elements);
$q \leftarrow 0$ (for node-1);
$k \leftarrow 0$ (for node-3);
for $i \leftarrow 1$ *to* My **do**
    $q \leftarrow q + 1$;
    $k \leftarrow q + Nx + 1$;
    for $j \leftarrow 1$ *to* Mx **do**
        $l \leftarrow l + 1$;
        WRITE to element file: l, q, $q+1$, k, $k-1$;
        $q \leftarrow q + 1$;
        $k \leftarrow k + 1$;
    end
end

Apéndice B

Codigos

B.0.1. Detección de las raices de una función en un intervalo dado

```
from numpy import zeros

def bracketing(fun, a, b, N):
    msg = "Maximum number of iterations reached."
    dx = (b - a)/(N - 1)
    iroot = 0
    x2 = a
    xR = zeros(N, float)
    for i in range(0, N):
        x1 = x2
        x2 = x1 + dx
        if (fun(x1) * fun(x2)) < 0:
            msg = "A change of sign was found."
            iroot = iroot + 1
            xR[i] = x1
    return xR, msg

# Function call
a = -10.0
b = 10.0
N = 21
fun = lambda x: x**3 + 4*x**2 - 10
xR, msg = bracketing(fun, a, b, N)
print(msg)
print(xR)
```

B.0.2. Método de bisección para la localización de raices en un intervalo dado

```
def bisection(fun, a, b, xtol=1e-6, ftol=1e-12, verbose=False):
    """
    Use bisection method to estimate the root of a real function
    """
    if fun(a) * fun(b) > 0:
        c = None
        msg = "The function should change sign in the interval."
    else:
        nmax = int(ceil(log2((b - a)/xtol)))
        for cont in range(nmax):
            c = 0.5*(a + b)
            if verbose:
                print("n: {}, x: {}".format(cont, c))
            if abs(fun(c)) < ftol:
                msg = "Root found with desired accuracy."
                break
            elif fun(a) * fun(c) < 0:
                b = c
            elif fun(b) * fun(c) < 0:
                a = c
            msg = "Maximum number of iterations reached."
    return c, msg

x, msg = bisection(lambda x: x**3 + 4*x**2 -10, -2, 2, xtol=1e-4,
                   verbose=True)
print(msg)
print(x)
```

Código 2. Método de bisección en Python.

B.0.3. Método de Newton-Raphson para la localización de raices en un intervalo dado

```
def newton(fun, grad, x, niter=50, ftol=1e-8, verbose=False):
    """
    Use Newton method to estimate the root of a real function
    """
    msg = "Maximum number of iterations reached."
    for cont in range(niter):
        if abs(grad(x)) < ftol:
            x = None
            msg = "Derivative near to zero."
            break
        if verbose:
            print("n: {}, x: {}".format(cont, x))
        x = x - fun(x)/grad(x)
        if abs(fun(x)) < ftol:
            msg = "Root found with desired accuracy."
            break
    return x, msg

func = lambda x:x**3 + 4*x**2 - 10
deriv = lambda x: 3*x**2 + 8*x
result = newton(func, deriv, 2, verbose=True)
print(result)
```

Código 3. Método de Newton-Rapshon en Python.

B.0.4. Integración numérica: Regla del trapecio

```
import numpy as np
from sympy import symbols, integrate
```

```
def trapz(fun, x0, x1, n):
"""Trapezoidal rule for integration

Parameters
----------
fun : callable
     Function to integrate.
x0 : float
     Initial point for the integration interval.
x1 : float
     End point for the integration interval.
n : int
     Number of points to take in the interval.

Returns
-------
inte : float
     Approximation of the integral

"""
x = np.linspace(x0, x1, n)
y = fun(x)
dx = x[1] - x[0]
inte = 0.5*dx*(y[0] + y[-1])
for cont in range(1, n - 1):
inte = inte + dx*y[cont]
return inte


fun = lambda x: x**3 + 4*x**2 - 10
for cont in range(2, 11):
    numeric_int = trapz(fun, -1, 1, cont)
    print("Approximation for {} subdivisions: {:.6f}".format(cont - 1,
           numeric_int))

x  = symbols('x')
analytic_int = integrate(fun(x) , (x , -1 , 1))
```

```
print("Analytic integral: {:.6f}".format(float(analytic_int)))
```

