

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

2020

Tabla de contenidos

1. Modelación Computacional	1
1.1. Motivación	2
1.2. Ejemplos	4
1.2.1. Caída libre de un paracaidista	4
1.2.2. Vibración en una cuerda	9
2. Métodos Numéricos	13
2.1. Raíces de Ecuaciones	15
2.1.1. Planteamiento del problema	15
2.1.2. Localización incremental de raíces	18
2.1.3. Método de bisección	21
2.1.4. Método de Newton-Raphson	24
2.2. Interpolación	29
2.2.1. Planteamiento del problema	29

TABLA DE CONTENIDOS	II
2.2.2. Teorema de interpolación de Lagrange	33
2.2.3. Distribución de los puntos de muestreo	40
2.2.4. Interpolación local usando una función a tramos	44
2.2.5. Generalización a dominios bi-dimensionales	46
2.3. Integración numérica	51
2.3.1. Planteamiento del problema	52
2.3.2. Integración numérica mediante polinomios de interpolación	55
2.3.3. Regla del trapecio	56
2.3.4. Cuadraturas Gaussianas	58
2.3.5. Transformación del dominio de solución	64
2.3.6. Extensión a dominios en 2D	68
3. Discretización de dominios	78
3.1. Uso de mallas	80
3.2. Concepto de malla	82
3.2.1. Mallas estructuradas	83
3.2.2. Mallas no-estructuradas	84
3.2.3. Representación de la malla	85
3.3. Creación de mallas	87

TABLA DE CONTENIDOS	III
3.4. Ejemplo de creación de geometría y malla con Gmsh	89
3.4.1. Creación de la geometría	90
3.4.2. Parámetros de malla en el archivo de entrada	93
3.4.3. Archivo de entrada completo	94
3.5. Lectura de mallas de Gmsh desde Python	97
4. Introducción a los Elementos Finitos	101
4.1. Introducción	102
4.1.1. Aplicaciones de los elementos finitos	104
4.2. Concepto intuitivo de rigidez	104
4.2.1. ¿Cómo describir el sólido?	106
4.2.2. Elementos finitos (aproximando el continuo)	107
4.3. Relación fuerza-desplazamiento para todo el sistema	108
4.3.1. Sistema partícula-resorte	109
4.4. Algoritmo de solución	111
4.4.1. Equilibrio de los resortes	112
4.4.2. Equilibrio de una partícula	113
4.4.3. Ensamblaje	113
4.5. Programa de análisis estructural	115
4.5.1. Programa principal	120

TABLA DE CONTENIDOS	IV
4.6. Aspectos a tener en cuenta	120
4.6.1. Unidades	120
4.6.2. Materiales	121
4.7. Geometría	122
4.7.1. Simplificación de la geometría	122
A. Códigos	124
A.1. Búsqueda de raíces	125
A.1.1. Detección de las raíces de una función en un intervalo dado	125
A.1.2. Método de bisección para la localización de raíces en un intervalo dado	126
A.1.3. Método de Newton-Raphson para la localización de raíces en un intervalo dado	127
A.2. Integración numérica	127
A.2.1. Regla del trapecio	127
A.2.2. Cuadratura Gaussiana	129
A.2.3. Integración en 2 dimensiones	130
B. SymPy	132
B.1. Ejemplos	139
B.1.1. Solución de ecuaciones algebraicas	139

TABLA DE CONTENIDOS

v

B.1.2. Álgebra lineal	140
B.1.3. Graficación	142
B.1.4. Derivadas y ecuaciones diferenciales	144
B.2. Convertir expresiones de SymPy en funciones de NumPy . . .	146
B.3. Ejercicios	148
B.4. Recursos adicionales	149
Bibliografía	150

Capítulo 1

Modelación Computacional

La modelación matemática y la simulación son herramientas matemáticas fundamentales en ciencia e ingeniería, y es válido afirmar que todo el mundo las usa (incluso aquellos que no se percatan de ello). La pregunta no es si usar estas herramienta o no, sino cómo usarlas de manera efectiva. Para tratar con la complejidad de un sistema de interés, los ingenieros y científicos usan versiones simplificadas del sistema, es decir, modelos del mismo [1].

Es común encontrar palabras como sistema, modelo, y simulación muy a menudo en la literatura. Sin el interés de formalizar mucho en el asunto damos algunas definiciones para términos comúnmente encontrados:

- **Sistema:** Un sistema es un objeto o una colección de objetos de los cuales queremos estudiar sus propiedades.
- **Experimento:** Estimular un sistema para evaluar su respuesta.
- **Modelo:** Abstracción de la realidad para un propósito específico.
- **Simulación (experimento virtual):** Estimular un modelo del sistema de interés para evaluar su respuesta.

Podemos decir que el mejor modelo es el modelo más simple que sirve para entender un sistema y resolver problemas.

1.1. Motivación

Como un primer ejemplo, podemos pensar en el puente de Lymira, que es uno de los puentes en arco más antiguos del mundo¹.



Figura 1.1. (Arriba) Foto del puente de Lymira en la actualidad.
(Abajo) Ubicación del puente, en la actual Turquía.

La figura 1.2 muestra un esquema de la geometría simplificada para una sección de un puente similar al mostrado anteriormente. También muestra los contornos de desplazamiento verticales obtenidos por el método de los

¹Se estima que el puente fue construido en el siglo III d.C.

elementos finitos bajo la aplicación de una carga uniforme y vertical en la superficie del puente.

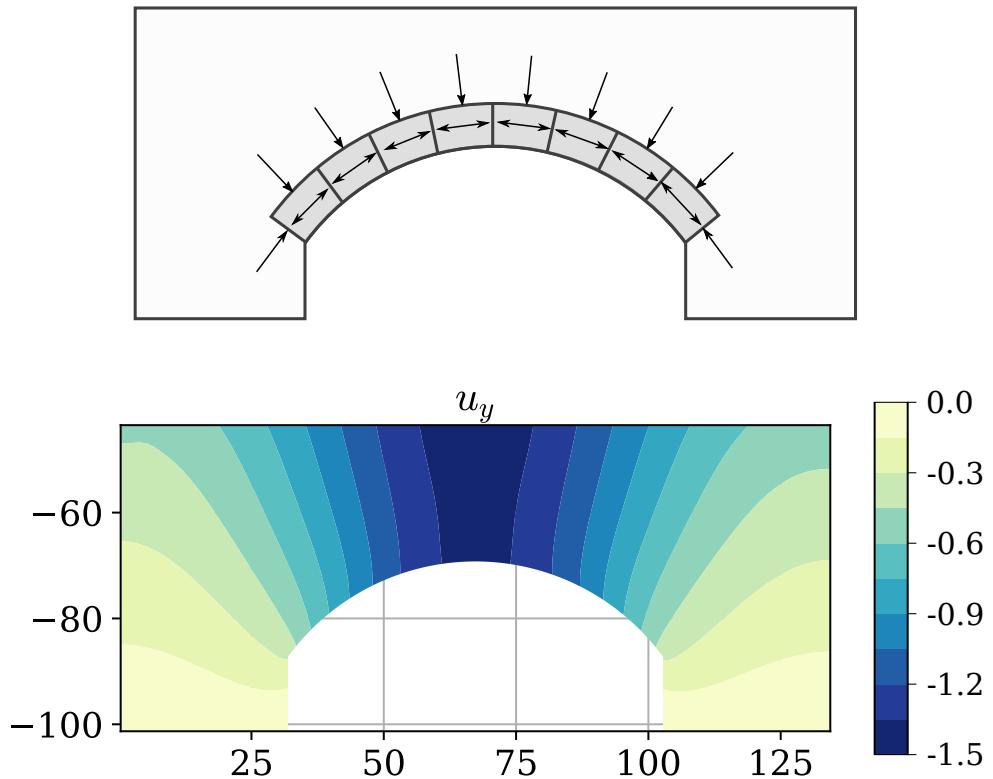


Figura 1.2. Un arco colecta las cargas verticales y las convierte en cargas laterales. Estas cargas laterales van a través del arco y son soportadas por el contrafuerte [2]. En la parte inferior vemos el resultado de una simulación por elementos finitos, los contornos representan el desplazamiento vertical en cada punto.

1.2. Ejemplos

1.2.1. Caída libre de un paracaidista

Como primer ejemplo, estudiemos la caída libre de un paracaidista. Despreciaremos la velocidad horizontal y consideraremos que la velocidad vertical inicial es cero. Para encontrar las ecuaciones de movimiento partamos de un diagrama de cuerpo libre.

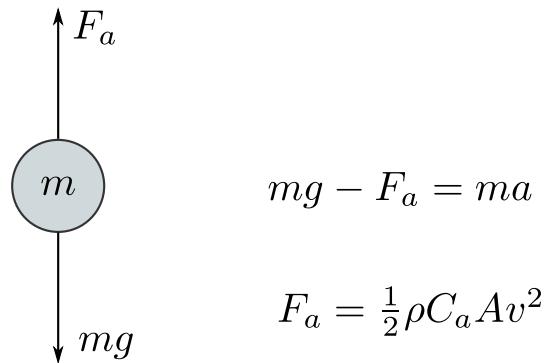


Figura 1.3. Diagrama de cuerpo libre para una partícula de masa m que cae en un fluido.

La resistencia del aire está dada por

$$F_a = \frac{1}{2} \rho C_a A v^2,$$

con ρ la densidad del fluido, C_a el coeficiente de arrastre, A la sección transversal y v la velocidad. Por tanto, tenemos la siguiente ecuación de movimiento

$$\frac{dv}{dt} = g - \frac{1}{2} \frac{\rho C_a A}{m} v^2.$$

A diferencia de la caída en la ausencia de un fluido, en este caso se tiene una “velocidad terminal” (v_∞), es decir, una velocidad máxima. Esto implica que la aceleración debe ser cero,

$$\frac{dv_\infty}{dt} = 0,$$

y por tanto

$$g = \frac{1}{2} \frac{\rho C_a A}{m} v_\infty^2, \quad \text{o} \quad v_\infty = \sqrt{\frac{2mg}{\rho C_a A}}.$$

Podemos, entonces, reescribir la ecuación diferencial como

$$\frac{dv}{dt} = g \left(1 - \frac{v^2}{v_\infty^2} \right),$$

e integrando tenemos

$$\int_0^v \frac{dv}{\left(1 - \frac{v^2}{v_\infty^2}\right)} = \int_0^t g dt,$$

y luego de una manipulación algebraica, obtenemos

$$v(t) = v_\infty \tanh\left(\frac{gt}{v_\infty}\right).$$

Tomemos los siguientes parámetros: $\rho = 1.22 \text{ kg/m}^3$, $g = 9.81 \text{ m/s}^2$, $C_a = 1.2$, $A = 0.43 \text{ m}^2$, y $m = 80 \text{ kg}$. Esto nos da una velocidad terminal

$$v_\infty = 49.94 \text{ m/s} = 179.76 \text{ km/h},$$

y la rapidez como función del tiempo sería

$$v(t) = 49.94 \tanh(0.196 t) \text{ m/s}.$$

La siguiente figura muestra esta función entre 0 y 10 segundos

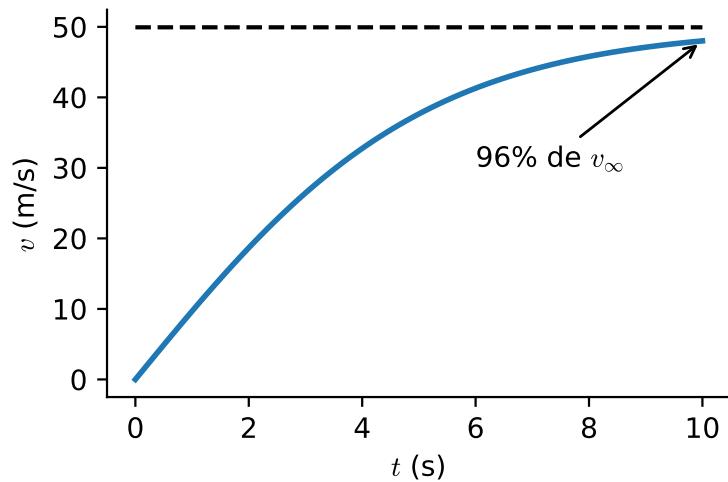


Figura 1.4. Rapidez en función del tiempo para un paracaidista en caída libre.

Ahora, supongamos que no sabemos cómo resolver la ecuación diferencial. Podemos usar el método de Euler para encontrar su solución numéricamente. El método de Euler se basa en aproximar la solución a partir de la solución en un instante actual usando una línea recta que pasa por el punto actual y tiene pendiente igual a la derivada evaluada en dicho punto. La idea se ilustra gráficamente a continuación.

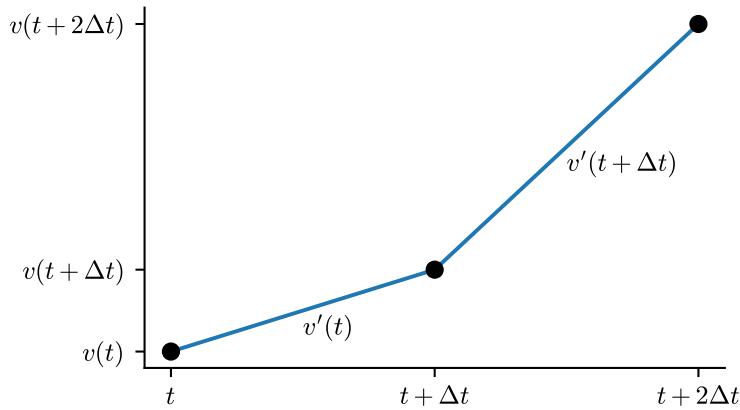


Figura 1.5. Esquema del método de Euler. El valor de velocidad para $t + \Delta t$ se calcula a partir del valor para t y la pendiente de la recta es igual a la derivada evaluada en t .

En nuestro ejemplo tenemos la siguiente ecuación diferencial

$$v'(t) \equiv \frac{dv}{dt} = g \left(1 - \frac{v^2(t)}{v_\infty^2} \right),$$

y usando la ecuación de la pendiente tenemos que

$$v'(t) \approx \frac{v(t + \Delta t) - v(t)}{(t + \Delta t) - (t)},$$

que resulta en

$$v(t + \Delta t) \approx v(t) + \Delta t v'(t),$$

y remplazando la ecuación diferencial, obtendríamos la siguiente formula iterativa.

$$v(t + \Delta t) \approx v(t) + \Delta t g \left(1 - \frac{v^2(t)}{v_\infty^2} \right).$$

Esto nos permitiría resolver la ecuación de forma iterativa en una hoja de cálculo o con algún lenguaje de programación. A continuación se muestra el código en Python para este problema.

```
import numpy as np

rho = 1.22 # kg/m**3
g = 9.81 # m/s**2
Ca = 1.2
A = 0.43 # m**2
m = 80.0 # kg
v_inf = np.sqrt(2*m*g/(rho * Ca * A))
t = np.linspace(0, 10, 10)
dt = t[1] - t[0]
niter = len(t)
v = np.zeros(niter)
for cont in range(1, niter):
    v[cont] = v[cont - 1] + dt * g * (1 - v[cont - 1]**2/v_inf**2)
```

Código 1. Método de Euler para el problema de caída libre de un paracaidista.

La figura 1.6 presenta la solución por este método y la compara con la solución analítica.

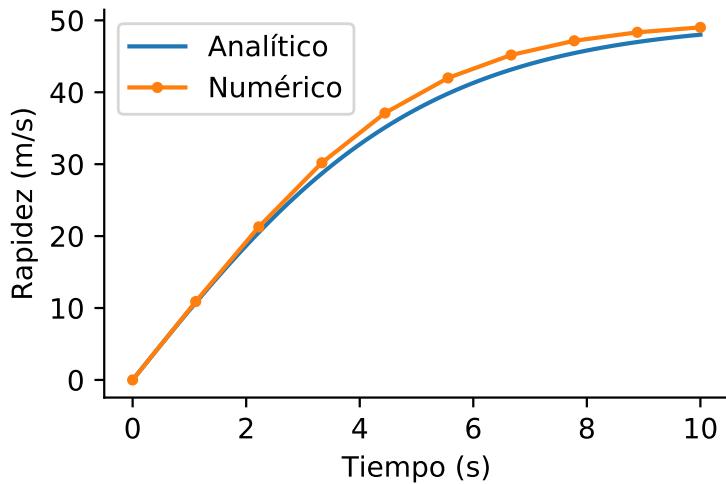


Figura 1.6. Rapidez en función del tiempo para un paracaidista en caída libre calculada con el método de Euler.

1.2.2. Vibración en una cuerda

En este ejemplo queremos estudiar la vibración en una cuerda con pequeñas amplitudes. Para ello, partimos de la ecuación de conservación del moméntum lineal (segunda ley de Newton). Asumiremos que la cuerda es homogénea y la tensión experimentada es la misma a lo largo de ella. Tomemos un elemento diferencial de la cuerda como se muestra en la siguiente figura.

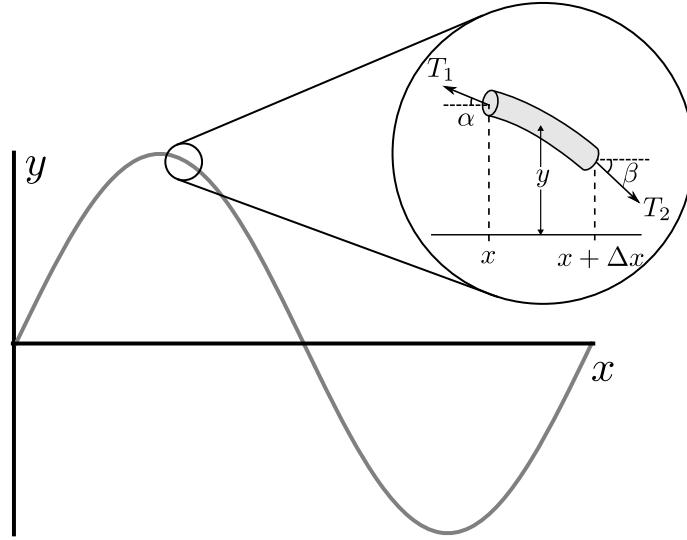


Figura 1.7. Fuerzas actuando sobre un elemento diferencial de una cuerda de longitud Δm .

Horizontalmente tenemos

$$\begin{aligned} T_{1x} &= T_1 \cos \alpha \approx T \\ T_{2x} &= T_2 \cos \beta \approx T. \end{aligned}$$

Si los ángulos son pequeños las fuerzas horizontales son iguales y su sumaatoria es cero. Verticalmente, tenemos entonces

$$\sum F_y = T_{1y} - T_{2y} = -T_2 \sin \beta + T_1 \sin \alpha = \Delta m a \approx \mu \Delta x \frac{\partial^2 y}{\partial t^2},$$

en donde tomamos que la masa del elemento diferencial $\Delta m = \mu \Delta x$, siendo μ la densidad de masa por unidad de longitud. Si dividimos por T y remplazamos la primera ecuación para T , obtenemos

$$\frac{\mu \Delta x}{T} \frac{\partial^2 y}{\partial t^2} = -\frac{T_2 \sin \beta}{T_2 \cos \beta} + \frac{T_1 \sin \alpha}{T_1 \cos \alpha} = -\tan \beta + \tan \alpha.$$

Y sabemos que las tangentes de estos ángulos son iguales a las pendientes, es decir,

$$\frac{1}{\Delta x} \left(\frac{\partial y}{\partial x} \Big|_{x+\Delta x} - \frac{\partial y}{\partial x} \Big|_x \right) = \frac{\mu}{T} \frac{\partial^2 y}{\partial t^2}.$$

En el límite $\Delta x \rightarrow 0$, tenemos

$$\frac{\partial^2 y(x, t)}{\partial x^2} = \frac{1}{c^2} \frac{\partial^2 y(x, t)}{\partial t^2},$$

con $c^2 = T/\mu$, la velocidad de propagación de la onda. Esta velocidad es directamente proporcional a la raíz cuadrada de la tensión en la cuerda e inversamente proporcional a la raíz cuadrada de la densidad lineal de masa.

