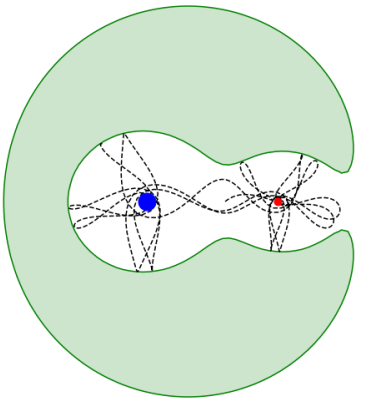
Mecánica Celeste y Analítica

Teoría, problemas y algoritmos



Jorge I. Zuluaga
Profesor titular de Física y Astronomía

Instituto de Física, Facultad de Ciencias Exactas y Naturales
Universidad de Antioquia

8 de octubre de 2019

BORRADOR

Índice general

BORRADOR

BORRADOR

Índice de figuras

BORRADOR

BORRADOR

Capítulo 1

Fundamentos

Resumen

En este capítulo haremos una síntesis práctica de los temas de matemáticas, física y astronomía que necesitaremos para desarrollar la mecánica celeste en el resto del libro. Repasaremos la geometría de las cónicas, (que son la base para describir la trayectorias de cuerpos celestes sometidos a la gavedad newtoniana), en el plano y en el espacio de tres dimensiones. Sintetizaremos, en un lenguaje moderno, los principios y leyes de la mecánica newtoniana, incluyendo la dinámica en sistemas de referencia rotantes. Presentaremos brevemente los fundamentos de la teoría de la relatividad (especial y general), especialmente el concepto de métrica y sus implicaciones cinemáticas básicas. Finalmente repasaremos a algunas conceptos y definiciones de astronomía que serán de relevancia en el texto, la organización y nomenclatura del sistema solar, los sistemas de medida de tiempo y los sistemas de referencia usados para especificar la posición de cuerpos en el cielo y en el espacio interplanetario.

1.1. Vectores y cálculo

En esta sección repasaremos, de manera práctica (y posiblemente poco rigurosa desde el punto de vista matemático), algunos resultados centrales del cálculo infinitesimal y la teoría de ecuaciones diferenciales que serán de utilidad en el resto del libro.

Para quienes conocen bien estos temas, puede servir de motivación para la lectura de esta sección, el hecho de que además de conceptos matemáticos ampliamente conocidos, hemos incluído aquí detalles sobre la **notación matemática**, **definiciones** y **teoremas** que usaremos en el resto del libro; escritos todos en un lenguaje muy propio del texto. Tal vez más interesante es el hecho de que a lo largo de esta sección ilustraremos también algunos de los conceptos claves usando **algoritmos**, con lo que sentarames las bases para todos los desarrollos *computacionales* de los demás capítulos.

Sea que lea esta sección o sea que no lo haga, antes de pasar a los siguientes capítulos intente resolver los problemas al final de este capítulo que están directamente relacionados con los temas de esta sección. Este ejercicio le permitirá valorar mejor las habilidades matemáticas y algorítmicas que tiene antes de comenzar y que serán indispensable en el resto del libro. Tal vez descubra que después todo no es mala idea hacer este repaso.

1.1.1. Conjunto, tuplas y vectores

Hay tres tipos de entidades matemáticas (además de los números reales y las funciones) que usaremos con frecuencia en este capítulo (y en general en todo el libro):

- **Conjuntos**. Muchas veces nos referiremos aquí a conjuntos (no necesariamente ordenados) de entidades que están relacionadas de alguna manera: las coordenadas de un punto en el espacio de fases, un conjunto de funciones, las ecuaciones diferenciales que describen el movimiento de un sistema dinámico, las partículas que interactúan gravitacionalmente en un sistema, etc. Los elementos de la mayoría de los conjuntos usados en este libro estarán numerados. Así por ejemplo, las masas de un sistema de N partículas, $m_0, m_1, \ldots, m_{N-1}$ se representarán como el conjunto:

  $$\{m_i\}_{i=0,1,\ldots,N-1}$$

  Una versión sintética más común de esta notación será $\{m_i\}_N$. En el caso en el que el número de elementos sea claro en el contexto se usará simplemente $\{m_i\}$