B.0.5. Integración numériva: Cuadratura Gaussiana

```
import numpy as np
from scipy.special import roots_legendre # Gauss points and weights
from sympy import symbols, integrate


def gauss1d(fun, x0, x1, n):
"""Gauss quadrature in 1D

Parameters
----------
fun : callable
    Function to integrate.
x0 : float
    Initial point for the integration interval.
x1 : float
    End point for the integration interval.
n : int
    Number of points to take in the interval.

Returns
-------
inte : float
     Approximation of the integral

"""
xi, wi = roots_legendre(n)
inte = 0
h = 0.5 * (x1 - x0)
xm = 0.5 * (x0 + x1)
for cont in range(n):
```

```
    inte = inte + h * fun(h * xi[cont] + xm) * wi[cont]
return inte

fun = lambda x: x**3 + 4*x**2 - 10
gauss_inte = gauss1d(fun, -1, 1, 4)
x  = symbols('x')
analytic_inte = integrate(fun(x) , (x , -1 , 1))
print("Analytic integral: {:.6f}".format(float(analytic_inte)))
print("Gauss quadrature: {:.6f}".format(gauss_inte))
```

B.0.6. Integración en un 2 dimensiones

```
 numpy as np
 ipy.special import roots_legendre # Gauss points and weights


 ss2d(fun, x0, x1, y0, y1, nx, ny):
 s quadrature for a rectangle in 2D

 ers
 ---
 allable
 n to integrate.
 oat
  point for the integration interval in x.
 oat
 nt for the integration interval in x.
 oat
  point for the integration interval in y.
 oat
 nt for the integration interval in y.
 t
 of points to take in the interval in x.
 t
 of points to take in the interval in y.
```

```
float
mation of the integral


= roots_legendre(nx)
= roots_legendre(ny)
0
5 * (x1 - x0)
5 * (y1 - y0)
5 * (x0 + x1)
5 * (y0 + y1)
t_x in range(nx):
t_y in range(ny):
(hx * xi[cont_x] + xm, hy * yj[cont_y] + ym)
inte + hx* hy * f * wi[cont_x] * wj[cont_y]
inte


ambda x, y: 3*x*y**2 - x**3
nte = gauss2d(fun, 0, 2, 0, 2, 2, 2)
Gauss quadrature: {:.6f}".format(gauss_inte))
```

Apéndice C

SymPy

SymPy es una biblioteca de Python que permite realizar cálculos simbólicos. Nos ofrece las capacidades de álgebra computacional, y se puede usar en línea a través de SymPy Live o SymPy Gamma, este último es similar a Wolfram Alpha.

Si usas Anaconda este paquete ya viene instalado por defecto pero si se usa miniconda o pip debe instalarse.

```
conda install sympy # Usando el gestor conda de
                    # Anaconda/Miniconda
pip install sympy # Usando el gestor pip (puede requerir
                  # instalar más paquetes)
```

Lo primero que debemos hacer, antes de usarlo, es importar el módulo, como con cualquier otra biblioteca de Python.

Si deseamos usar SymPy de forma interactiva usamos

```
from sympy import *
init_printing()
```

Para scripting es mejor importar la biblioteca de la siguiente manera

```
import sympy as sym
```

Y llamar las funciones de la siguiente manera

```
x = sym.Symbols("x")
expr = sym.cos(x)**\DecValTok{2} + \DecValTok{3}*x
deriv = expr.diff(x)
```

en donde calculamos la derivada de $\cos^2(x)+3x$, que debe ser $-2\sin(x)\cos(x)+3$.

```
%matplotlib notebook
import numpy as np
import matplotlib.pyplot as plt
from sympy import *
```

```
init_printing()
```

Definamos la variable x como un símbolo matemático. Esto nos permite hacer uso de esta variable en SymPy.

```
x = symbols("x")
```