A este tipo de ecuaciones se le denomina “ecuaciones en derivadas parciales” porque ecuación incluye la incógnita, $y(x, t)$, así como sus derivadas parciales. A fijar la cuerda en ambos extremos denominamos condiciones de frontera, y a conocer su configuración inicial (posición y velocidad) condiciones iniciales. Para este problema se cuenta con una solución analítica, sin embargo, usaremos una estrategia numérica llamada “diferencias finitas”.

Este método aproxima las derivadas por medio de *diferencias* de la función en puntos adyacentes, y se asemeja a la definición de derivada usando el límite, es decir

$$\frac{\partial^2 u(x, t)}{\partial x^2} \approx \frac{u_{x+1}^t - 2u_x^t + u_{x-1}^t}{\Delta x^2},$$

y

$$\frac{\partial^2 u(x, t)}{\partial t^2} \approx \frac{u_x^{t+2} - 2u_x^{t+1} + u_x^t}{\Delta t^2},$$

en donde los subíndices se refieren a una discretización en el espacio (x) y los superíndices se refieren a una discretización en el tiempo (t).

Usando estas aproximaciones en la ecuación diferencial, tenemos

$$\frac{y_{x+1}^t - 2y_x^t + y_{x-1}^t}{\Delta x^2} = \frac{1}{c^2} \left(\frac{y_x^{t+1} - 2y_x^t + y_x^{t-1}}{\Delta t^2} \right),$$

y despejando y_x^{t+1} ,

$$y_x^{t+2} = 2y_x^{t+1} - y_x^t + \frac{c^2 \Delta t^2}{\Delta x^2} [y_{x+1}^t - 2y_x^t + y_{x-1}^t], \quad (1.1)$$

y podemos ver que tenemos una fórmula recursiva para obtener la solución en el tiempo $t + 2$ a partir de la solución en dos tiempos anteriores.

Esta fórmula no funciona para la primera iteración, pues requiere conocer la solución en un instante anterior al tiempo inicial (y_x^{-1}), sin embargo podemos usar la condición inicial para la primera derivada de y para obtener

$$y_x^1 = y_x^0 - \Delta y'_x - \frac{1}{2} \frac{c^2 \Delta t^2}{\delta x^2} [y_{x+1}^0 - 2y_x^0 + y_{x-1}^0].$$

Capítulo 2

Métodos Numéricos

Presentación

Una amplia gama de problemas de ingeniería se reducen a la solución de un sistema de ecuaciones diferenciales que gobiernan el comportamiento de una región 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 características 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 analíticos que produzcan soluciones cerradas, es decir funciones. Como alternativa de solución aparece 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 con una base mecánica como mecánica de suelos, mecánica de sólidos, y 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 más complejos se usan los métodos de elementos finitos (FEM por sus siglas en inglés), los métodos de diferencias finitas (FDM por sus siglas en inglés) y el método de elementos de frontera (BEM por sus siglas en inglés).

En este capítulo revisamos de manera práctica algunas técnicas 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 raíces)
- Interpolación
- Integración numérica

Al final de este capítulo 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 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 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 o con la ayuda de implementaciones en el computador problemas de cálculo numérico de integrales de funciones.

2.1. 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.

2.1.1. Planteamiento del problema

Consideremos una función escrita de la forma

$$y = f(x) \quad (2.1)$$

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 2.1.

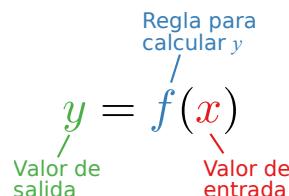


Figura 2.1. Descripción del concepto de función como una regla de transformación de valores (posiblemente) reales en otros valores (posiblemente) reales.

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

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

El problema de búsqueda 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 . Matemáticamente esta pregunta la podemos plantear como encontrar valores de x que satisfagan la condición:

$$f(x) = 0.0.$$

Por ejemplo, la figura 2.2 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 intersección de la línea $y = 0$ y la función corresponde a una raíz de la función. Esta tiene un valor aproximado de $x = 1.4$.

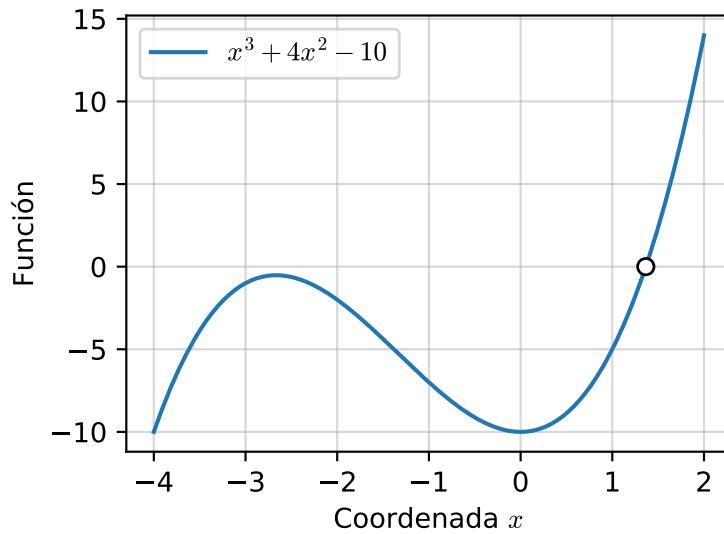


Figura 2.2. Concepto de raíz de una función. La función sobre el intervalo $[-4.0, 2.0]$ se muestra con la línea azul y la raíz con el círculo blanco.

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 específica 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:

$$R_T Q^3 - \Delta H Q + \phi = 0. \quad (2.2)$$

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

$$f(Q) = 0.0,$$

donde

$$f(Q) = R_T Q^3 - \Delta H Q + \phi. \quad (2.3)$$

En computación se denomina **iteración** 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 2.3):

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

Antes de iniciar el estudio de un par de métodos clásicos para determinación de raíces es importante tener una idea de los conceptos de convergencia y divergencia. Intuitivamente, decimos que un método converge cuando los valores se acercan cada vez más y que diverge cuando pasa lo contrario, se alejan entre ellos. Adicionalmente, llamamos **tasa de convergencia** a la rapidez con la que un cálculo iterativo se aproxima a una solución. Ahora, concentrémonos en la figura 2.3 en la cual se grafica la variación del error vs el número de iteraciones en un algoritmo. 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. Podemos ver que uno de los métodos converge más rápidamente que el otro.

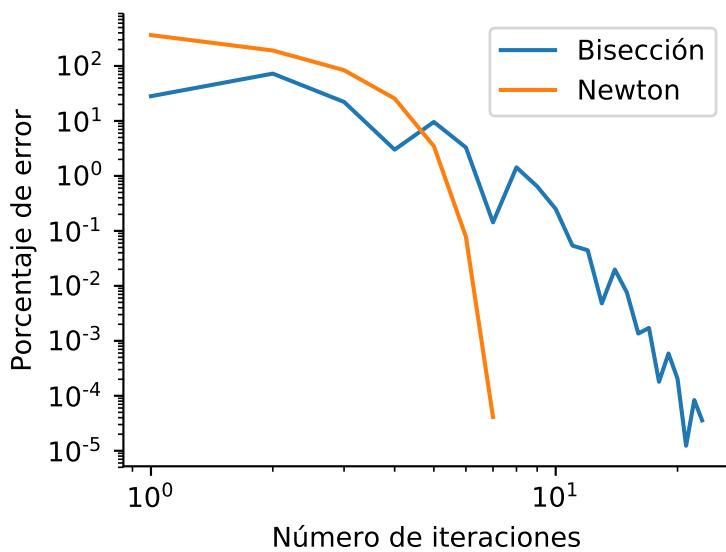


Figura 2.3. Concepto de convergencia o número de iteraciones requeridas para alcanzar una solución.

2.1.2. 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 definición aceptable de cero para determinar si efectivamente se encontró la raíz. Esta es necesaria ya 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 **determinación** de la raíz. En resumen, el proceso de búsqueda 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 límites 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 2.4 en la que se presenta una función que tiene 2 raíces (círculos 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 función $f(x)$ cuya raíz se desea determinar. El algoritmo entregará como resultado las raíces 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 incrementará en 1 cada que el algoritmo detecte una raíz.

La parte principal del algoritmo consiste en un ciclo que recorre los N subintervalos en los que se ha partido el problema y detecta cambios de signo haciendo 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 apéndice de estas notas.

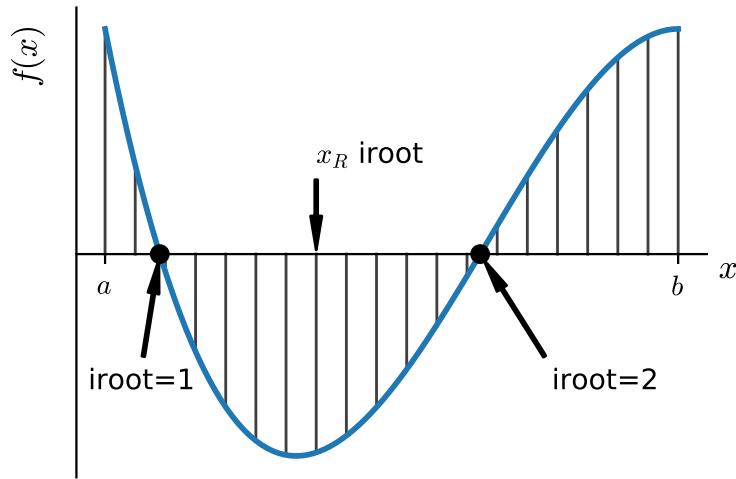


Figura 2.4. Proceso de detección de raíces de una función en un intervalo $[a, b]$.

Algoritmo 1: Detección o *bracketing*.**Data:** a, b, N, f **Result:** x_R : Vector con la aproximación de las raíces

```

 $\Delta x \leftarrow \frac{b - a}{N - 1};$ 
 $iroot \leftarrow 0;$ 
 $x_2 \leftarrow a;$ 
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. 1. 0. 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 refinamiento de la raíz.

2.1.3. 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 2.5, 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 .

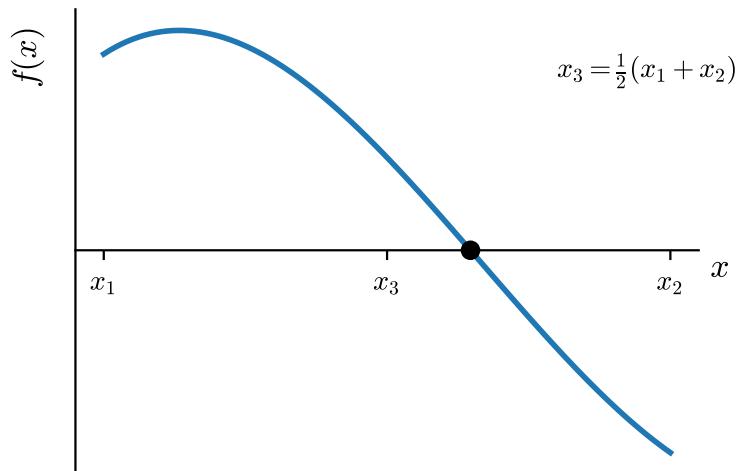


Figura 2.5. Método de bisección para encontrar las raíces.

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

$x_2 \leftarrow x_3$

$x_3 \leftarrow \frac{1}{2}(x_1 + x_2)$

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

$x_1 \leftarrow x_3$

$x_3 \leftarrow \frac{1}{2}(x_1 + x_2)$

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 ;

$n_{\max} \leftarrow \lceil \log_2 \left(\frac{b-a}{tol} \right) \rceil$;

for $cont \leftarrow 0$ to n_{\max} **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 apéndice se presenta el código en Python correspondiente 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
```

2.1.4. Método de Newton-Raphson

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 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 derivadas. 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.

Derivación matemática

Sea x_i la i -ésima¹ aproximación a la raíz buscada y p el valor refinado de la raíz que se encuentra en la vecindad de x_i . Escribiendo la función como una serie de Taylor alrededor de p se tiene:

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

Luego, asumiendo que $f(p)$ es cercano a cero ($f(p) \approx 0$) 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 términos de orden 2 y superiores y retener solo los términos lineales en una expresión. 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. El proceso se ilustra en la figura 2.6 en donde se ve que para iniciar las iteraciones

¹Por ejemplo, x_0 es el valor encontrado por el algoritmo de detección.

se requiere una aproximación inicial (x_0) a la raíz. Este valor inicial 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.

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \quad (2.4)$$

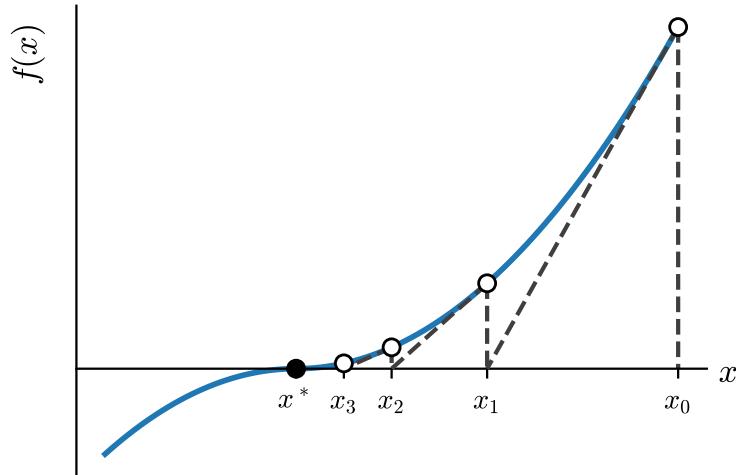


Figura 2.6. Iteración de Newton-Raphson.

El algoritmo 3 presenta el método de Newton para un punto inicial x mientras que su correspondiente código en Python se presenta en los apéndices.

Algoritmo 3: Newton-Raphson

Data: x , tol , n_{\max} , f , f'
Result: Raíz aproximada
for $cont \leftarrow 0$ to n_{\max} **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 $x_0 = 2.0$ 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 2.7) en el intervalo $[-10.0, 10.0]$.

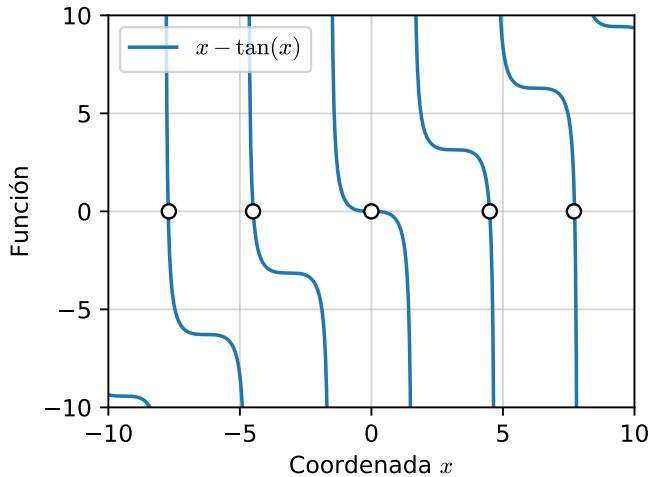


Figura 2.7. Función $f(x) = x - \tan x$.

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 $R_T = 149 \text{ s}^2/\text{m}^5$, un salto o diferencia de niveles $\Delta H = 500 \text{ m}$ y una potencia específica $\phi = 2548 \text{ m}^4/\text{s}$. 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:

$$H^3\sigma_{\text{adm}} - HP \pm 6(M_1 + M_2) = 0.0 \quad (2.5)$$

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 $P = 32 \text{ tf}$ y los momentos $M_1 = 18 \text{ tf} \cdot \text{m}$ y $M_2 = 23 \text{ tf} \cdot \text{m}$.

2.2. 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.

2.2.1. Planteamiento del problema

Sea $f(x)$ una función desconocida analiticamente pero con valores conocidos de manera discreta en n puntos x_1, x_2, \dots, 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 $\{f_1, f_2, \dots, f_n\}$. Por ejemplo,

tomemos los valores discretos de una función $f(x)$ que se tiene en la tabla 2.1 y que se muestran mediante círculos negros en la figura 2.8.

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

Tabla 2.1. Valores conocidos de la función $f(x)$.

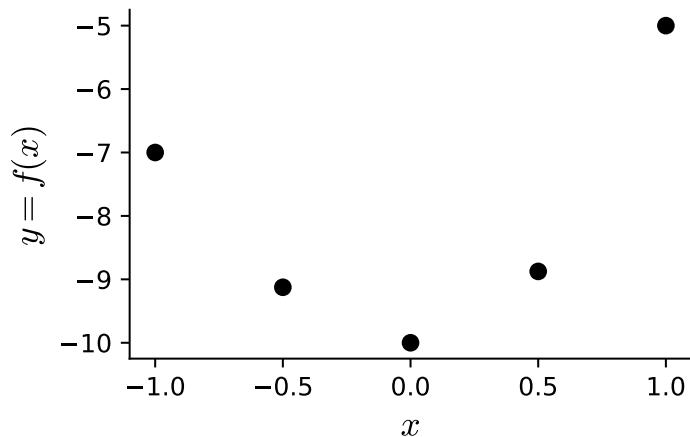


Figura 2.8. Puntos correspondientes a valores conocidos de una función. Se desea determinar el valor de la función para puntos diferentes a los de medición.

Para determinar los valores desconocidos de la función $f(x)$ usando los valores dados $\{f_1, f_2, \dots, f_n\}$ 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 2.9. 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.

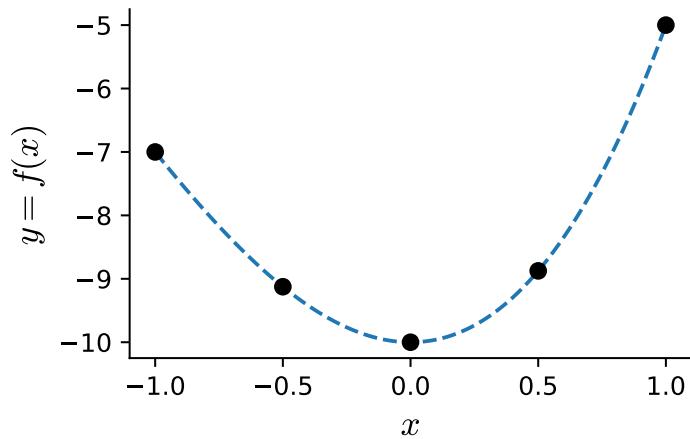


Figura 2.9. Polinomio de interpolación usando los n -datos. El polinomio resultante es de orden $n - 1$.

Alternativamente, es posible dividir el dominio del problema en subdominios y proceder con interpolaciones locales. Esta alternativa se muestra en la figura 2.10 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 función 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 2.11 muestra la distribución de los equipos de registro sísmico que conforman la red acelerográfica de Medellín. Tras la ocurrencia de un

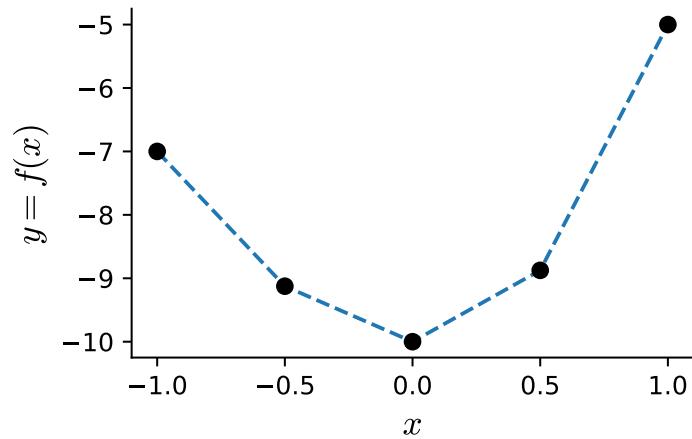


Figura 2.10. Función de interpolación definida por tramos usando pares de puntos para producir polinomios locales de orden 1.

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.