  Nota

  Numeración comenzando en cero. En lo sucesivo numeraremos todas las cantidades físicas y matemáticas (partículas, variables auxiliares, componentes de un vector o una matriz, etc.) comenzando en cero, tal y como se acostumbra en programación. Esta elección facilitará la implementación de las fórmulas en algoritmos y programas de computadora. Si bien la numeración comenzando en cero no es muy común en matemáticas o física, existen justificaciones poderosas para su uso, algunas de las cuáles están enumeradas en el documento "Why numbering should start at zero"[a] del maestro de maestros de la programación científica, Edsger Wybe Dijkstra.

  [a] https://www.cs.utexas.edu/users/EWD/transcriptions/EWD08xx/EWD831.html

- **Tuplas**. Las tuplas (pares, tripletas, etc.) son conjuntos ordenados de números reales. Para las tuplas usaremos la notación convencional $(x_0, x_2, \ldots, x_{N-1})$, donde los paréntesis, a diferencia de las llaves de los conjuntos más generales, nos permitirán reconocer el hecho de que el orden de

PRADER

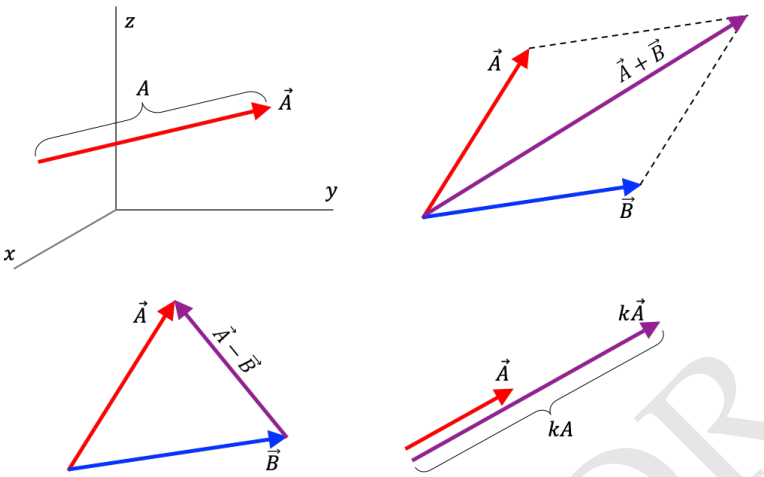


Figura 1.1: *Definición geométrica de vector espacial y de sus operaciones básicas (suma, resta y multiplicación por un escalar). Aunque la resta de $\vec{A} - \vec{B}$ es un caso particular de la suma, es importante aquí familiarizarse con la dirección que tiene este vector (va de la cabeza del sustraendo $\vec{B}$ a la del minuendo $\vec{A}$.)*

los elementos es importante. Las tuplas forman, con el conjunto de los números reales, un *espacio vectorial*. En este espacio se definen las siguientes operaciones básicas:

- Suma:

$$(a_0, a_1, \ldots) + (b_0, b_1, \ldots) \equiv (a_0 + b_0, a_1 + b_1, \ldots) \tag{1.1}$$

- Multiplicación por un escalar:

$$k(a_0, a_1, \ldots) \equiv (ka_0, ka_1, \ldots) \tag{1.2}$$

donde k es un número real.

- **Vectores geométricos (euclidianos)**. Los vectores geométricos o en breve vectores, son *segmentos orientados* en el espacio de tres dimensiones (ver Figura 1.1) que tienen las siguientes propiedades:

  - Se denotarán en este libro como $\vec{A}$ o $\hat{e}$ (este último es un vector unitario) en lugar de usar la notación más común con letras en negrilla.
  - Todo vector tiene: 1) magnitud, A, igual a la longitud (euclidiana) del segmento correspondiente y 2) una dirección en el espacio.
  - Los vectores forman con los números reales, un espacio vectorial con operaciones definidas, geométricamente, como se muestra en la Figura 1.1.