Empecemos con cálculos simples. Abajo, tenemos una *celda de código* con una suma. Ubica el cursor en ella y presiona SHIFT + ENTER para evaluarla.

```
1 + 3
```

4

Realicemos algunos cálculos.

```
factorial(5)
```

120

```
1 // 3
```

0

```
1 / 3
```

0.3333333333333333

```
S(1) / 3
```

$$\frac{1}{3}$$

Podemos evaluar esta expresión a su versión en punto flotante

```
sqrt(2*pi)
```

$$\sqrt{2}\sqrt{\pi}$$

```
float(sqrt(2*pi))
```

2.5066282746310007

También podemos almacenar expresiones como variables, como cualquier variable de Python.

```
radius = 10
height = 100
area = pi * radius**2
volume = area * height
```

```
volume
```

$$10000\pi$$

```
float(volume)
```

$$31415.926535897932$$

Hasta ahora, hemos usado SymPy como una calculadora. Intentemos algunos cálculos más avanzados. Por ejemplo, algunas integrales.

```
integrate(sin(x), x)
```

$$-\cos(x)$$

```
integrate(sin(x), (x, 0, pi))
```

2

Podemos definir una función, e integrarla

```
f = lambda x: x**2 + 5
```

```
f(5)
```

30

```
integrate(f(x), x)
```

$$\frac{x^3}{3} + 5x$$

```
y = symbols("y")
integrate(1/(x**2 + y), x)
```

$$-\frac{\sqrt{-\frac{1}{y}}}{2}\log\left(x - y\sqrt{-\frac{1}{y}}\right) + \frac{\sqrt{-\frac{1}{y}}}{2}\log\left(x + y\sqrt{-\frac{1}{y}}\right)$$

Si asumimos que el denominador es positivo, esta expresión se puede simplificar aún más

```
a = symbols("a", positive=True)
integrate(1/(x**2 + a), x)
```

$$\frac{1}{\sqrt{a}}\operatorname{atan}\left(\frac{x}{\sqrt{a}}\right)$$

Hasta ahora, aprendimos lo más básico. Intentemos algunos ejemplos más complicados ahora.

Nota: Si quieres saber más sobre una función específica se puede usar la función `help()` o el comándo *mágico* de IPython `??`

```
help(integrate)
```

```
  Help on function integrate in module sympy.integrals.integrals:

integrate(*args, **kwargs)
    integrate(f, var, ...)

    .
    .
    .
```

```
>>> integrate(x**a*exp(-x), (x, 0, oo), conds='separate')
(gamma(a + 1), -re(a) < 1)

See Also
========

Integral, Integral.doit
```

El bloque de salida anterior fue resumido (indicación de los puntos verticales).

```
integrate??
```

C.1. Ejemplos

C.1.1. Solución de ecuaciones algebraicas

Para resolver sistemas de ecuaciones algebraicos podemos usar: `solveset` and `solve` <http://docs.sympy.org/latest/tutorial/solvers.html>__. El método preferido es solveset, sin embargo, hay sistemas que se pueden resolver usando solve‘ y no `solveset`.

Para resolver sistemas usando `solveset`:

```
a, b, c = symbols("a b c")
solveset(a*x**2 + b*x + c, x)
```

$$\left\{-\frac{b}{2a} - \frac{1}{2a}\sqrt{-4ac+b^2}, -\frac{b}{2a} + \frac{1}{2a}\sqrt{-4ac+b^2}\right\}$$

Debemos ingresar la expresión igualada a 0, o como una ecuación

```
solveset(Eq(a*x**2 + b*x, -c), x)
```

$$\left\{-\frac{b}{2a} - \frac{1}{2a}\sqrt{-4ac+b^2}, -\frac{b}{2a} + \frac{1}{2a}\sqrt{-4ac+b^2}\right\}$$

`solveset` no permite resolver sistemas de ecuaciones no lineales, por ejemplo

```
solve([x*y - 1, x - 2], x, y)
```

$$\left[\left(2, \quad \frac{1}{2}\right)\right]$$

C.1.2. Álgebra lineal

Usamos `Matrix` para crear matrices. Las matrices pueden contener variables y expresiones matemáticas.