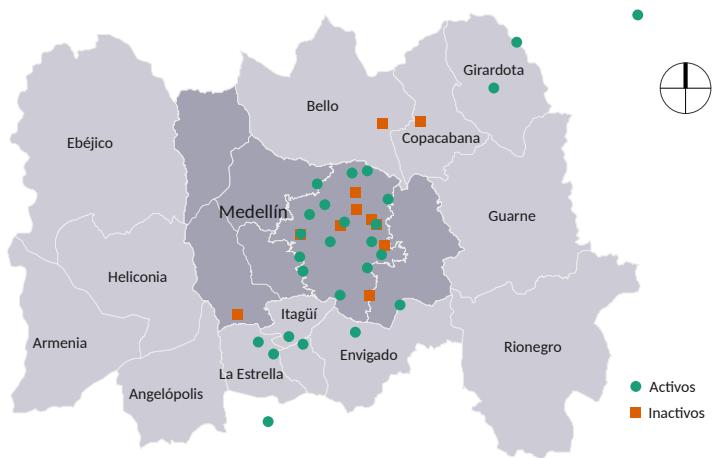


Figura 2.11. Red acelerográfica de Medellín. Los círculos verdes representan estaciones activas. Mientras que los cuadrados naranja son estaciones inactivas. Datos de julio de 2017, tomados de SIATA [3]

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.

2.2.2. Teorema de interpolación de Lagrange

Dado un conjunto de n -puntos $\{(x_1, y_1), \dots, (x_n, y_n)\}$, 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 $i = 1, 2, \dots, n$.

El polinomio está dado por

$$p(x) = L_1(x)f(x_1) + L_2(x)f(x_2) + \dots + L_n(x)f(x_n), \quad (2.6)$$

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

$$L_i(x) = \prod_{\substack{j=1 \\ i \neq j}}^n \frac{(x - x_j)}{(x_i - x_j)}, \quad (2.7)$$

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 común denominar la función $p(x)$ con la que se aproxima la función $f(x)$ como **polinomio de interpolación** 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 función usando 3 valores conocidos. El tabla 2.2 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 2.2. 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 (2.7), obtenemos

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

Los polinomios $L_1(x)$, $L_2(x)$ y $L_3(x)$ se muestran en la figura 2.12. 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.

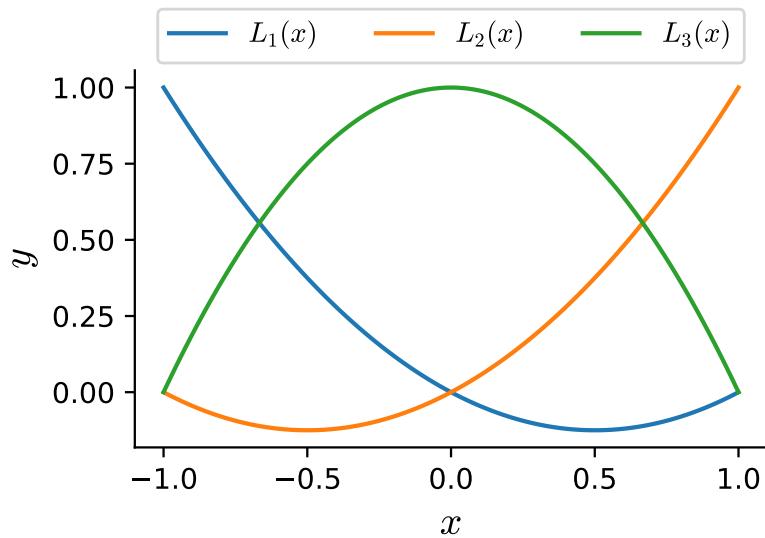


Figura 2.12. Polinomios interpolantes de Lagrange correspondientes a los puntos $x_1 = -1.0$, $x_2 = 1.0$ y $x_3 = 0.0$

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 2.13. 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.

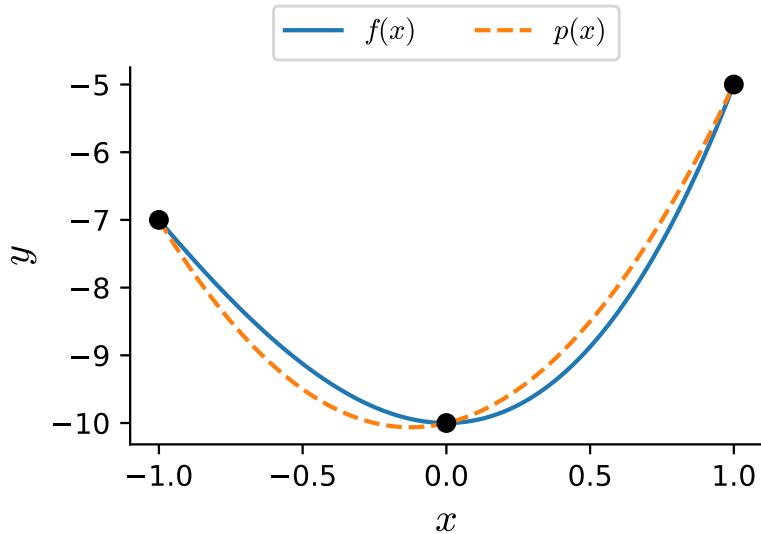


Figura 2.13. Función de interpolación (en línea punteada) resultante tras aplicar la superposición de la ecuación (2.7) conjuntamente con los datos de la tabla 2.2 (puntos negros) y correspondientes a la función $f(x) = x^3 + 4x^2 - 10$ (línea continua) $x_1 = -1.0$, $x_2 = 1.0$ y $x_3 = 0.0$

Supongamos que además se desea calcular una aproximación a la primera derivada de la función. Para calcular dichas aproximaciones derivamos la ecuación (2.7) para obtener

$$\frac{dp(x)}{dx} = \frac{dL_1(x)}{dx}f_1 + \frac{dL_2(x)}{dx}f_2 + \frac{dL_3(x)}{dx}f_3 \quad (2.8)$$

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 2.14.

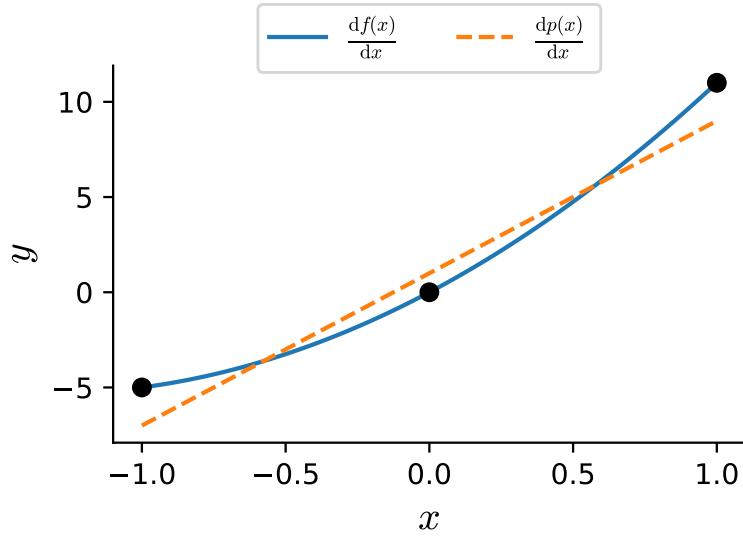


Figura 2.14. Comparación entre la derivada exacta de $f(x)$ y la derivada obtenida a partir de la aproximación con polinomios de Lagrange usando la ecuación (2.8).

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 (2.8) se usan los valores conocidos de f y no los de $\frac{df(x)}{dx}$ y se usan como polinomios $\frac{dL_i(x)}{dx}$, los cuales no satisfacen la propiedad de interpolación, es decir

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

Con el propósito de asimilar la variación de la solución con los diferentes esquemas de interpolación, la figura 2.15 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 usamos interpolación para se aproximar la función de desplazamientos entonces también es posible determinar la aproximación a la función de deformaciones. Sin embargo, esta última sería una función de más bajo nivel de aproximación.

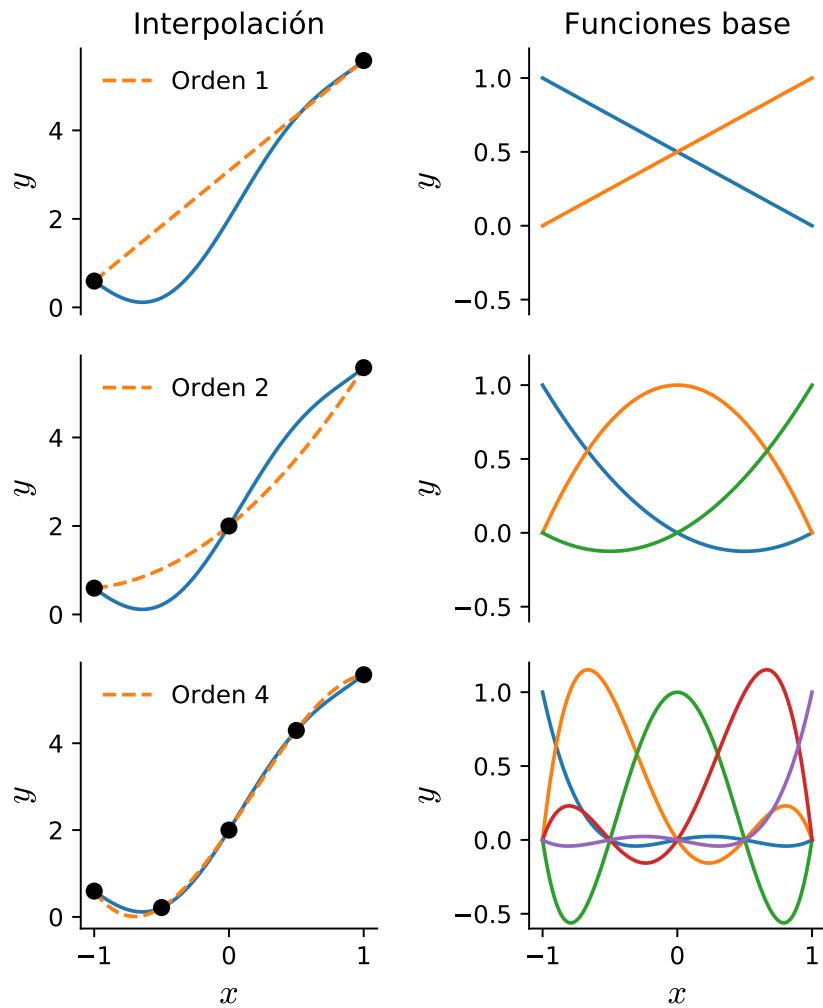


Figura 2.15. Interpolación para la función $f(x) = 2e^x + \sin(3x)$ mediante polinomios de diferente orden.

Finalmente, la figura 2.16 muestra las primeras derivadas de los polinomios de interpolación de orden 4 así como la aproximación a la primera derivada usando la expresión

$$\frac{dp(x)}{dx} = \sum_i \frac{dL_I(x)}{dx} f_i .$$

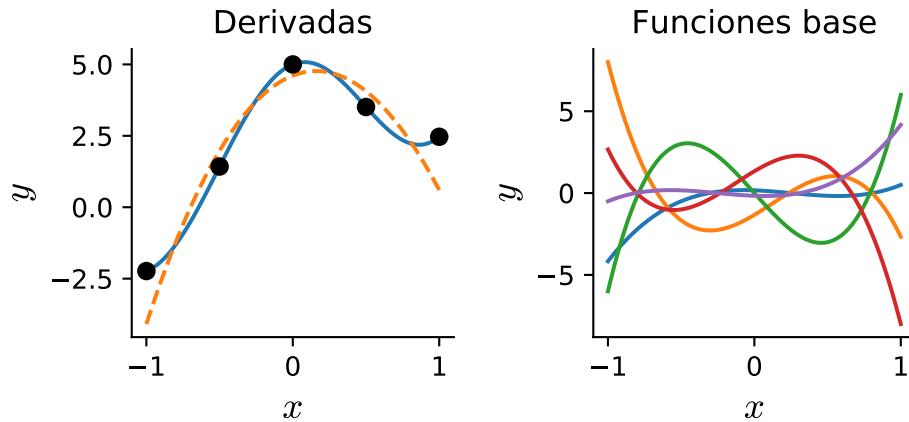


Figura 2.16. Derivadas de los polinomios de interpolación de orden 4 y aproximación a la primera derivada de la función $f(x) = 2e^x + \sin(3x)$.

2.2.3. Distribución de los puntos de muestreo

Considere la función

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

mostrada en la figura 2.17 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 2.18 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

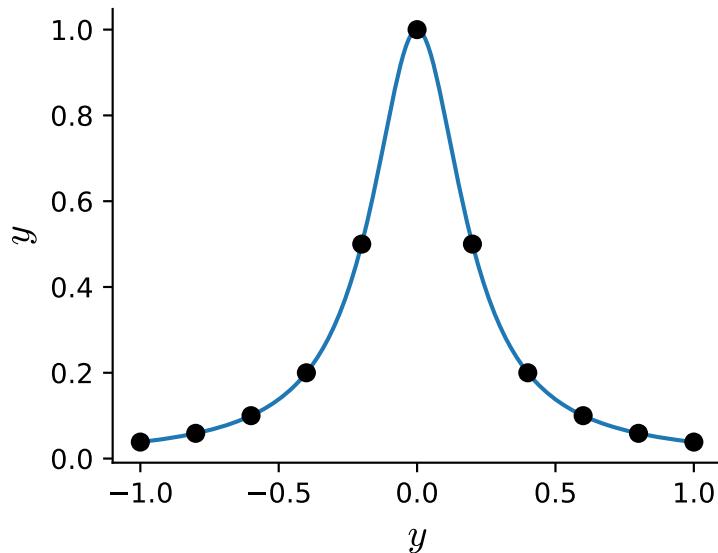


Figura 2.17. Función de Runge $f(x) = \frac{1}{1+25x^2}$.

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

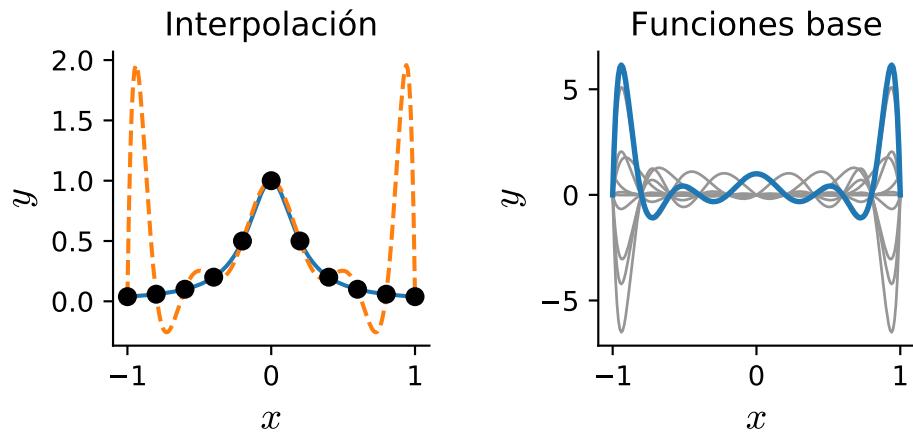


Figura 2.18. Función de interpolación para aproximar la función de Runge usando los puntos de muestreo mostrados en figura 2.17. A la derecha se muestran las funciones base para esta interpolación, se resalta la función de interpolación para el nodo de la mitad.

La aproximación se puede mejorar variando el espaciamiento de los puntos de muestreo como se muestra en la figura 2.19 donde los nodos tienen una mayor densidad en los extremos del dominio de solución.

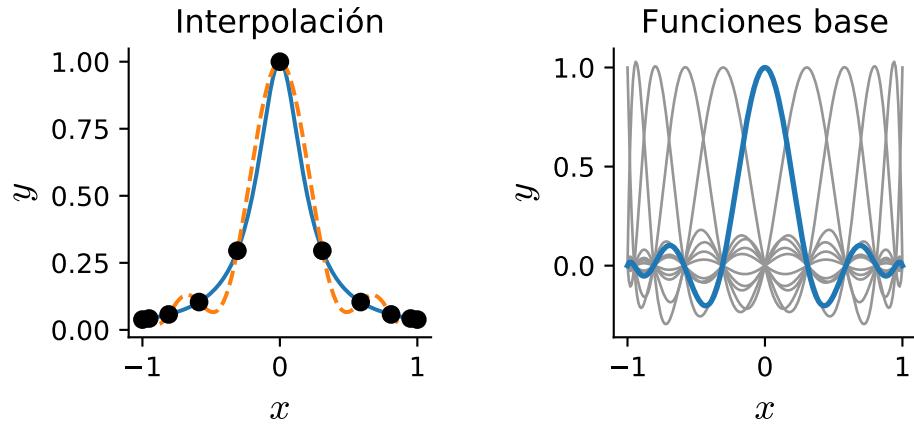


Figura 2.19. Función de interpolación para aproximar la función de Runge usando los puntos de muestreo no equidistantes. A la derecha se muestran las funciones base para esta interpolación, se resalta la función de interpolación para el nodo de la mitad. Note que las funciones base toman valores mucho 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 2.18) que se muestran en la parte (a) de la figura 2.20. 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 análoga, 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.

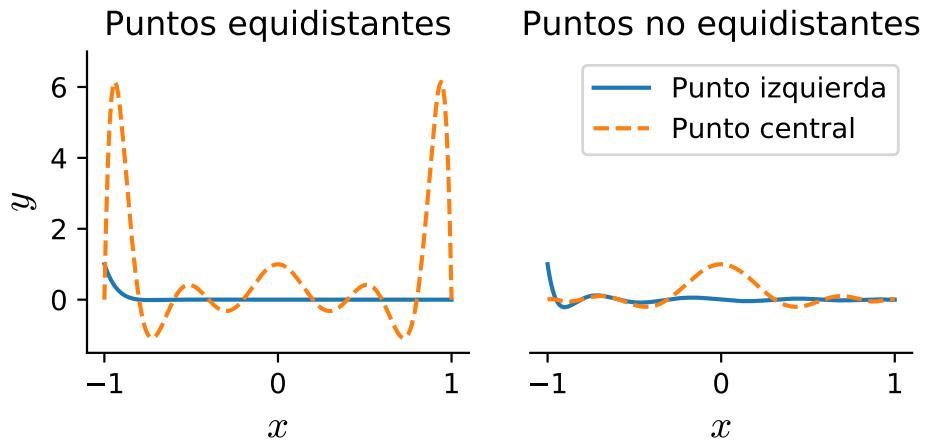


Figura 2.20. Polinomios de Lagrange de orden 10 asociados con el punto inicial y el punto medio para diferentes muestreos.

2.2.4. 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 2.3 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 2.3. 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 2.21 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.

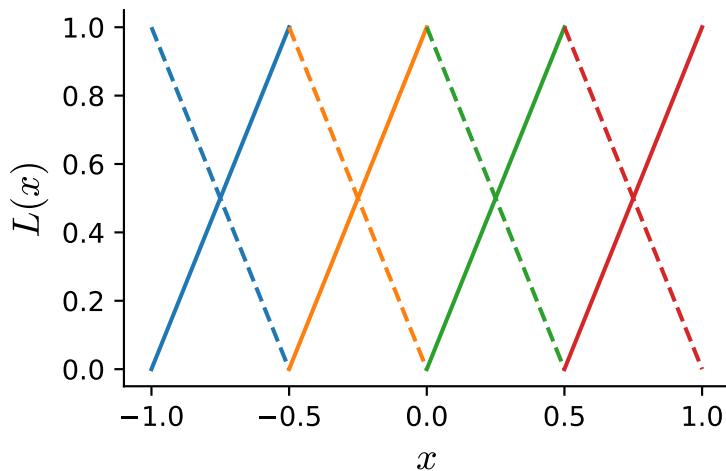


Figura 2.21. Polinomios de interpolación local. Cada dominio tiene dos líneas rectas como base. La interpolación en cada uno de ellos es la combinación lineal estas rectas.

Esta estrategia resulta en una interpolación de la función como la mostrada en la figura 2.22 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.