BORRADOR

- Todo vector, por definición, se puede escribir como una combinación lineal de tres vectores de una base ortonormal: $\hat{e}_0, \hat{e}_1, \hat{e}_2$. Los coeficientes de la combinación se conocen como componentes del vector:

$$\vec{A} = A_0\hat{e}_0 + A_1\hat{e}_1 + A_2\hat{e}_2. \tag{1.3}$$

- El espacio de vectores es *isomorfico* a el espacio vectorial de tripletas. Por la misma razón nos referiremos al vector, o bien como la entidad abstracta $\vec{A}$, como su representación en términos de los vectores unitarios de una base ortonormal (ver ítem anterior) o aún mejor, en términos de la tripleta:

$$\vec{A} : (A_0, A_1, A_2)$$

  En este caso usaremos el símbolo ":" en lugar de "=" para dar entender que el vector *no es* una tripleta sino una entidad geométrica más abstracta.

  El isomorfismo implica también, que las componentes de los vectores en las operaciones geométricas definidas en la Figura 1.1, cumplen la Ecs. (1.1) y (1.2).

- Además de la suma y la multiplicación por un escalar, que caracterizan el espacio vectorial, se definen dos productos adicionales:

  - **Producto escalar** o **producto punto**, $\vec{A} \cdot \vec{B}$. El producto escalar se define, a partir los vectores unitarios de la base, como:

$$\hat{e}_i \cdot \hat{e}_j = \delta_{ij},$$

    donde δ_{ij} es el "delta de kroenecker":

$$\delta_{ij} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases} \tag{1.4}$$

    Con esta esta definición y usando la representación de los vectores dada por la Ec. (1.3) puede probarse que:

$$\vec{A} \cdot \vec{B} = A_0B_0 + A_1B_1 + A_2B_2$$

  - **Producto vectorial** o **producto cruz**, $\vec{A} \times \vec{B}$. El producto vectorial se define, a partir los vectores unitarios de la base, como:

$$\hat{e}_i \times \hat{e}_j = \epsilon_{ijk}\hat{e}_k, \tag{1.5}$$

    donde ϵ_{ijk} es el "simbolo de Levi-Civita" ("levi chivita"[1]):

$$\epsilon_{ijk} = \begin{cases} +1 & \text{si } (i,j,k) \text{ es } (0,1,2), (1,2,0) \text{ o } (2,0,1) \\ -1 & \text{si } (i,j,k) \text{ es } (2,1,0), (0,2,1) \text{ o } (1,0,2) \\ 0 & \text{de otro modo } i = j \text{ o } j = k \text{ o } k = i \end{cases} \tag{1.6}$$

[1] https://forvo.com/word/levi-civita/#it

BORRADOR

Al conjunto de vectores unitarios de una base que se definen cumpliendo la Ec. (1.5) se lo llama un *conjunto de vectores de mano derecha*. Con esta esta definición y usando la representación de los vectores dada por la Ec. (1.3) puede probarse que:

$$\begin{aligned}\vec{A}\times\vec{B} &= (A_1B_2 - A_2B_1)\hat{e}_0 + \\ &\quad -(A_0B_2 - A_2B_0)\hat{e}_1 + \\ &\quad (A_0B_1 - A_1B_0)\hat{e}_2\end{aligned} \tag{1.7}$$

Está última expresión es tan elaborada que con frecuencia se usa la regla mnemoténica:

$$\vec{A}\times\vec{B} = \begin{vmatrix} \hat{e}_0 & \hat{e}_1 & \hat{e}_2 \\ A_0 & A_1 & A_2 \\ B_0 & B_1 & B_2 \end{vmatrix} \tag{1.8}$$

Donde $|M|$ es el determinante de la matriz M.

- Otras identidades útiles:
  - Propiedad cíclica del **triple producto escalar**:

$$\vec{A}\cdot(\vec{B}\times\vec{C}) = \vec{C}\cdot(\vec{A}\times\vec{B}) = \vec{B}\cdot(\vec{C}\times\vec{A}) \tag{1.9}$$

  - **Triple producto vectorial**:

$$\vec{A}\times(\vec{B}\times\vec{C}) = (\vec{A}\cdot\vec{C})\vec{B} - (\vec{A}\cdot\vec{B})\vec{C} \tag{1.10}$$

Algoritmos para conjuntos y tuplas

Todos los lenguajes modernos de programación, definen tipos especiales para representar conjuntos y tuplas. En `Python` existen tres tipos de objetos básicos para este propósito: `listas`, `tuplas` y diccionarios. Existen sútiles diferencias entre las listas y las tuplas en `Python` y en general usaremos con más frecuencia las primeras. Para para los algoritmos de este libro, es importante entender las *operaciones* entre listas, que son diferentes a las operaciones en el espacio vectorial de las tuplas matemáticas que definimos antes.