Usamos el método `.inv()` para calcular la inversa, y `*` para multiplicar matrices.

```
A = Matrix([
        [1, -1],
        [1, sin(c)]
    ])
display(A)
```

$$\begin{bmatrix} 1 & -1 \\ 1 & \sin(c) \end{bmatrix}$$

```
B = A.inv()
display(B)
```

$$\begin{bmatrix} \frac{\sin(c)}{\sin(c)+1} & \frac{1}{\sin(c)+1} \\ -\frac{1}{\sin(c)+1} & \frac{1}{\sin(c)+1} \end{bmatrix}$$

```
A * B
```

$$\begin{bmatrix} \frac{\sin(c)}{\sin(c)+1} + \frac{1}{\sin(c)+1} & 0 \\ 0 & \frac{\sin(c)}{\sin(c)+1} + \frac{1}{\sin(c)+1} \end{bmatrix}$$

Esta expresión debería ser la matriz identidad, simplifiquemos la expresión. Existen varias formas de simplificar expresiones, y `simplify` es la más general.

```
simplify(A * B)
```

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

C.1.3. Graficación

SymPy permite realizar gráficos 2D y 3D

```
from sympy.plotting import plot3d
```

```
plot(sin(x), (x, -pi, pi));
```

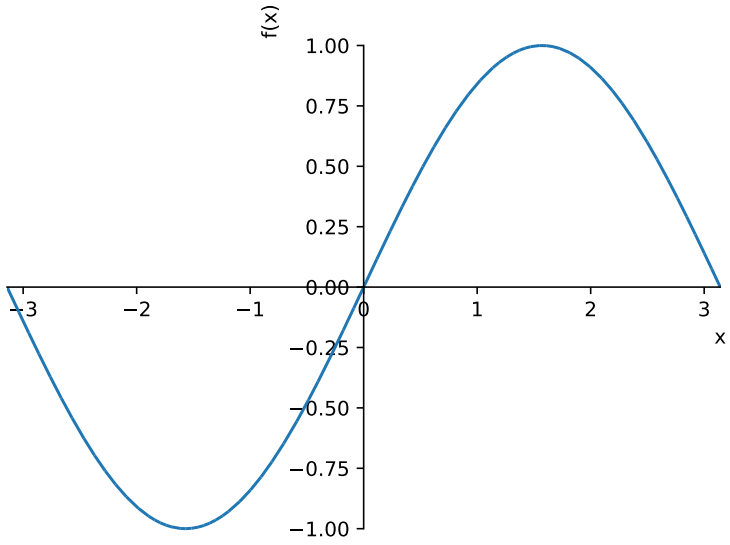


Figura C.1. Ejemplo de gráfico con SymPy.

```
monkey_saddle = x**3 - 3*x*y**2
p = plot3d(monkey_saddle, (x, -2, 2), (y, -2, 2))
```

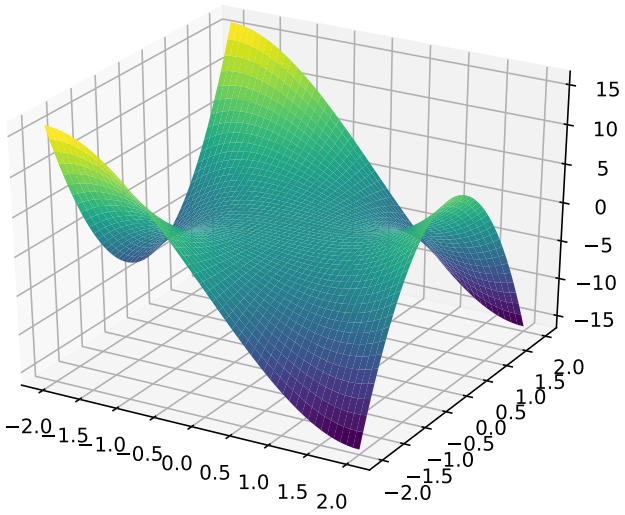


Figura C.2. Ejemplo de gráfico 3D con sympy.