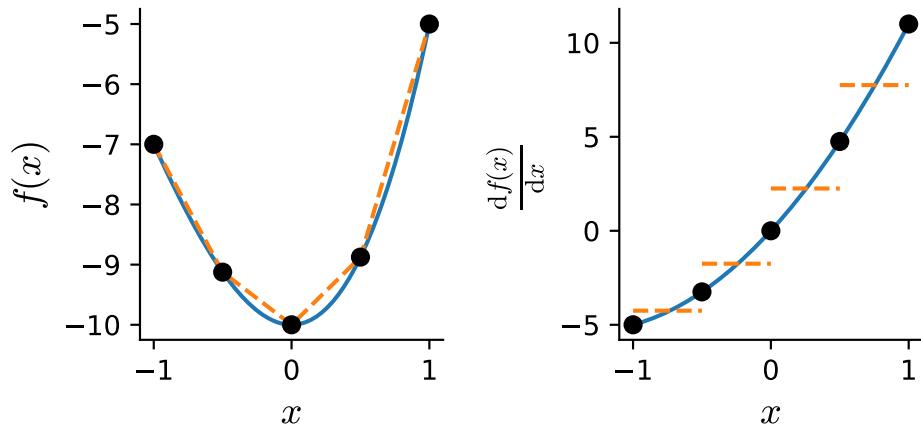


Figura 2.22. Interpolación lineal por tramos de la función $f(x) = x^3 + 4x^2 - 10$.

2.2.5. Generalización a dominios bi-dimensionales

Asumamos que estamos interesados en interpolar una función definida en un dominio plano (ver figura 2.23) 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 $\{(\mathbf{x}_1, f_1), \dots, (\mathbf{x}_n, f_n)\}$. Este problema es la base para resolver el presentado al principio de la sección relativo a la Red Acelerográfica de Medellín.

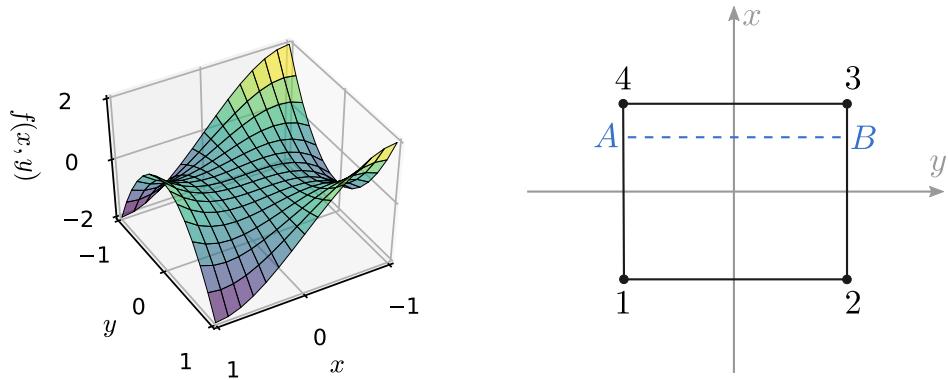


Figura 2.23. Función $f(x, y)$ sobre un dominio cuadrado con puntos de muestreo rotulados como 1, 2, 3 y 4

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 2.24.

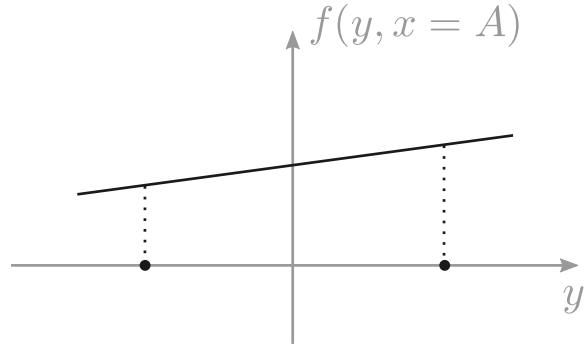


Figura 2.24. Interpolación en la dirección y sobre la línea 1-4.

Usando una expresión análoga a la ecuación (2.6) 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,

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

La figura 2.25 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.

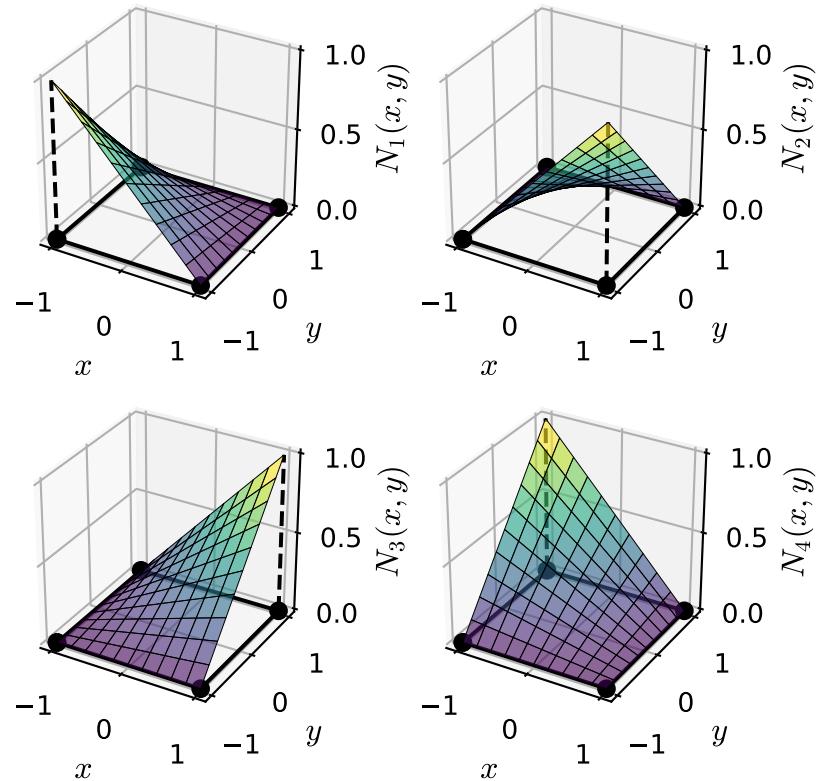


Figura 2.25. Polinomios de interpolación para un dominio bidimensional con 4 puntos de muestreo.

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 2.3 pero esta vez con al menos 4 sub-dominios cada uno de 3 puntos de muestreo para el intervalo $[-1.0, 1.0]$. Usando los 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 última 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.

2.3. 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.

2.3.1. Planteamiento del problema

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

$$I = \iiint f(x, y, z) dV \quad (2.9)$$

donde la integral triple representa integración sobre un volumen determinando. 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:

$$\int_a^b f(x) dx \approx \sum_{i=1}^n w_i f(x_i), \quad (2.10)$$

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 2.4 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 2.4. Abscisas y factores de ponderación para calcular $\int_{-1}^{+1} f(x) dx$.

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:

$$\begin{aligned} \int_{-1}^{+1} (x^3 + 4x^2 - 10) dx &\approx 0.34785 \cdot f(-0.86113) + 0.65214 \cdot f(-0.33998) \\ &\quad + 0.34785 \cdot f(0.86113) + 0.65214 \cdot f(0.33998) = -17.3333 \end{aligned}$$

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.

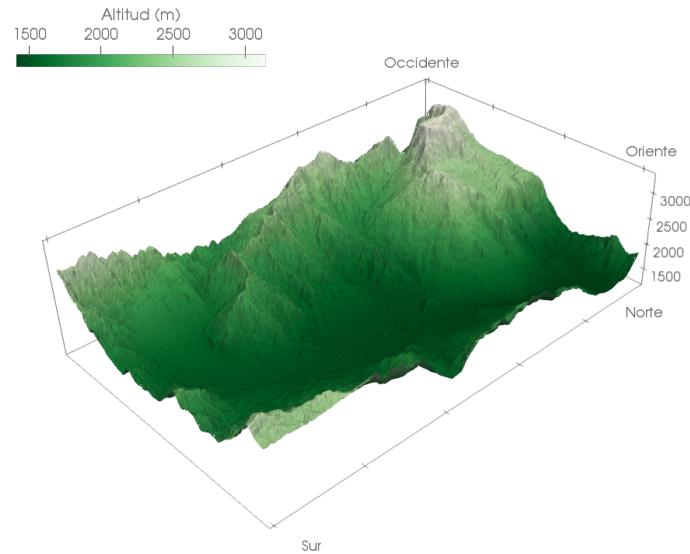


Figura 2.26. Topografía del Valle de Aburrá.

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

$$A = \int_{\text{Dominio}} \sqrt{\left(\frac{\partial f(x, y)}{\partial x} \right)^2 + \left(\frac{\partial f(x, y)}{\partial y} \right)^2 + 1} \, dx \, dy ,$$

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.

2.3.2. 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 $\int_a^b p(x) dx$ 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:

$$\int_a^b f(x) dx \approx \int_a^b p(x) dx ,$$

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:

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

la cual es equivalente a la ecuación (2.10) y en la cual se identifica que los factores de ponderación están dados por:

$$w_i \equiv \int_a^b L_i(x) dx . \quad (2.11)$$

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.

2.3.3. Regla del trapeo

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:

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

Reemplazando en (2.11) se tiene que:

$$\begin{aligned} w_1 &= -\frac{1}{h} \int_a^b (x - b) dx \equiv \frac{h}{2}, \\ w_2 &= +\frac{1}{h} \int_a^b (x - a) dx \equiv \frac{h}{2}, \end{aligned}$$

de donde:

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

Y, finalmente,

$$\int_a^b f(x) dx \approx \frac{h}{2} [f(a) + f(b)]. \quad (2.12)$$

Ejemplo

Calcular

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

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 (2.12). 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 2.27)

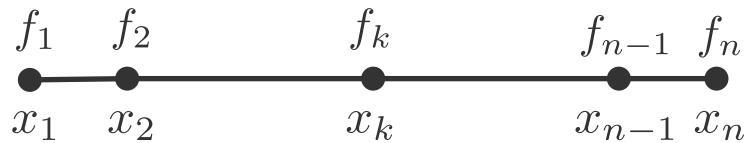


Figura 2.27. Intervalo de integración extendido con puntos de muestreo entre x_1 y x_n .

Aplicando la ecuación (2.12) $n - 1$ veces para hacer la integración en los sub-intervalos $(x_1, x_2), (x_2, x_3), \dots, (x_{n-1}, x_n)$ y sumando los resultados se obtiene:

$$\int_a^b f(x) dx = h \left[\frac{1}{2}f_1 + f_2 + f_3 + f_4 + \dots + f_{n-1} + \frac{1}{2}f_n \right] + O \left[\frac{(b-a)^3 f''}{n^2} \right] \quad (2.13)$$

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

En el apéndice se describe la función **trapz** (correspondiente a la ecuación (2.13)) 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 cálculo de la integral del ejemplo anterior 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 están localizados en ciertas posiciones de manera que la cuadratura sea lo más eficiente posible.

2.3.4. Cuadraturas Gaussianas

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

$$\int_a^b f(x) dx \approx \sum_{i=1}^n w_i f(x_i), \quad (2.14)$$

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:

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

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 2.5 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 2.5. Abscisas y factores de ponderación para calcular $\int_{-1}^{+1} f(x) dx$

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 2.3.5. La figura 2.28 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.

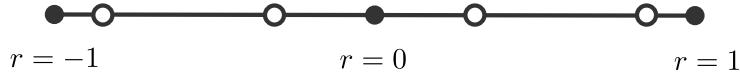


Figura 2.28. Esquematización de una cuadratura Gaussiana en el intervalo canónico $[-1.0, 1.0]$.

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

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 coincide con el del intervalo canónico. 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:

$$f(x) = a_0 + a_1x + a_2x^2 + a_3x^3.$$

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

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

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

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

donde identificamos un término constante, un término de orden 1, un término cuadrático y un término de orden 3. Para que la cuadratura sea exacta cada uno de estos términos debe poder ser integrado de manera exacta. Usando esta condición se tiene:

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

Se tiene un sistema de ecuaciones en 4 incógnitas 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 expresión de la cuadratura I se tiene:

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

Claramente esta cuadratura es exacta si se trata de integrar funciones polinómicas de orden 3 o menor. Notese que la posibilidad de ajustar 4 parámetros (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 simétrica 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

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

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

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

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

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

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

$$I = \int_{-1}^{+1} f(x) dx = \sum_{i=1}^n w_i f(x_i) . \quad (2.15)$$

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 (2.15).
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:

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

la cual se reduce a:

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

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.

2.3.5. 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 canónico y descrito mediante una variable independiente, comúnmente denotada como r . Se tiene entonces que el espacio de la variable independiente correspondiente a $x \in [a, b]$ se transforma el espacio canónico 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 representación 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:

$$\int_a^b f(x) dx \equiv \int_{-1}^{+1} F(r) dr . \quad (2.16)$$

La transformación 2.16 se describe en la figura 2.29 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 $r = 1$.

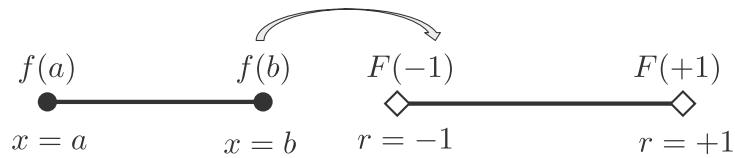


Figura 2.29. Transformación del rango de integración $[a, b]$ al intervalo canónico $[-1.0, +1.0]$.

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 mínimamente 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 correspondiente diferencial en el espacio canonico dr . Procediendo directamente desde la transformación en términos de los polinomios de Lagrange se tiene que:

$$\frac{dx}{dr} = \frac{dL_1(r)}{dr}x(r_1) + \frac{dL_2(r)}{dr}x(r_2) \equiv \frac{1}{2}(b - a),$$

y por lo tanto

$$\frac{dx}{dr} = \frac{1}{2}(b - a)$$

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

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

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

Ejemplo

Usar una cuadratura Gaussiana de 2 puntos (ver tabla 2.6) para evaluar la integral:

$$I = \int_0^3 (2^x - x) dx .$$

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

Tabla 2.6. Abscisas y factores de ponderación para calcular $\int_{-1}^{+1} f(r) dr$

Para realizar la integración usando la cuadratura Gaussiana de 2 puntos dada en la tabla 2.6 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