Así, por ejemplo, en el siguiente algorítmo se construye una lista con las componentes del vector de estado de una partícula, "sumando" las listas de las componentes de su vector posición y velocidad:

```
#Lista de componentes del vector posición
r=[1,0,3]
#Lista de componentes del vector velocidad
v=[0,-1,0]
#Lista de componentes del vector de estado
X=r+v
```

```
X = [1, 0, 3, 0, -1, 0]
```

El operador `+`, entre listas y tuplas de `Python` produce la unión de los elementos de las listas.

BORRADOR

Usando este operador se pueden hacer algoritmos prácticos como el que se muestra a continuación:

```
def f(x):
    from math import sin
    y=sin(x)/x
    return y

valores_de_x=[1.0,2.0,3.0]
valores_de_f=[]
for x in valores_de_x:
    valores_de_f+=[f(x)]
```

```
valores de f = [0.8414709848078965, 0.45464871341284085, 0.0470400026866224]
```

aquí, comenzamos con un conjunto vacío, `valores_de_f=[]` y después, dentro de un ciclo, usamos el operador de acumulación `+=` para agreganr elementos al conjunto. Este es un método muy común usado en el lenguaje para construir "tablas de valores", que pueden, por ejemplo usarse para hacer gráficos de funciones.

Algoritmos para vectores

Los vectores forman un "capítulo" en la computación separado de las listas y las tuplas. La razón básica son sus propiedades matemáticas y las operaciones definidas entre ellos. Bibliotecas de rutinas muy completas existen en todos los lenguajes de programación para representar este tipo de entidades matemáticas.

En `Python` y a lo largo de este libro usaremos los objetos y rutinas de los paquetes `NumPy` y `SPICE` para definir y manipular vectores.

El ejemplo abajo muestra como se calcula el ángulo entre dos vectores θ_{AB}, a partir de la interpretación geométrica del producto punto $\vec{A} \cdot \vec{B} = AB \cos\theta_{AB}$ (ver problemas al final del capítulo):

```
#Definimos los vectores
from numpy import array
A=array([1.0,0.0,2.0])
B=array([0.0,1.0,3.0])

#Calculamos el producto escalar y vectorial
from numpy import dot
ApuntoB=dot(A,B)

#El ángulo entre los vectores
from numpy import arccos
from numpy.linalg import norm
anguloAB=arccos(ApuntoB/(norm(A)*norm(B)))
```

```
AnguloAB = 31.948059431330062 grados
```

BORRADOR

Nota

Radianes y grados en los algoritmos. Es importante entender que las funciones trigonométricas inversas como `arccos`, devuelven, en todos los lenguajes de programación, valores de los ángulos en radianes. En el caso anterior, por ejemplo, el valor de la variable `anguloAB` al final del algoritmo en realidad era `0.5575988266995369`. Sin embargo, decidimos mostrar su valor en grados después de multiplicar `anguloAB` por el factor de conversión $\pi/180$ (esta operación no se muestra en el código.) Así lo seguiremos haciendo en el resto del libro. El lector que use los algoritmos no debe olvidar multiplicar por el factor de conversión para reconstruir los resultados mostrados aquí.