C.1.4. Derivadas y ecuaciones diferenciales

Podemos usar la función `diff` o el método `.diff()` para calcular derivadas.

```
f = lambda x: x**2
```

```
diff(f(x), x)
```

$2x$

```
f(x).diff(x)
```

$$2x$$

```
g = lambda x: sin(x)
```

```
diff(g(f(x)), x)
```

$$2x\cos\left(x^2\right)$$

Y sí, ¡SymPy sabe sobre la regla de la cadena!

Para terminar, resolvamos una ecuación diferencial de segundo orden

$$u''(t) + \omega^2 u(t) = 0$$

```
t = symbols("t")
u = symbols("u", cls=Function)
omega = symbols("omega", positive=True)
```

```
ode = u(t).diff(t, 2) + omega**2 * u(t)
dsolve(ode, u(t))
```

$$u(t) = C_1 \sin(\omega t) + C_2 \cos(\omega t)$$

C.2. Convertir expresiones de SymPy en funciones de NumPy

`lambdify` permite convertir expresiones de sympy en funciones para hacer cálculos usando NumPy.

Veamos cómo.

```
f = lambdify(x, x**2, "numpy")
f(3)
```

9

```
f(np.array([1, 2, 3]))
```

array([1, 4, 9])

Intentemos un ejemplo más complejo

```
fun = diff(sin(x)*cos(x**3) - sin(x)/x, x)
fun
```

$$-3x^2 \sin(x) \sin\left(x^3\right) + \cos(x) \cos\left(x^3\right) - \frac{1}{x} \cos(x) + \frac{1}{x^2} \sin(x)$$

```
fun_numpy = lambdify(x, fun, "numpy")
```

y evalúemoslo en algún intervalo, por ejemplo, $[0, 5]$.

```
pts = np.linspace(0, 5, 1000)
fun_pts = fun_numpy(pts + 1e-6) # Para evitar división por 0
```

```
plt.figure()
plt.plot(pts, fun_pts)
```

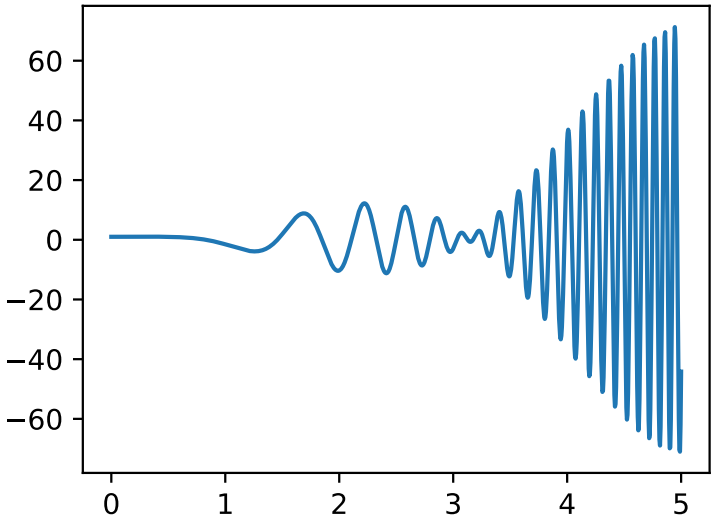


Figura C.3. Gráfico de la función evaluada luego de usar lambdify.

C.3. Ejercicios

1. Calcule el límite

$$\lim_{x \to 0} \frac{\sin(x)}{x} \, .$$

2. Resuelva la ecuación diferencial de Bernoulli

$$x \frac{\mathrm{d}u(x)}{\mathrm{d}x} + u(x) - u(x)^2 = 0 \, .$$

C.4. Recursos adicionales

- Equipo de desarrollo de SymPy. SymPy Tutorial, (2018). Consultado: Julio 23, 2018
- Ivan Savov. Taming math and physics using SymPy, (2017). Consultado: Julio 23, 2018