$$dx = \frac{3}{2} dr .$$

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

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

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
```

2.3.6. 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 2.30 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

$$I = \iint_R f(x, y) \, dA .$$

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 $\Delta x_i \times \Delta y_i$.

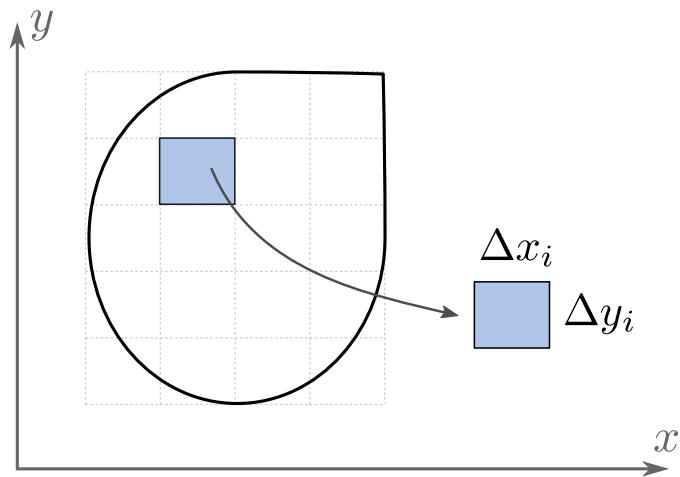


Figura 2.30. Partición de Riemann para un dominio plano.

Definiendo

$$p = \max\{\Delta x_i, \Delta y_i\} ,$$

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

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

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:

$$I = \iint f(x, y) \, dx \, dy .$$

Para identificar los límites de integración considere la figura 2.31. 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.

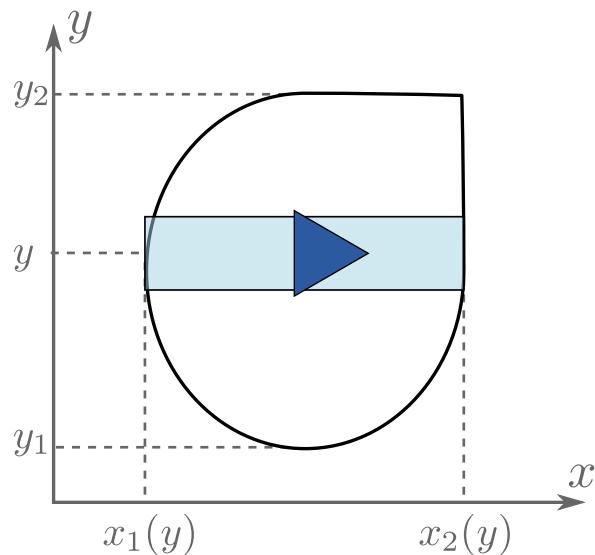


Figura 2.31. Integración en la dirección x.

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:

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

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

$$I = \int_{y_1}^{y_2} \left\{ \int_{x_1(y)}^{x_2(y)} f(x, y) dx \right\} dy \quad (2.17)$$

Para dar mayor claridad a la ecuación (2.17), nótese que la integral interna puede escribirse como una función de y

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

y la integral externa como

$$I = \int_{y_1}^{y_2} F(y) dy .$$

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 2.32) también es posible definir

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

de manera que la integral completa I queda definida por

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

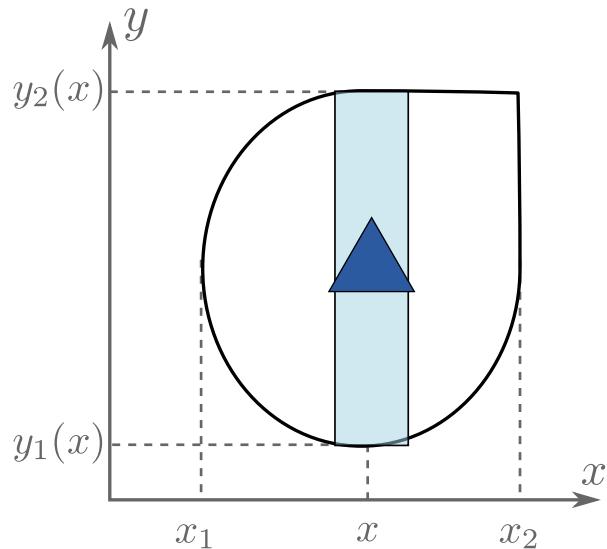


Figura 2.32. Integración en la dirección y.

Ejemplo

Considere la región (o dominio) triangular de la figura 2.33.

Utilice el concepto de integrales iteradas discutido anteriormente para

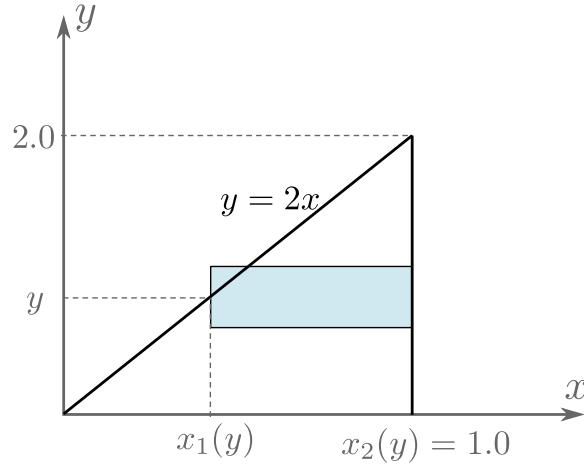


Figura 2.33. Integración en la dirección x sobre la región triangular R .

determinar la integral de la función

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

sobre la región R de la figura.

La integral sobre R está dada por

$$I = \iint f(x, y) \, dA$$

donde dA representa un elemento diferencial de superficie. Supongamos que tomaremos rectángulos 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:

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

luego

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

Identificando ahora los límites 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:

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

Procediendo de manera alternativa (ver figura 2.34) es posible escribir:

$$H(x) = \int_{y_1(x)}^{y_2(x)} xy^2 dy$$

y

$$I = \int_{x_1}^{x_2} H(x) dx .$$

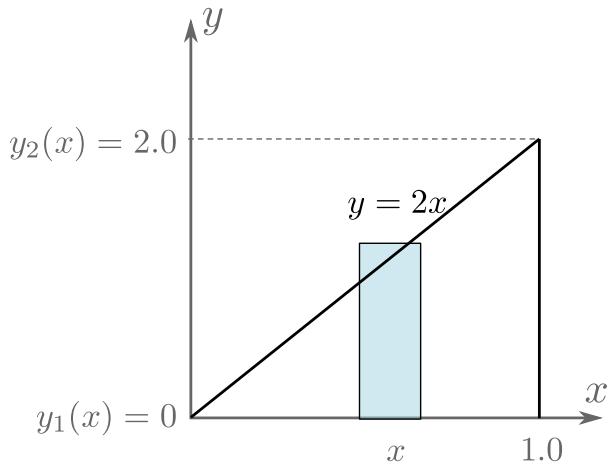


Figura 2.34. Integración en la dirección y sobre la región triangular R .

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 apéndice 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

$$\int_0^2 \int_0^2 [3xy^2 - x^3] dx dy = 8,$$

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

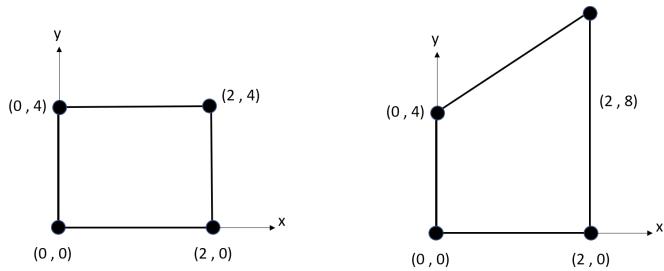


Figura 2.35. Dominios de integración para el problema 1.

2. Calcule la integral de la función

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

sobre el dominio triangular mostrado en la figura

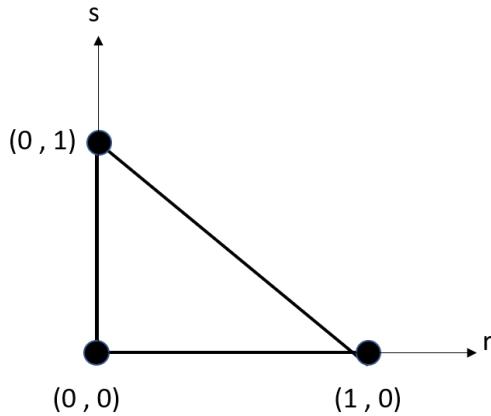


Figura 2.36. Dominios de integración para el problema 2.

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

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

$$I = \int_0^{\frac{\pi}{4}} e^{3x} \sin 2x \, dx$$

$$I = \int_0^{\frac{\pi}{4}} \cos^2 x \, dx$$

Capítulo 3

Discretización de dominios

A continuación se presenta una visualización de las unidades geológicas obtenidas a partir de un estudio geológico realizado en Úmbita, Boyacá [4,5]. La representación de la geometría de estas unidades se hace a través del uso de mallas, el objeto de este capítulo.

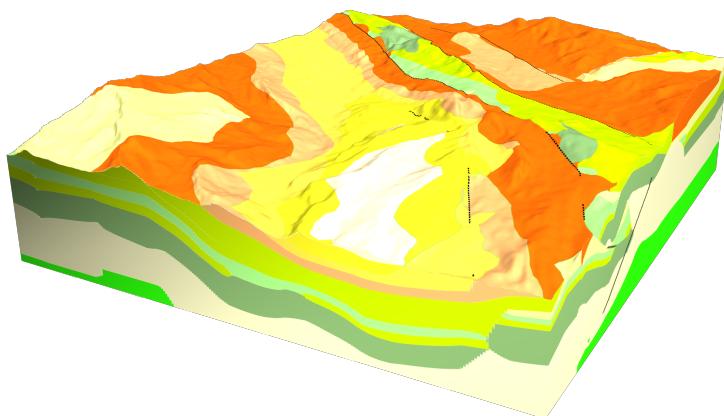


Figura 3.1. Visualización de las unidades geológicas obtenidas a partir del estudio presentado en [4]. Cortesía de [Geomodelr, Inc.](#)

Entre las unidades geológicas que se analizaron en este estudio se encuentra un (potencial) acuífero. Lograr describir la geometría del acuífero (u otras unidades geológicas) permitiría planear el desarrollo urbano alrededor. Por ejemplo, permitiría estimar el tamaño del mismo y asimismo la disponibilidad de agua potable en un futuro. La figura 3.2 presenta dos de estas unidades geológicas.

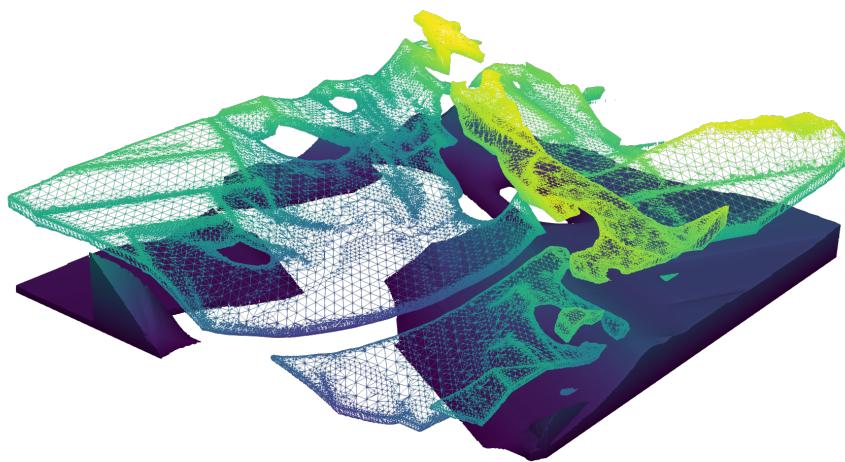


Figura 3.2. Visualización de dos las unidades geológicas obtenidas a partir del estudio presentado en [4]. La unidad superior tiene un volumen de 23 mil millones de m^3 y la inferior de 46 mil millones de m^3 . Datos cortesía de [Geomodelr, Inc.](#)

3.1. Uso de mallas

Las mallas se utilizan para representar geometrías en aplicaciones como:

- Gráficos por computador

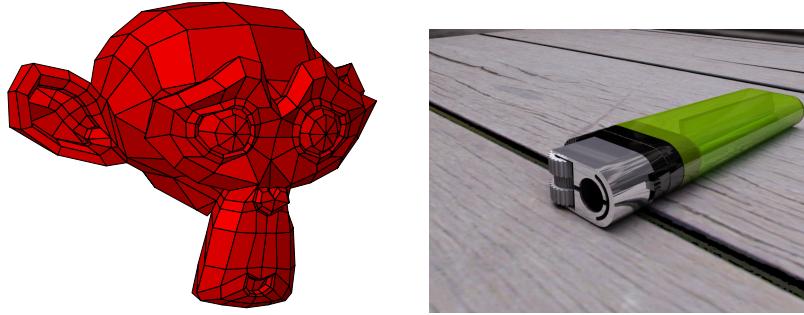


Figura 3.3. Ejemplos de gráficos por computador. A la izquierda se tiene una malla que representa la cabeza de un mono. A la derecha se tiene un rénder fotorrealista. Tomados de [6, 7].

- Sistemas de información geográfica

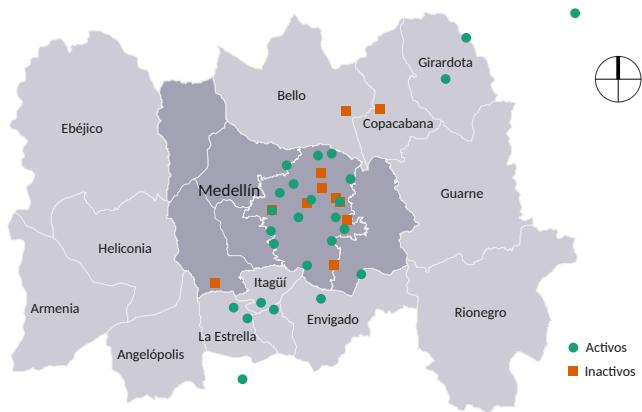


Figura 3.4. Red acelerográfica de Medellín. Los círculos verdes representan estaciones activas. Mientras que los cuadrados naranja son estaciones inactivas. Datos de julio de 2017, tomados de SIATA [3]

- Impresión 3D



Figura 3.5. Ejemplo de impresión 3D: soporte para vasos de café. A la izquierda se tiene el modelo computacional de la pieza a imprimir. A la derecha se presenta la pieza impresa en un vaso de 12 onzas.

- Visualización

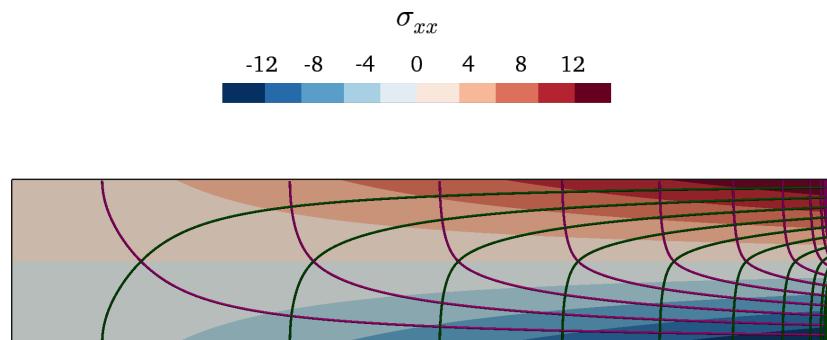


Figura 3.6. Regiones de iso-esfuerzo para una viga en voladizo. Las líneas representan las direcciones principales del tensor.

- Solución de ecuaciones diferenciales

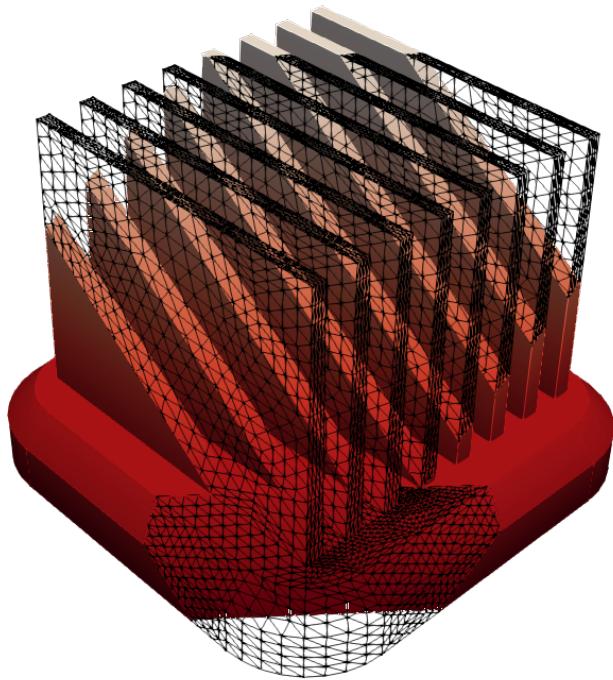


Figura 3.7. Distribución de temperatura en un disipador de calor.

3.2. Concepto de malla

Una malla es una colección de elementos cuya unión representa la forma de un dominio dado. Estos elementos suelen ser polígonos/poliedros simples como triángulos, tetraedros o hexaedros. Los elementos están organizados de tal forma que su intersección se da sólo en puntos, líneas o caras (en 3D).

Una malla puede estar conformada por elementos de diferente geometría. Una malla consta de un número finito de segmentos en una dimensión; segmentos, triángulos y cuadriláteros (quads) en dos dimensiones; y de los elementos anteriores, tetraedros (tets), pentaedros y hexaedros (hexes) en tres

dimensiones.

3.2.1. Mallas estructuradas

Una malla es estructurada si la conectividad es la misma en todos los elementos interiores. Para mallas de cuadriláteros estos implica que en un vértice coinciden 4 elementos y para una de hexaedros coinciden 8. Los siguientes son ejemplos de mallas estructuradas.

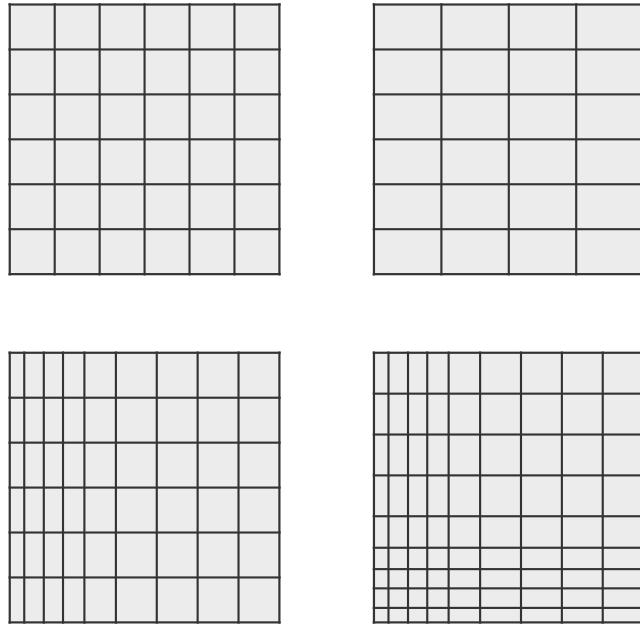


Figura 3.8. Ejemplos de mallas estructuradas.

Debido a que la conectividad de la malla es la misma sólo es necesario almacenar la información de las coordenadas de los vértices. Una característica de las mallas estructuradas es que cada elemento puede indicarse por los índi-

ces (i, j) en dos dimensiones, o (i, j, k) en tres dimensiones, como se muestra a continuación.

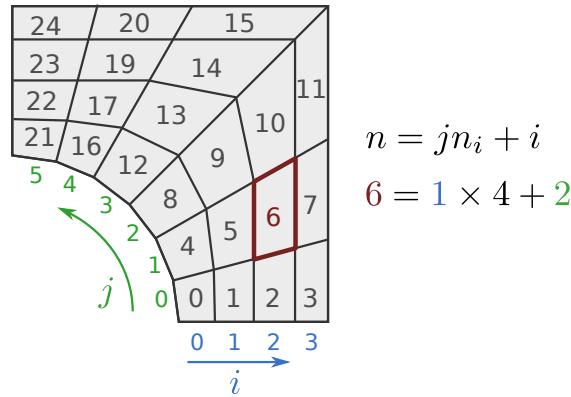


Figura 3.9. Ejemplo de enumeración de malla estructurada en dos dimensiones y su relación con dos índices (i, j) .

3.2.2. Mallas no-estructuradas

Una malla no-estructurada no tiene un patrón definido en la conectividad de los elementos. Este tipo de malla requiere almacenar las coordenadas de los vértices y las conectividades de todos los elementos.

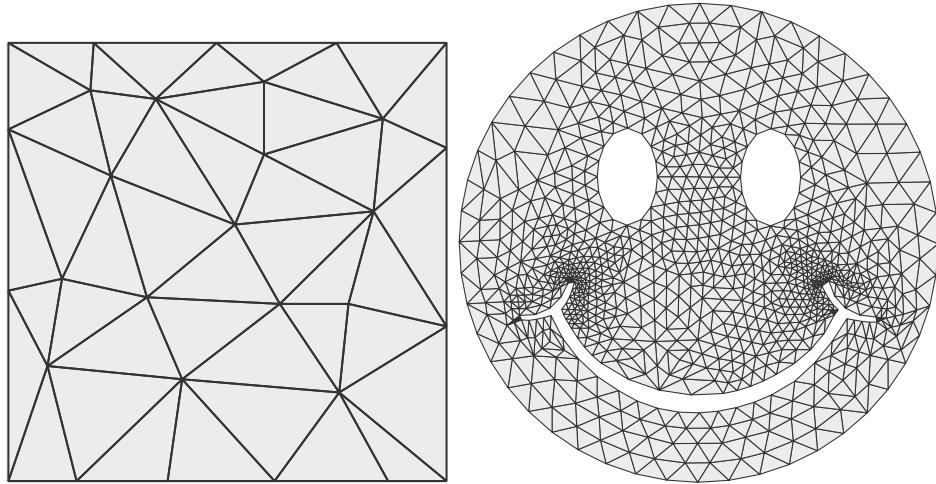


Figura 3.10. Ejemplos de mallas no-estructuradas.

Una ventaja de las mallas no-estructuradas sobre las mallas estructuradas es su versatilidad, ya que permiten representar geometrías más diversas más fácilmente.

3.2.3. Representación de la malla

La forma más común de representar una malla es usando la ubicación de sus nodos y la conectividad entre ellos para formar *elementos*. Veamos un ejemplo para una malla simple dada en la siguiente figura.

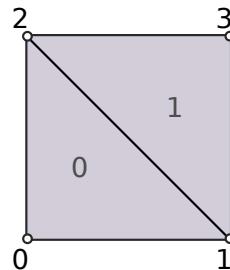


Figura 3.11. Ejemplo de una malla simple formada por dos elementos triangulares y 4 nodos. La colección de elementos en este caso conforman un cuadrado.

Si asumimos que el nodo 0 tiene coordenadas $(0, 0, 0)$ y el nodo 3 coordenadas $(1, 1, 0)$ podemos definir la lista con las coordenadas de los nodos de la siguiente manera

0.0	0.0	0.0
1.0	0.0	0.0
0.0	1.0	0.0
1.0	1.0	0.0

en donde vemos que la primera línea corresponde a las coordenadas del nodo 0 y así sucesivamente para los otros nodos. La lista con la conectividad de los elementos es la siguiente

0	1	2
1	3	2

en donde podemos notar que el elemento 0 está conformado por los puntos (nodos) 0, 1 y 2 y el elemento 1 por los puntos 1, 3 y 2. En ambos casos la numeración de los nodos se hizo considerando una orientación antihoraria

(con la mano derecha). La selección del primer nodo en el elemento es arbitraria, por ejemplo, el elemento 0 podría definirse como (1 2 0), pero no como (0 2 1), ya que este último está orientado en sentido horario.

3.3. Creación de mallas

Cuando tenemos geometrías sencillas podemos crear fácilmente mallas estructuradas para ellas, por ejemplo, usando Python. Sin embargo, el proceso se complica bastante cuando queremos crear mallas para geometrías arbitrarias o mallas no-estructuradas. Por esta razón se requiere de programas externos para la creación de mallas.

Veamos cómo visualizar un campo escalar dado por la función

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

en una sección de una elipse. Utilizando la función `meshgrid` de `numpy` podemos crear una rejilla definida por dos arreglos de números (coordenadas).

A continuación se presenta un bloque de código que permite generar la siguiente visualización.

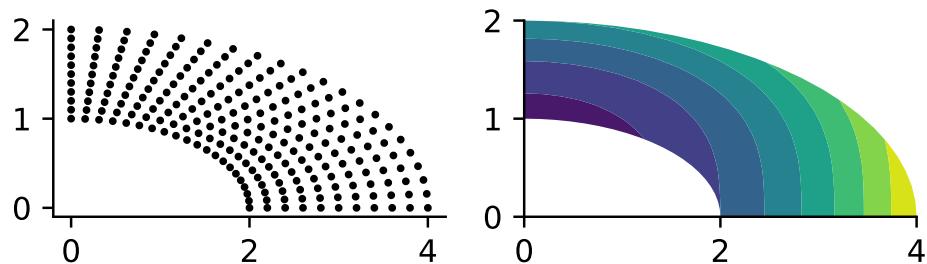


Figura 3.12. Ejemplo de una malla estructurada generada a partir de una parametrización de una sección de una elipse. No se muestra la conectividad, pero esta puede inferirse fácilmente.