Un procedimiento similar, esta vez usando vectores y rutinas de `SPICE` (internamente `SPICE` usa vectores o arreglos de `NumPy`), puede usarse para calcular el triple producto vectorial:

```
from numpy import array
from spiceypy import vdot,vcrss

A=array([2.0,2.0,1.0])
B=array([0.0,-1.0,0.0])
C=array([0.0,0.0,2.0])

AxBxC=vdot(A,B)*C-vdot(A,C)*B
```

```
A x (B x C) = [-0.  2. -4.]
```

También podemos usarlas para verificar la propiedad cíclica del triple producto escalar:

```
from numpy import array
from spiceypy import vnorm,vdot,vcrss
A=array([2.0,2.0,1.0])
B=array([0.0,-1.0,0.0])
C=array([0.0,0.0,2.0])

ABC=vdot(A,vcrss(B,C))
CAB=vdot(C,vcrss(A,B))
BCA=vdot(B,vcrss(C,A))
```

```
A.(BxC) = -4.0
C.(AxB) = -4.0
B.(CxA) = -4.0
```

Con lo que se verifica la identidad (al menos para los vectores escogidos.)

1.1.2. Sistemas de coordenadas

A lo largo de este libro, usaremos los tres sistemas de coordenadas ortogonales *clásicos* (cartesianas, cilíndricas y esféricas, ver Figura 1.2) con algunas convencio-

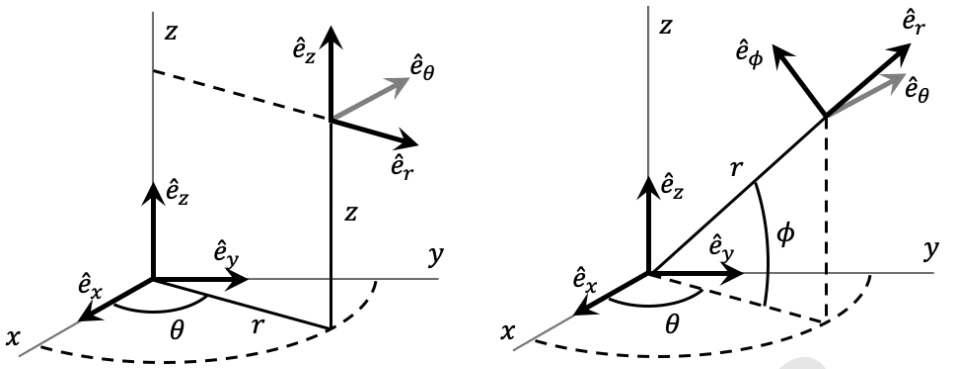


Figura 1.2: *Definción de los sistemas de coordenadas usadas en este texto*

nes más propias de la astronomía y la mecánica celeste que del cálculo.

A continuación, y en especial para clarificar nuestra notación, enumeramos detalladamente las propiedades de cada sistema.

- **Sistema de coordenadas cartesiano**.
  - Coordenadas: $x \in (-\infty, +\infty)$, $y \in (-\infty, +\infty)$, $z \in (-\infty, +\infty)$.
  - Vectores unitarios: $\hat{e}_x$, $\hat{e}_y$, $\hat{e}_z$.
  - Comentarios:
    - En todos los casos la orientación de los ejes obedecerá la *regla de la mano derecha*, es decir, los sistemas cartesianos usados en el texto y cuyos ejes están definidos por el conjunto de vectores unitarios ($\hat{e}_x$, $\hat{e}_y$, $\hat{e}_z$) forman un *conjunto de mano derecha* (ver Ec. 1.5), a saber, en forma explícita:

$$\begin{aligned} \hat{e}_x \times \hat{e}_y &= \hat{e}_z \\ \hat{e}_y \times \hat{e}_z &= \hat{e}_x \\ \hat{e}_z \times \hat{e}_x &= \hat{e}_y \end{aligned} \tag{1.11}$$

- **Sistema de coordenadas cilíndrico** (ver Figura 1.2).
  - Coordenadas: $r \in [0, +\infty)$, $\theta \in [0, 2\pi)$, $z \in (-\infty, +\infty)$.
  - Conversión al sistema de coordenadas cartesianas:

$$\begin{aligned} x &= r\cos\theta \\ y &= r\sin\theta \end{aligned} \tag{1.12}$$

  - Vectores unitarios expresados en el sistema de coordenadas cartesianas:

$$\begin{aligned} \hat{e}_r &= \cos\theta\ \hat{e}_x + \sin\theta\ \hat{e}_y \\ \hat{e}_\theta &= -\sin\theta\ \hat{e}_x + \cos\theta\ \hat{e}_y \\ \hat{e}_z &= \hat{e}_z \end{aligned} \tag{1.13}$$