```

import numpy as np
import matplotlib.pyplot as plt

rad = np.linspace(1, 2, 11)
angle = np.linspace(0, np.pi/2, 21)
rad, angle = np.meshgrid(rad, angle)

a = 2.0 # semieje mayor
b = 1.0 # semieje menor
x = a * rad * np.cos(angle)
y = b * rad * np.sin(angle)
campo = x**2 + y**3

plt.figure(figsize=(5, 2.5))
plt.subplot(1, 2, 1)
plt.plot(x, y, color="black", marker=".", linewidth=0)
plt.axis("image")
plt.subplot(1, 2, 2)
plt.contourf(x, y, campo)
plt.axis("image")
plt.show()

```

Para la creación de mallas en geometrías más complejas suelen utilizarse programas especializados. Algunos de estos son:

- Gmsh
- Tetgen
- Triangle
- GiD

Nosotros nos centraremos en Gmsh¹.

¹Gmsh es un software libre para la generación de mallas 2D y 3D para simulaciones de elementos finitos [8].

La geometría en Gmsh se crea usando objetos geométricos que forman una jerarquía de la siguiente manera:

- **Puntos:** definidos por sus coordenadas.
- **Líneas:** definidos por los puntos extremos en el caso de líneas rectas. Y definidos por el punto inicial, centro y punto final en el caso de arcos de circunferencia.
- **Contornos (loops):** definidos por una unión de segmentos.
- **Superficies:** definidas por un contorno.
- **Volúmenes:** definidos por las superficies que forman la frontera.

La siguiente sección presenta estos conceptos con mayor detalle a través de un ejemplo.

3.4. Ejemplo de creación de geometría y malla con Gmsh

En este sección vamos a crear la geometría y malla para una sección transversal de un cilindro con un agujero coaxial. La geometría constará de dos superficies, cada una con un arco de π radianes. Este documento no pretende ser una introducción al manejo de Gmsh, para ello se sugiere el tutorial de la referencia [9], el tutorial oficial de Gmsh [10] (para el manejo en modo texto) o los *screencasts* oficiales [11] (para el manejo de la interfaz gráfica).

3.4.1. Creación de la geometría

Puntos y líneas

Llamemos r_{in} al radio interno y r_{ext} al radio externo. Estos parámetros podemos incluirlos en el archivo de geometría con

```
// Parametros
rad_int = 2.0; // Radio interior
rad_ext = 4.0; // Redio exterior
```

Los puntos base para la geometría estarían en las coordenadas

$$\begin{aligned} P1 &= (0.0, 0.0, 0.0) \\ P2 &= (-r_{\text{ext}}, 0.0, 0.0) \\ P3 &= (r_{\text{ext}}, 0.0, 0.0) \\ P4 &= (-r_{\text{int}}, 0.0, 0.0) \\ P5 &= (r_{\text{int}}, 0.0, 0.0). \end{aligned}$$

Que en el archivo de entrada serían

```
// Puntos
Point(1) = {0.0, 0.0, 0, 1.0};
Point(2) = {-rad_ext, 0.0, 0, 1.0};
Point(3) = { rad_ext, 0.0, 0, 1.0};
Point(4) = {-rad_int, 0.0, 0, 1.0};
Point(5) = { rad_int, 0.0, 0, 1.0};
```

Los arcos de circunferencia se definen mediante la terna (Punto inicial, Centro, Punto final). El arco subtendido debe ser menor o igual a π radianes.

Los puntos y líneas pueden verse en la siguiente figura

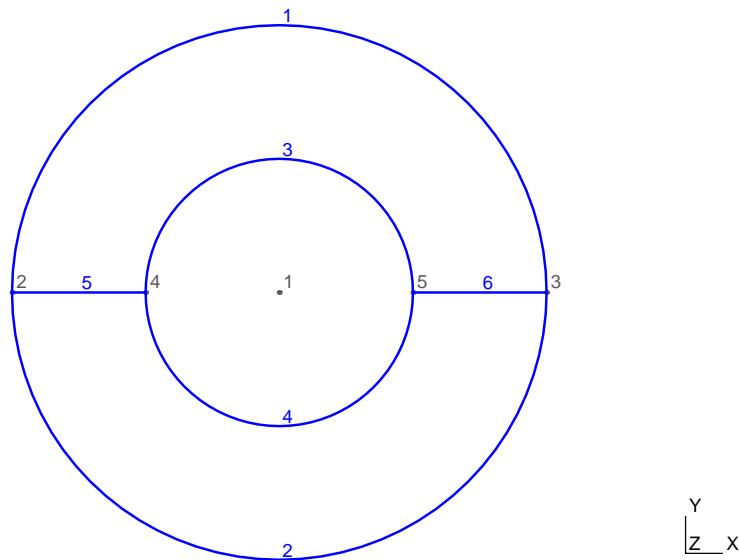


Figura 3.13. Líneas y puntos para la geometría. Los puntos se representan en gris y las líneas en azul

La definición de las líneas sería la siguiente

```
// Lineas
Circle(1) = {3, 1, 2};
Circle(2) = {2, 1, 3};
Circle(3) = {5, 1, 4};
Circle(4) = {4, 1, 5};
Line(5) = {2, 4};
Line(6) = {5, 3};
```

Contornos y superficies

Los contornos son conjuntos de líneas que forman una figura cerrada. Por convención, elegimos el sentido antihorario como el sentido positivo. En este caso, los contornos o bucles² se definen como muestra la siguiente figura.

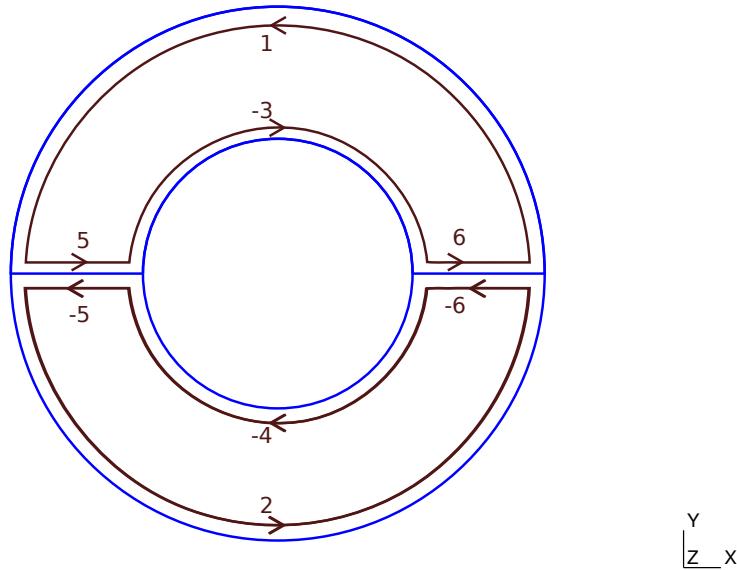


Figura 3.14. Contornos de la geometría.

Creamos, entonces, los contornos y luego las superficies correspondientes a cada contorno como a continuación

```
// Bucle y superficies
Line Loop(7) = {1, 5, -3, 6};
Plane Surface(8) = {7};
```

²En inglés se denominan *loops*.

```
Line Loop(9) = {2, -6, -4, -5};
Plane Surface(10) = {9};
```

Grupos físicos

En Gmsh, los grupos físicos se refieren a conjuntos de entidades geométricas –puntos, líneas, superficies o volúmenes–. Estos conjuntos, o grupos, pueden usarse para denotar una región de un dominio que posee las mismas características, por ejemplo: un mismo material, o una condición de carga o empotramiento. En este caso vamos a definir dos líneas físicas, correspondientes con los radios interno y externo y una superficie física.

En el archivo de entrada sería

```
// Grupos físicos
Physical Line(11) = {1, 2};
Physical Line(12) = {3, 4};
Physical Surface(13) = {8, 10};
```

3.4.2. Parámetros de malla en el archivo de entrada

Adicionalmente, Gmsh permite definir ciertos parámetros que determinan la forma en la que se escribe la malla. En este caso, queremos hacer una subdivisión de los arcos y líneas radiales, equivalente a una rejilla en coordenadas cartesianas. Esto lo logramos especificando cuántas divisiones queremos para las líneas que representan arcos y cuántas para las líneas radiales.

Eso se logra con

```
// Subdivision de lineas
ndiv_arco = 44; // Subdivisiones de los arcos
ndiv_rad = 12; // Subdivisiones en la direccion radial
Transfinite Line {5, 6} = ndiv_rad Using Progression 1;
Transfinite Line {1, 3, 4, 2} = ndiv_arco Using Progression 1;
```

En el caso de la subdivisión de la superficie usamos

```
// Subdivision superficies
Transfinite Surface {8};
Transfinite Surface {10};
```

Si, además, queremos que la malla use cuadriláteros en lugar de triángulos, podemos indicar en el archivo de entrada que queremos unir los triángulos para formar cuadriláteros.

Esto se hace a usando

```
// Recombinar triangulos en cuadrilateros
Recombine Surface {8, 10};
```

3.4.3. Archivo de entrada completo

El archivo .geo completo es el siguiente

```
/*
Ejemplo de creacion de malla para un anillo
```

```
*/  
  
// -- Definicion geometria --  
  
// Parametros  
rad_int = 2.0; // Radio interior  
rad_ext = 4.0; // Redio exterior  
  
// Puntos  
Point(1) = {0.0, 0.0, 0, 1.0};  
Point(2) = {-rad_ext, 0.0, 0, 1.0};  
Point(3) = { rad_ext, 0.0, 0, 1.0};  
Point(4) = {-rad_int, 0.0, 0, 1.0};  
Point(5) = { rad_int, 0.0, 0, 1.0};  
  
// Lineas  
Circle(1) = {3, 1, 2};  
Circle(2) = {2, 1, 3};  
Circle(3) = {5, 1, 4};  
Circle(4) = {4, 1, 5};  
Line(5) = {2, 4};  
Line(6) = {5, 3};  
  
// Bucles y superficies  
Line Loop(7) = {1, 5, -3, 6};  
Plane Surface(8) = {7};  
Line Loop(9) = {2, -6, -4, -5};  
Plane Surface(10) = {9};  
  
// Grupos fisicos  
Physical Line(11) = {1, 2};  
Physical Line(12) = {3, 4};  
Physical Surface(13) = {8, 10};  
  
// -- Parametros de malla --  
  
// Subdivision de lineas
```

```
ndiv_arco = 44; // Subdivisiones de los arcos
ndiv_rad = 12; // Subdivisiones en la direccion radial
Transfinite Line {5, 6} = ndiv_rad Using Progression 1;
Transfinite Line {1, 3, 4, 2} = ndiv_arco Using Progression 1;

// Subdivision superficies
Transfinite Surface {8};
Transfinite Surface {10};

// Recombinar triangulos en cuadrilateros
Recombine Surface {8, 10};
```

Para obtener la malla en la interfaz gráfica vamos a **Mesh/2D** o **Malla/2D**, si está en español. Desde una terminal podemos usar

```
gmsh anillo.geo -2 -o anillo.msh
```

En ambos casos, deberíamos obtener la malla deseada. Esta, debe lucir como la siguiente figura.

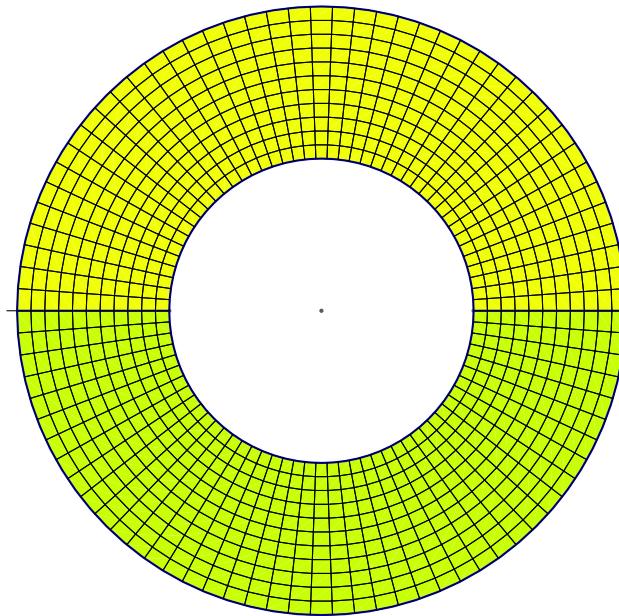


Figura 3.15. Malla para la geometría descrita.

3.5. Lectura de mallas de Gmsh desde Python

Supongamos que tenemos un cuadrado de lado 1, dado por el siguiente archivo de Gmsh.

```
// Cuadrado simple
Point(1) = {0, 0, 0, 1.0};
Point(2) = {1, 0, 0, 1.0};
Point(3) = {1, 1, 0, 1.0};
Point(4) = {0, 1, 0, 1.0};
```

```
Line(1) = {1, 2};  
Line(2) = {2, 3};  
Line(3) = {3, 4};  
Line(4) = {4, 1};  
Line Loop(1) = {1, 2, 3, 4};  
Plane Surface(1) = {1};
```

Si mallamos este geometría en Gmsh, obtendríamos lo siguiente.

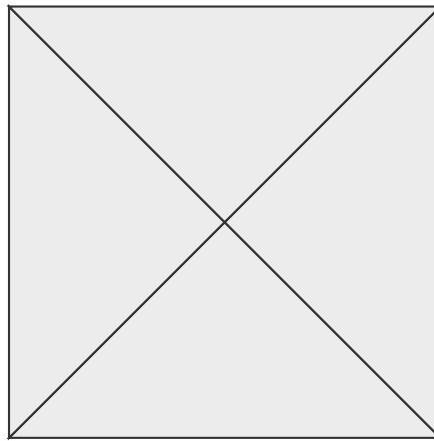


Figura 3.16. Malla para la geometría descrita.

El archivo (.msh) correspondiente a este malla es el siguiente.

```
1 $MeshFormat  
2 2.2 0 8  
3 $EndMeshFormat  
4 $Nodes  
5 5  
6 1 0 0 0  
7 2 1 0 0  
8 3 1 1 0
```

```

9 4 0 1 0
10 5 0.5 0.5 0
11 $EndNodes
12 $Elements
13 12
14 1 15 2 0 1 1
15 2 15 2 0 2 2
16 3 15 2 0 3 3
17 4 15 2 0 4 4
18 5 1 2 0 1 1 2
19 6 1 2 0 2 2 3
20 7 1 2 0 3 3 4
21 8 1 2 0 4 4 1
22 9 2 2 0 1 1 2 5
23 10 2 2 0 1 1 5 4
24 11 2 2 0 1 2 3 5
25 12 2 2 0 1 3 4 5
26 $EndElements

```

Podemos ver que los nodos están definidos entre la línea 5 y la línea 10. Y los elementos entre las líneas 14 y 25. Particularmente, los elementos triangulares³ están definidos entre las líneas 22 y 25. La primera columna en la sección de elementos (\$Elements)⁴ indica el número del elemento, las siguientes indican los grupos a los cuales pertenece, y a partir de la sexta columna tenemos la conectividad de los elementos.

Realizar un script de Python que lea este tipo de archivos no es una tarea difícil, sin embargo, sí es una tarea tediosa. El paquete `meshio`⁵ nos permite leer estos archivos de forma simple.

El siguiente bloque de código leer este archivo.

³Los puntos y las líneas también se consideran elementos y por esto aparecen en el archivo, a menos que se especifique lo contrario.

⁴Esta información se encuentra en la documentación de Gmsh, disponible en: <http://gmsh.info/doc/texinfo/gmsh.html#MSH-file-format-version-2>.

⁵Disponible en: <https://github.com/nschloe/meshio>.

```
import meshio
mesh = meshio.read("cuadrado.msh")
```

Y luego podemos acceder a la información de la malla como se muestra en el siguiente bloque de código.

```
points = mesh.points
cells = mesh.cells
point_data = mesh.point_data
cell_data = mesh.cell_data
```

Capítulo 4

Introducción a los Elementos Finitos

La figura 4.1 presenta el primer modo de pandeo para un pilón del puente Chirajara.



Figura 4.1. Primer modo de pandeo para un pilón del puente Chirajara. Se presenta la configuración deformada y la configuración original como referencia.

4.1. Introducción

El método de los elementos finitos es una técnica numérica para la solución de problemas de valores en la frontera propuesta inicialmente a finales de los 50 y principios de los 60 [12, 13]. Desde entonces, ha experimentado un fuerte desarrollo tecnológico y académico, plasmado en una gran cantidad de paquetes comerciales y de aplicaciones en diferentes áreas de ciencia e ingeniería. Paralelamente, se han escrito una gran cantidad de textos cubriendo aspectos teóricos relativos a la formulación y a la implementación del método [14–17]. En la actualidad el método de los elementos finitos puede considerarse como la herramienta de simulación numérica más poderosa y versátil para el estudio de problemas de ingeniería.

Desde el punto de vista teórico el método se basa en la solución de la forma débil del problema de valores en la frontera sobre una discretización del dominio en elementos finitos en cada uno de los cuales se expresa la solución usando interpolación a partir de valores determinados para unos cuantos puntos o nodos. La contribución de cada elemento finito del dominio es posteriormente sumada usando leyes gobernantes entre las diferentes variables del problema. En el caso mecánico la forma débil del problema de valores en la frontera corresponde al principio de los desplazamientos virtuales y el ensamblaje o consideración de todos los elementos se construye a partir de condiciones de equilibrio y compatibilidad de desplazamientos.

En ingeniería civil el método de elementos finitos es aplicable en la simulación de problemas relativos a áreas como mecánica de fluidos, análisis dinámico de estructuras, mecánica de suelos, análisis de presas, o estructuras de contención. Algunos programas comerciales, específicamente desarrollados para uso en ingeniería civil son: Plaxis¹, SAP2000², OpenSees³.

A pesar del gran desarrollo del método y de su amplio uso en oficinas de ingeniería, especialmente en países desarrollados, este no es impartido en los currículos de ingeniería civil en la mayoría de universidades colombianas. En este capítulo se hace uso de un programa por elementos finitos desarrollado en Python y dirigido al análisis de sólidos elásticos en condiciones de tensión plana y deformación plana. El programa ha sido desarrollado principalmente con fines educativos en los cursos IC0285 Modelación Computacional y IC0602 Introducción al Método de los Elementos Finitos de la Universidad EAFIT. El software es libre y de código abierto⁴. Paquetes similares, desarrollados en Python y dirigidos a uso educativo pueden identificarse en las herramientas SfePy [18], PyFEM [19] and FEniCS [20].

¹<http://www.plaxis.nl/plaxis2d/>

²<https://www.csiamerica.com/products/sap2000>

³http://opensees.berkeley.edu/wiki/index.php/Main_Page

⁴La licencia es MIT y puede descargarse del siguiente enlace: <https://github.com/AppliedMechanics-EAFIT/SolidsPy>.

4.1.1. Aplicaciones de los elementos finitos

El rango de aplicaciones de los elementos finitos es muy diverso, para brindar una idea de su versatilidad se presenta la siguiente lista [21]:

- análisis térmico y de esfuerzos en partes industriales como circuitos integrado, dispositivos electrónicos, válvulas, tuberías, tanques, motores de autos y aviones;
- análisis sísmico de presas, plantas eléctricas, ciudades y edificios altos;
- análisis de choques de carros, trenes y aviones;
- análisis del flujo de contaminantes y sistemas de ventilación;
- análisis electromagnético de antenas, transistores y aviones;
- análisis de procedimientos quirúrgicos como cirugías plásticas, reconstrucción de quijadas, corrección de escoliosis y muchas otras.

El uso de análisis por elementos finitos de estructuras ha reducido los ciclos de diseño considerablemente y mejorado la calidad de los productos en general.

4.2. Concepto intuitivo de rigidez

Considere los 2 modelos mostrados en la siguiente figura. El primero corresponde a un resorte de constante k el cual tras ser sometido a la acción de una fuerza externa F experimenta un cambio en su longitud igual a δ . El mismo problema se ilustra en el modelo abstracto de un sólido representado por la ‘papa’ y en el cual la línea continua describe la forma de la ‘papa’ antes de aplicar la fuerza externa F , mientras que la línea punteada describe la forma de la ‘papa’ tras aplicar la fuerza. Los apoyos esquemáticos (de triángulo y rodillo) que se muestran en la figura 4.2 indican que la ‘papa’ se podrá desplazar como un cuerpo rígido.

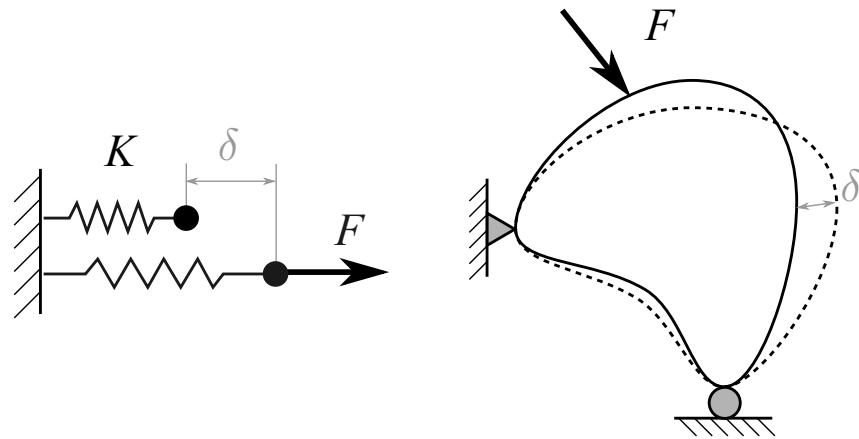


Figura 4.2. Esquema del concepto de rigidez, se hace una analogía entre la deformación de un resorte de constante k , y la deformación que sufre un cuerpo ante cierta carga.

En el resorte se satisface la relación:

$$F = k\delta$$

y en la cual el coeficiente de rigidez k representa la fuerza necesaria para producir un desplazamiento unitario $\delta = 1.0$.

En el caso de la ‘papa’, el problema es más complejo ya que existen muchas direcciones posibles y puntos en los cuales aplicar las fuerzas y en los cuales medir los desplazamientos. ¿Cómo podemos cuantificar la rigidez para este caso?

Sea δ_i el desplazamiento en un punto cualquiera medido en una dirección i y sea F_j la fuerza aplicada en la dirección j también en un punto arbitrario de la ‘papa’. En este caso siguiendo la idea del resorte es posible escribir:

$$F_j = k\delta_i .$$

Comparando esta relación con la correspondiente al problema del resorte

se puede concluir que el coeficiente k representa la fuerza necesaria a lo largo de la dirección j para producir un desplazamiento unitario en la dirección i .

Nótese que si se cambia la dirección en la que se aplica la fuerza y la dirección en la que se mide el desplazamiento es esperable que el coeficiente de rigidez también cambie. Por lo tanto un modelo mas general sería:

$$F_j = k_{ji}\delta_i. \quad (4.1)$$

y en el cual los subíndices ij dan cuenta de que el coeficiente de rigidez k_{ij} representa la fuerza necesaria a lo largo de la dirección j para producir un desplazamiento unitario en la dirección i .

4.2.1. ¿Cómo describir el sólido?

El modelo propuesto para el caso del sólido es físicamente correcto pero operativamente existen infinitas direcciones posibles i y j y surge entonces la pregunta: **¿cómo cuantificar la rigidez para el caso de la ‘papa’?**

Consideremos algunas posibles soluciones:

1. Solución discreta: Aproximar el sólido mediante un sistema finito de partículas conectadas por resortes.
2. Solución del continuo: Estudiar el comportamiento de un elemento diferencial generando un sistema de ecuaciones diferenciales (en las variables u_i , ϵ_{ij} y σ_{ij}).
3. Solución numérica: Combinar 1 y 2 para hacer una aproximación discreta del continuo (métodos por elementos finitos). La comparación entre las soluciones 1 y 2 se describe en la siguiente tabla:

Planteamiento del continuo	Planteamiento discreto
Número infinito de elementos diferenciales por resortes.	Número finito de partículas conectadas por resortes.
Desplazamientos, deformaciones y tensiones.	Desplazamientos y fuerzas.

4.2.2. Elementos finitos (aproximando el continuo)

La figura muestra nuevamente el problema esquemático de la ‘papa’. Esta ha sido dividida en un número **finito** de triángulos conectados en las esquinas. Cada triángulo esta definido por 3 puntos correspondientes a sus vértices. Denominemos un triángulo típico como Ω_e . Asuma que cada vértice del triángulo típico Ω_e se comporta como una partícula y que esta puede experimentar desplazamientos (y fuerzas) en la dirección horizontal y vertical. Asuma además que para el triángulo típico las 3 partículas que lo definen están conectadas por resortes.

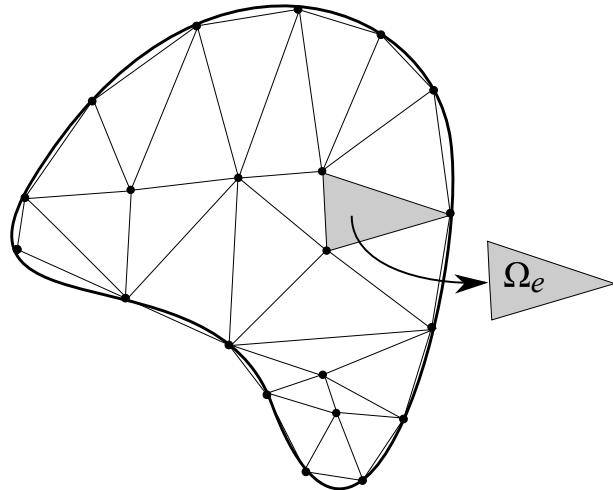


Figura 4.3. Esquema de la discretización del dominio arbitrario en un conjunto finito de elementos triangulares.

4.3. Relación fuerza-desplazamiento para todo el sistema

Se tiene entonces que para un triángulo típico Ω_e es posible relacionar las 6 fuerzas con los 6 desplazamientos mediante una ecuación general como:

$$F_j = k_{ji}\delta_i \quad (4.2)$$

y en la cual el término k_{ji} representa 36 coeficientes de rigidez. Si se considera que el índice j puede tomar 6 valores correspondientes a 6 fuerzas y que el índice i puede tomar 6 valores correspondientes a 6 desplazamientos entonces se puede reconocer que se tiene un sistema de 6 ecuaciones el cual podemos escribir matricialmente como:

$$\{F\} = [K]\{u\}. \quad (4.3)$$

De manera similar, es posible anticipar que es posible escribir una relación que considere todos los puntos del dominio. Por ejemplo, si se tienen N puntos en total habrán $2N$ fuerzas y $2N$ desplazamientos. Escribamos la relación entre fuerzas y desplazamientos para todo el sólido (representado por N puntos) como:

$$F_j^G = k_{ji}^G \delta_i. \quad (4.4)$$

En esta relación hemos usado el superíndice G para enfatizar que las fuerzas, desplazamientos y coeficientes de rigidez corresponden a todo el sólido y no a un elemento típico. En forma matricial se tiene:

$$\{F^G\} = [K^G]\{U^G\}. \quad (4.5)$$

El superíndice G también indica que las ecuaciones son válidas para todo el sistema global representado por todos los triángulos en los que ha sido dividido el sólido.

De acuerdo con la discusión anterior, tanto el equilibrio de un triángulo típico Ω_e como el de un resorte pueden formularse mediante una relación fuerza-desplazamiento de la forma:

$$\{F\} = [K]\{u\}. \quad (4.6)$$

En el caso del resorte es sencillo determinar la matriz de rigidez $[K]$.

En el caso del sólido el problema es un poco más elaborado y por el momento nos conformaremos con suponer que esta matriz se encuentra disponible. En cualquier caso, sea que se trata de un sistema de partículas conectadas por resortes, o un sólido representado por triángulos, el problema se resuelve ensamblando la matriz de rigidez global $[K^G]$ y resolviendo el sistema de ecuaciones:

$$\{F^G\} = [K^G]\{U^G\}. \quad (4.7)$$

A continuación se discute con más detalle el problema usando un sistema de partículas unidas por resortes y se implementa su solución en el computador.

4.3.1. Sistema partícula-resorte

El siguiente ejemplo permite entender desde un punto de vista físico el proceso de solución del problema de varias partículas interactuando a través de un sistema de resortes como el que se muestra en la figura 4.4.

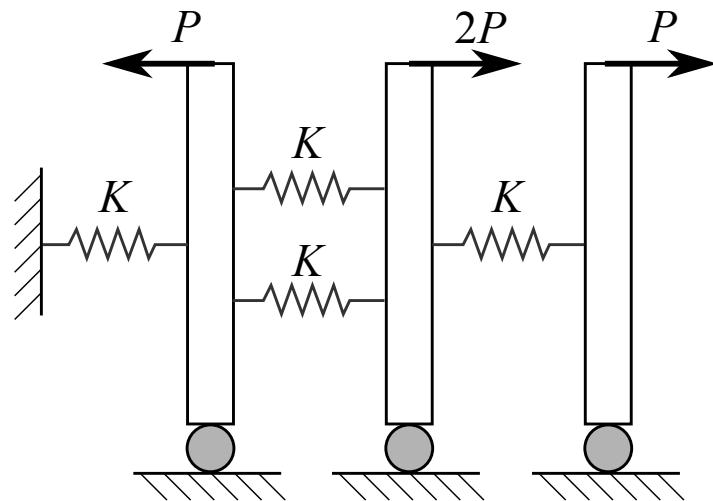


Figura 4.4. Sistema de masas y resortes acoplados.

El sistema de varias partículas se encuentra conectado por resortes con coeficiente de rigidez k . El sistema se encuentra sometido a diferentes fuerzas externas (representadas en múltiplos de P). Por efectos de visualización las partículas se representan como carritos rectangulares.

Se requiere:

1. Plantear las ecuaciones de equilibrio para una partícula típica mostrada en la figura ???. Asuma que la partícula esta conectada a un resorte i por la izquierda y a un resorte $i + 1$ por la derecha de manera que el diagrama de cuerpo libre para la partícula es como el que se muestra en la figura 4.5.

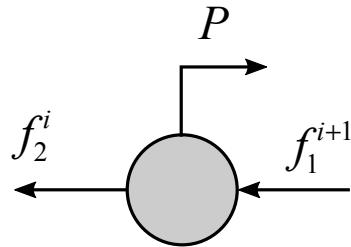


Figura 4.5. Diagrama de cuerpo libre para una partícula típica.

2. Plantear las relaciones fuerza-desplazamiento para el resorte típico mostrado en la figura 4.6. En esta los subíndices 1 y 2 hacen referencia a las fuerzas y desplazamientos de los puntos 1 y 2 que definen el resorte.

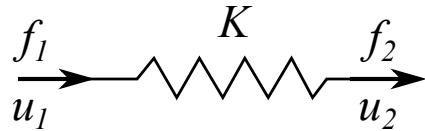


Figura 4.6. Diagrama de las relaciones fuerza-desplazamiento en un resorte típico.

3. Enumerar las partículas y resortes del sistema. (Asumir el empotramiento como una partícula).
4. Escribir las ecuaciones de equilibrio para cada una de las 4 partículas.
5. Usando las relaciones fuerza-desplazamiento de los diferentes resortes re-escribir las ecuaciones del numeral 4.

4.4. Algoritmo de solución

En la solución del problema anterior se usaron planteamientos mecánicos para formular las ecuaciones de equilibrio del sistema de partículas. Consideremos ahora su solución de una manera sistemática de tal forma que el problema se pueda codificar en un programa general de análisis estructural.

4.4.1. Equilibrio de los resortes

Considere el siguiente sistema de 3 partículas, las cuales están conectadas por resortes i e $i+1$ con coeficientes de rigidez K^i y K^{i+1} . De manera similar, las partículas tienen asignados los nombres $j-1$, j y $j+1$ y sus respectivos desplazamientos se denotan por u_{j-1} , u_j y u_{j+1} .

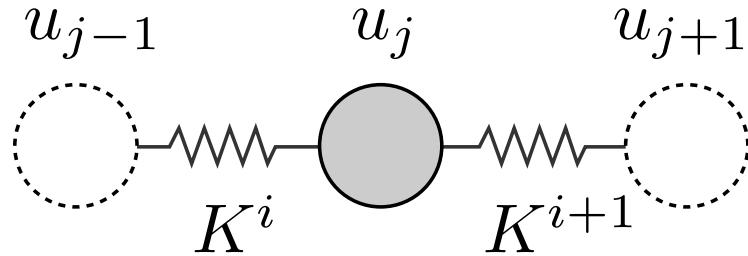


Figura 4.7. Esquema de sistema de 3 partículas conectadas por resortes.

Escribamos las ecuaciones de equilibrio para los resortes i e $i+1$ en términos de los desplazamientos u_{j-1} , u_j y u_{j+1} como:

$$\begin{Bmatrix} f_1^i \\ f_2^i \end{Bmatrix} = \begin{bmatrix} k_{11}^i & k_{12}^i \\ k_{21}^i & k_{22}^i \end{bmatrix} \begin{Bmatrix} u_{j-1} \\ u_j \end{Bmatrix},$$

y

$$\begin{Bmatrix} f_1^{i+1} \\ f_2^{i+1} \end{Bmatrix} = \begin{bmatrix} k_{11}^{i+1} & k_{12}^{i+1} \\ k_{21}^{i+1} & k_{22}^{i+1} \end{bmatrix} \begin{Bmatrix} u_j \\ u_{j+1} \end{Bmatrix}.$$

Note que hemos usado notación fila-columna en los índices para los coeficientes de rigidez para simplificar su implementación en el computador.

4.4.2. Equilibrio de una partícula

Asumiendo que la partícula j se encuentra sometida a una carga externa P , se tiene, de su diagrama de cuerpo libre, que:

$$k_{21}^i u_{j-1} + (k_{22}^i + k_{11}^{i+1}) u_j + k_{12}^{i+1} u_{j+1} = P_j.$$

Procediendo de manera similar para las partículas $j-1$ y $j+1$ y considerando la contribución de los resortes K^i y K^{i+1} obtenemos el siguiente bloque del sistema completo de ecuaciones:

$$\begin{bmatrix} k_{11}^i & k_{12}^i & 0 \\ k_{21}^i & k_{22}^i + k_{11}^{i+1} & k_{12}^{i+1} \\ 0 & k_{21}^{i+1} & k_{22}^{i+1} \end{bmatrix}.$$

Al considerar el sistema completo de partículas y resortes se obtiene un sistema de ecuaciones lineales de la forma general:

$$[K_G] \{U_G\} = \{F_G\}.$$

En este sistema cada ecuación representa el equilibrio de una partícula.

4.4.3. Ensamblaje

La construcción de las matrices globales que describen el equilibrio de cada masa (o partícula) en el sistema puede programarse en el computador mediante un algoritmo que considere el aporte (en términos de fuerzas) de cada elemento (o resorte).

A este proceso se le denomina como **Ensamblaje** de las ecuaciones globales. La operación de ensamblaje puede realizarse después de identificar la conexión entre los (desplazamientos o) grados de libertad globales y los (desplazamientos o) grados de libertad locales mediante una matriz que almacena en cada fila los identificadores de los grados de libertad globales asociados a cada elemento.

Por ejemplo, en el sistema de 3-párticulas los resortes i y $i + 1$ tienen como desplazamientos de sus extremos los denotados como u_{j-1} y u_j y u_j y u_{j+1} . Dichos índices contienen toda la información necesaria para realizar el ensamblaje. La matriz que almacena los índices globales para todos los elementos en el modelo es la matriz DME dada por:

$$DME = \begin{bmatrix} j-1 & j \\ j & j+1 \end{bmatrix}.$$

Nótese que en esta matriz los datos de la primera fila mostrada son $j - 1$ y j , los cuales corresponden a los identificadores de los grados de libertad asociados con el resorte i . De la misma manera los datos de la segunda fila mostrada son j y $j + 1$ los cuales corresponden a los identificadores de los grados de libertad asociados con el resorte $i + 1$.

Una vez se tiene disponible la matriz DME , el proceso de ensamblaje se realiza como se describe a continuación:

$$\begin{aligned} K_{j-1,j-1} &\leftarrow K_{j-1,j-1} + k_{11}^i \\ K_{j-1,j} &\leftarrow K_{j-1,j} + k_{12}^i \\ K_{j,j-1} &\leftarrow K_{j,j-1} + k_{21}^i \\ K_{j,j} &\leftarrow K_{j,j} + k_{22}^i \end{aligned}$$

y

$$\begin{aligned} K_{j,j} &\leftarrow K_{j,j} + k_{11}^{i+1} \\ K_{j,j+1} &\leftarrow K_{j,j+1} + k_{12}^{i+1} \\ K_{j+1,j} &\leftarrow K_{j+1,j} + k_{21}^{i+1} \\ K_{j+1,j+1} &\leftarrow K_{j+1,j+1} + k_{22}^{i+1} \end{aligned}$$

Note la conexión entre los índices locales, correspondientes a 1 y 2, y las posiciones en la matriz global, correspondientes a $j - 1$, j y $j + 1$.

4.5. Programa de análisis estructural

De las secciones anteriores se concluye que el problema de equilibrio se resuelve tras ensamblar la contribución de cada resorte (o triángulo) en una matriz de rigidez global. En esta sección discutimos su solución en el computador. Para definir el modelo usaremos archivos de texto en los cuales indicaremos al programa las partículas que lo conforman, los resortes del sistema y su conexión con las partículas, la localización de las cargas externas y las propiedades de los resortes.

Inicialmente se describen las rutinas o funciones que conforman el programa y en la parte final estas son llamadas desde el programa principal.

El programa principal ejecuta los siguientes pasos:

- Lee el modelo.
- Calcula la matriz *DME*.
- Ensambla el sistema global de ecuaciones.
- Resuelve el sistema para determinar los desplazamientos globales *UG*.

Los archivos de texto con los datos de entrada de las partículas, elementos, coeficientes de rigidez y cargas de este problema se presentan a continuación. Se aclara que al momento de ejecutar los códigos que se presentan más adelante, se debe tener en la misma carpeta donde estos se encuentren, otra carpeta llamada *files*, que contenga dichos archivos de texto.

0	0.000	-1
1	1.000	0
2	2.000	0
3	3.000	0

Tabla 4.1. Archivo de texto *sprnodes.txt*

0	5	1	0	1
1	5	1	1	2
2	5	1	1	2
3	5	1	2	3

Tabla 4.2. Archivo de texto *spreles.txt*

1	-1
2	1
3	2

Tabla 4.3. Archivo de texto *sprloads.txt*, asumiendo que $P = 1\text{N}$.

1000.0
1000.0

Tabla 4.4. Archivo de texto *sprmater.txt*, asumiendo que $k = 1000\text{N/m}$.

La rutina *readin()* lee los archivos de entrada que se encuentran en la carpeta *files* y que deben ser almacenados localmente bajo esta misma estructura.

```
def readin():
    nodes = np.loadtxt('files/sprnodes.txt', ndmin=2)
    mats = np.loadtxt('files/sprmater.txt', ndmin=2)
    elements = np.loadtxt('files/spreles.txt', ndmin=2, dtype=np.int)
    loads = np.loadtxt('files/sprloads.txt', ndmin=2)
    return nodes, mats, elements, loads
```

En el archivo de nodos, en el cual se almacena la información de cada partícula, hay un código con valor -1 o 0 indicando si la partícula se encuentra restringida (como en el empotramiento) o libre. Con esa información la rutina *eqcounter()* asigna identificadores de grados de libertad o números de ecuaciones a cada una de las partículas declaradas como libres.

```
def eqcounter(nodes):
    nn = nodes.shape[0]
    IBC = np.zeros((nn, 1), dtype=np.integer)
    neq = 0
    for cont in range(nn):
        IBC[cont] = int(nodes[cont, 2])
        if IBC[cont] == 0:
            IBC[cont] = neq
        neq = neq + 1
    return neq, IBC
```

El numero de ecuación asignado a cada partícula es usado ahora para formar la matriz *DME*. Cada fila de esta matriz contiene los identificadores de los desplazamientos de los extremos del resorte.

```
def DME(nodes, elements):
    nels = elements.shape[0]
    DME_mat = np.zeros((nels, 2), dtype=np.integer)
    neq, IBC = eqcounter(nodes)
    ndof = 2
    nnodes = 2
    for ele in range(nels):
        for node in range(nnodes):
            pos = elements[ele, node + 3]
            DME_mat[ele, node] = IBC[pos]
    return DME_mat, IBC, neq
```

Usando la matriz *DME* es posible ensamblar la matriz global de coeficientes de rigidez en términos de ecuaciones como:

$$K_{j-1,j-1} \leftarrow K_{j-1,j-1} + k_{11}^i.$$

```
def assembly(elements, mats, nodes, neq, DME_mat):
    IELCON = np.zeros([2], dtype=np.integer)
    KG = np.zeros((neq, neq))
```

```

nels = elements.shape[0]
nnodes = 2
ndof = 2
for el in range(nels):
    elcoor = np.zeros((nnodes))
    im = np.int(elements[el, 2])
    par = mats[im]
    for j in range(nnodes):
        IELCON[j] = elements[el, j+3]
        elcoor[j] = nodesIELCON[j], 1]
        kloc = uelspring(par[0])
        dme = DME_mat[el, :ndof]
        for row in range(ndof):
            glob_row = dme[row]
            if glob_row != -1:
                for col in range(ndof):
                    glob_col = dme[col]
                    if glob_col != -1:
                        KG[glob_row, glob_col] = KG[glob_row, glob_col] +
                        kloc[row, col]

return KG

```

La función *uelspring* calcula la matriz de rigidez de un resorte típico, mientras que la función *loadassem* ensambla el vector de cargas externas aplicadas sobre las partículas.

```

def uelspring(kcof):
    """
    Calcula la matriz de rigidez para
    un elemento tipo resorte conformado por 2 nodos.

    Parametros
    -----
    kcof : float
        Coeficiente de rigidez del resorte (>0).

```

```

Retorna
-----
kloc : ndarray
Matriz de coeficientes de rigidez local para
el elemento tipo resorte (2, 2).

"""
kloc = np.array([
    [kcof, -kcof],
    [-kcof, kcof]])
return kloc

def loadasem.loads, IBC, neq, nl):
    """
Ensambla el vector de cargas o de valores conocidos
en el sistema global de ecuaciones.

Parametros
-----
loads : ndarray
Arreglo con las cargas impuestas a las particulas.
IBC : ndarray (int)
Arreglo que almacena el identificador de grado de libertad
a cada particula.
neq : int
Numero de ecuaciones en el sistema.
nl : int
Numero de cargas en el sistema.

Retorna
-----
rhs_glob : ndarray
Arreglo con las cargas impuestas a las particulas,
rhs se refiere a lado derecho (del ingles right-hand-side).

```

```

"""
rhs_glob = np.zeros((neq))
for cont in range(nl):
    il = int.loads[cont, 0])
    ilx = IBC[il]
    if ilx != -1:
        rhs_glob[ilx] = loads[cont, 1]
return rhs_glob

```

4.5.1. Programa principal

```

import numpy as np

nodes, mats, elements, loads = readin()
DME, IBC, neq = DME(nodes, elements)
KG = assembly(elements, mats, nodes, neq, DME)
RHSG = loadasem(loads, IBC, neq, 3)
UG = np.linalg.solve(KG, RHSG)
print(UG)

```

El programa arroja como resultado $[0.002, 0.0035, 0.0045]$ que corresponde a los desplazamientos de los nodos 1,2 y 3.

4.6. Aspectos a tener en cuenta

4.6.1. Unidades

La mayoría de programas de elementos finitos o simulaciones en general no cuentan con un sistema de unidades interno. Por tanto, la responsabilidad de que las unidades sean consistentes es del analista⁵. Dependiendo del problema

⁵ Algunos programas sí tienen un manejo interno de las unidades. Una lista que compara algunas características está disponible en www.feacompare.com.

de interés, un sistema de unidades puede ser más práctico que otro⁶. El tabla 4.5 presenta algunas cantidades y las unidades usadas para 4 sistemas consistentes de unidades.

Cantidad	SI	SI (mm)	EEUU (ft)	EEUU (in)
Longitud	m	mm	ft	in
Fuerza	N	N	lbf	lbf
Masa	kg	tonne (10^3 kg)	slug	lbf s ² /in
Tiempo	s	s	s	s
Esfuerzo	Pa (N/m ²)	MPa (N/mm ²)	lbf/ft ²	psi (lbf/in ²)
Energía	J	mJ (10^{-3})	ft lbf	in lbf
Densidad	kg/m ³	tonne/mm ³	slug/ft ³	lbf s ² /in ⁴

Tabla 4.5. Sistemas de unidades consistentes que pueden usarse en un análisis por elementos finitos.

Se sugiere entonces siempre usar un sistema consistente de unidades. La presencia de las unidades se dará tanto en la geometría (las unidades de las dimensiones de la región de interés) como en los materiales.

4.6.2. Materiales

Para muchos análisis, los materiales se consideran homogéneos (no dependen de la posición) e isotrópicos (no dependen de la dirección). En algunos casos se requiere modelar problemas en donde los materiales no son homogéneos, por ejemplo, concreto reforzado. Esta situación puede modelarse con varios subdominios con las propiedades del material de cada material constituyente.

Los siguientes cuadros presentan los valores del módulo de Young y densidad para algunos materiales comunes en ingeniería.

⁶Una buena práctica es la de llevar las ecuaciones a grupos no dimensionales, ya que esto usualmente reduce la cantidad de variables independientes [22].

Material	SI (Pa)	SI (MPa)	EEUU (lbf/ft ²)	EEUU (psi)
Acero	200×10^9	200×10^3	30×10^6	4.32×10^9
Aluminio	70×10^9	70×10^3	10.2×10^6	1.46×10^9
Concreto	30×10^9	30×10^3	648×10^6	4.50×10^6

Tabla 4.6. Valores de módulo de Young para algunos materiales en diferentes sistemas de unidades.

Material	SI (kg/m ³)	SI (kg/mm ³)	EEUU (slug/ft ³)	EEUU (lbf s ² /in ⁴)
Acero	7800	7.8×10^{-6}	15.2	0.28
Aluminio	2700	2.7×10^{-6}	5.24	0.098
Concreto	2380	2.38×10^{-6}	4.61	0.086

Tabla 4.7. Valores de densidad para algunos materiales en diferentes sistemas de unidades.

4.7. Geometría

4.7.1. Simplificación de la geometría

Toda modelación que se haga de un sistema implica una idealización. Esta idealización incluye una simplificación de la geometría, es decir, despreciar algunos detalles que puedan ser de menor importancia para la estructura. Saber cuáles detalles incluir y cuáles no, no es una tarea trivial y en general dependerá de la aplicación y experiencia del analista.

Un aspecto que facilita la idealización de la geometría es la presencia de simetrías. La figura 4.8 muestra algunos tipos de simetría que son comunes. Hay que tener en cuenta que una simetría en la geometría no necesariamente implica una simetría en la solución del problema, para ello es necesario que las condiciones de carga también sean simétricas.

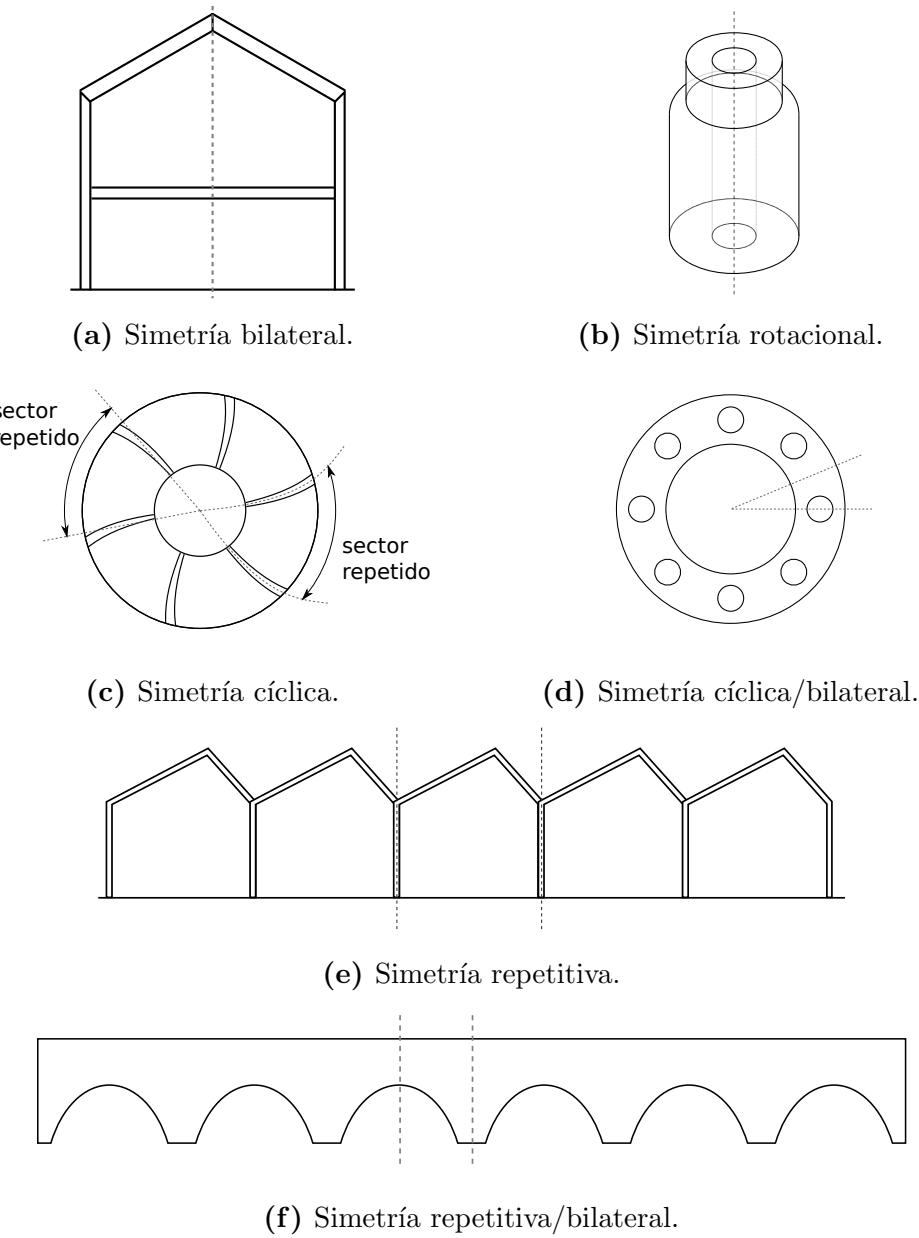


Figura 4.8. Tipos de simetrías en diferentes geometrías [23].

Apéndice A

Códigos

A.1. Búsqueda de raíces

A.1.1. Detección de las raíces 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)
```

A.1.2. Método de bisección para la localización de raíces 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 3. Método de bisección en Python.

A.1.3. Método de Newton-Raphson para la localización de raíces 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 4. Método de Newton-Raphson en Python.

A.2. Integración numérica

A.2.1. 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)))

```

A.2.2. 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))

```

A.2.3. Integración en 2 dimensiones

```

import numpy as np
from scipy.special import roots_legendre # Gauss points and weights

def gauss2d(fun, x0, x1, y0, y1, nx, ny):
    """Gauss quadrature for a rectangle in 2D

    Parameters
    -----
    fun : callable
        Function to integrate.
    x0 : float
        Initial point for the integration interval in x.
    x1 : float
        End point for the integration interval in x.
    y0 : float
        Initial point for the integration interval in y.
    y1 : float
        End point for the integration interval in y.

```

nx : int
Number of points to take in the interval in x.
ny : int
Number of points to take in the interval in y.

Returns

inte : float
Approximation of the integral

""""

```
xi, wi = roots_legendre(nx)
yj, wj = roots_legendre(ny)
inte = 0
hx = 0.5 * (x1 - x0)
hy = 0.5 * (y1 - y0)
xm = 0.5 * (x0 + x1)
ym = 0.5 * (y0 + y1)
for cont_x in range(nx):
    for cont_y in range(ny):
        f = fun(hx * xi[cont_x] + xm, hy * yj[cont_y] + ym)
        inte = inte + hx* hy * f * wi[cont_x] * wj[cont_y]
return inte
```

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

Apéndice B

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
```

10000π

```
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)
```

$$\frac{1}{\sqrt{a}} \tan^{-1} \left(\frac{x}{\sqrt{a}} \right)$$

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 comando *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??

B.1. Ejemplos

B.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)
```

$$\left[\begin{pmatrix} 2, & \frac{1}{2} \end{pmatrix} \right]$$

B.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)
```

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

B.1.3. Graficación

SymPy permite realizar gráficos 2D y 3D

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

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

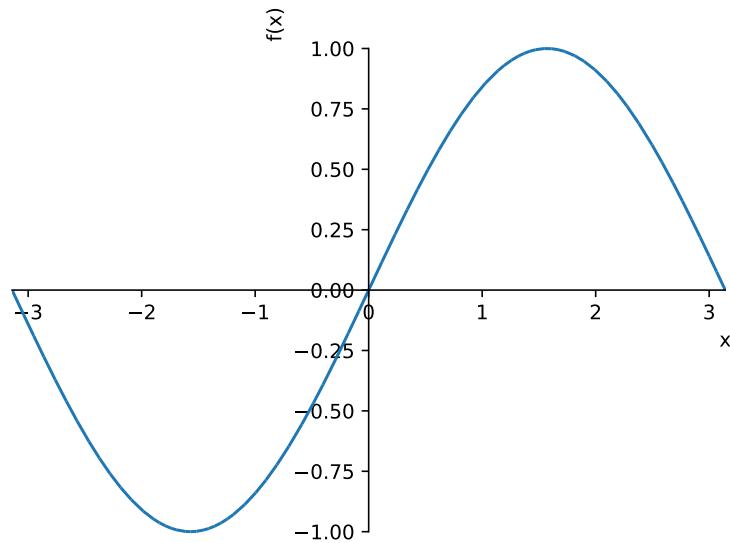


Figura B.1. Ejemplo de gráfico con SymPy.

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

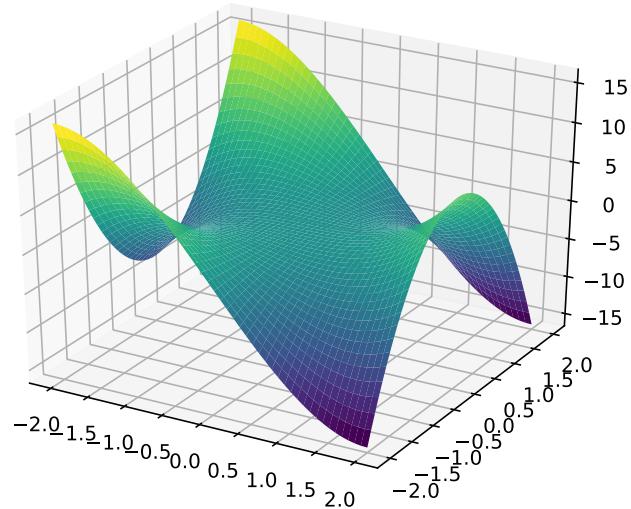


Figura B.2. Ejemplo de gráfico 3D con sympy.

B.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)
```

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

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))
```

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

B.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
```

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

```
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)
```

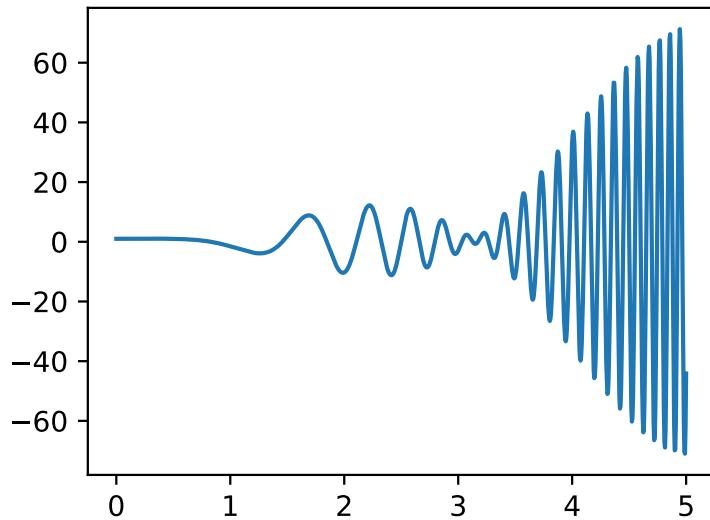


Figura B.3. Gráfico de la función evaluada luego de usar lambdify.

B.3. Ejercicios

1. Calcule el límite

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

2. Resuelva la ecuación diferencial de Bernoulli

$$x \frac{du(x)}{dx} + u(x) - u(x)^2 = 0.$$

B.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

Bibliografía

- [1] Kai Velten. *Mathematical modeling and simulation: introduction for scientists and engineers*. John Wiley & Sons, 2009.
- [2] James Edward Gordon. *Structures: or why things don't fall down*. Da Capo Press, 2003.
- [3] SIATA: Sistema de Alerta Temprana del valle de Aburrá. Estaciones acelerográficas del valle de Aburrá, 2017. Disponible en https://siata.gov.co/gposada/sismicidad/estaciones_acel.csv.
- [4] G. Borda. Modelo hidrogeológico del potencial acuífero de la formación socha inferior, Úmbita, boyacá. Master's thesis, Universidad Nacional de Colombia, 2017.
- [5] D. Giraldo, R. Serrano, D. Álvarez, M. Jaramillo, and G. Borda. Geomodelr, una herramienta novedosa para la creación de modelos hidrogeológicos conceptuales. *Sexto congreso de Hidrogeología*, 2018.
- [6] Wikimedia Commons. File:suzanne.svg — wikimedia commons, the free media repository, 2011. Fecha de acceso: 13 de septiembre de 2018.
- [7] Wikimedia Commons. File:blender-yafray-render-lighter.png — wikimedia commons, the free media repository, 2015. Fecha de acceso: 13 de septiembre de 2018.
- [8] Christophe Geuzaine & Jean-François Remacle. Gmsh: A 3-d finite element mesh generator with built-in pre-& post-processing facilities. *International Journal for Numerical Methods in Engineering*, 79(11), 2009.

- [9] A. & S. L. Mouradian Avdis. A gmsh tutorial, 2012.
- [10] Christophe Geuzaine & Jean-François Remacle. Gmsh tutorial, 2017.
- [11] Christophe Geuzaine & Jean-François Remacle. Gmsh screencasts, 2017.
- [12] R. W. Clough. Finite element stiffness matrices for analysis of plates in bending. *Proceedings of the 1st Conference on Matrix Methods in Structural Mechanics*, 1965.
- [13] MJ Turner. Stiffness and deflection analysis of complex structures. *Journal of the Aeronautical Sciences*, 23(9):805–823, 1956.
- [14] Klaus-Jürgen Bathe. *Finite element procedures*. Klaus-Jürgen Bathe, 2006.
- [15] Thomas JR Hughes. *The finite element method: linear static and dynamic finite element analysis*. Courier Corporation, 2012.
- [16] Junuthula Narasimha Reddy. *An introduction to the finite element method*. McGraw-Hill Education, 4 edition, 2018.
- [17] Olek C. Zienkiewicz and Robert L. Taylor. *The Finite Element Method: Its basis and fundamentals*. Butterworth-Heinemann, 7 edition, 2013.
- [18] Robert Cimrman. Sfepy-write your own fe application. *arXiv preprint arXiv:1404.6391*, 2014.
- [19] Mike A Crisfield, Joris JC Remmers, Clemens V Verhoosel, et al. *Non-linear finite element analysis of solids and structures*. John Wiley & Sons, 2012.
- [20] Anders Logg, Kent-Andre Mardal, and Garth Wells. *Automated solution of differential equations by the finite element method: The FEniCS book*, volume 84. Springer Science & Business Media, 2012.
- [21] Jacob Fish and Ted Belytschko. *A first course in finite elements*, volume 1. John Wiley & Sons New York, 2007.
- [22] Hans Petter Langtangen and Geir K Pedersen. *Scaling of Differential Equations*. Springer, 2016.

- [23] D.R Hose D. Baguley. How to model with finite elements, 1997.
- [24] ASTM D3967-16. Standard test method for splitting tensile strength of intact rock core specimens. *ASTM International*, 2016.