BORRADOR

- Comentarios:
  - El conjunto de vectores unitarios $(\hat{e}_r, \hat{e}_\theta, \hat{e}_z)$ forman un conjunto de mano derecha tal y como se definió en las Ecs. (1.11).
  - Nótese que, a diferencia de la notación usada generalmente en los textos de cálculo, la coordenada cilíndrica r usa la misma letra que la coordenada esférica r y la magnitud del vector posición (ver siguiente sección). La distinción entre las tres, dependerá del contexto.
  - Usaremos la letra griega θ para denotar el ángulo *acimutal*, a diferencia de la notación convencional que usa esta letra para la coordenada esférica polar (ángulo del vector posición respecto al eje z.)

- **Sistema de coordenadas esférico** (ver Figura 1.2).
  - Coordenadas: $r \in [0, +\infty)$, $\theta \in [0, 2\pi)$, $\phi \in [-\pi/2, +\pi/2]$.
  - Conversión al sistema de coordenadas cartesianas:

$$\begin{aligned} x &= r\cos\phi\cos\theta \\ y &= r\cos\phi\sin\theta \\ z &= r\sin\phi \end{aligned} \tag{1.14}$$

  - Vectores unitarios expresados en el sistema de coordenadas cartesianas:

$$\begin{aligned} \hat{e}_r &= \cos\phi\cos\theta\ \hat{e}_x + \cos\phi\sin\theta\ \hat{e}_y + \sin\phi\ \hat{e}_z \\ \hat{e}_\theta &= -\sin\theta\ \hat{e}_x + \cos\theta\ \hat{e}_y \\ \hat{e}_\phi &= \cos\phi\cos\theta\ \hat{e}_x + \sin\phi\sin\theta\ \hat{e}_y - \sin\phi\ \hat{e}_z \end{aligned} \tag{1.15}$$

  - Comentarios:
    - El conjunto de vectores unitarios $(\hat{e}_r, \hat{e}_\theta, \hat{e}_\phi)$ forman un conjunto de mano derecha tal y como se definió en las Ecs. (1.11).
    - Nótese que, a diferencia de la notación usada generalmente en los textos de cálculo, la coordenada esférica ϕ se medirá respecto al plano $x - y$ (como una *latitud*) en lugar de hacerlo respecto al eje z (como una *colatitud*).

Una interesante página interactiva que permite visualizar mejor la definición de los sistemas de coordenadas y la orientación de los vectores coordenadas puede encontrarse en los siguientes enlaces, tanto para el sistema de coordenadas cilíndrica[2] como para el sistema de coordenadas esféricas[3].

1.1.3. Funciones

Una función es, en términos informales, una regla de correspondencia que asocia los elementos de un conjunto de partida o *dominio* (p.e. el conjunto de los números reales $\mathbb{R}$, el conjunto de puntos en un plano $\mathbb{R}^2$ o de eventos en el espacio

[2] http://dynref.engr.illinois.edu/rvy.html
[3] http://dynref.engr.illinois.edu/rvy.html

BORRADOR

tiempo $\mathbb{R}^4$) con los de otro conjunto, llamado rango, de modo a que cada elemento del dominio le corresponde **uno y solo un elemento del rango**.

Entre los distintos tipos de funciones que reconoce el análisis matemático, en este libro nos concentraremos en:

- **Funciones de variable real**: Dominio y rango $\mathbb{R}$. Ejemplo: $f(t) = t^2$.
- **Funciones de muchas variables** o **Campos escalares**: Dominio $\mathbb{R}^n$, rango $\mathbb{R}$. Ejemplos: $f(x,y) = x^2 + y^2$, $H(\{q_i\}_N) = \sum_i q_i^2$ (para la notación del conjunto $\{q_i\}$ ver la Sección 1.1.1).
- **Funciones vectoriales**: Dominio $\mathbb{R}$, rango $\mathbb{R}^n$. Ejemplo: $\vec{a}(t) = kr^{-n}\hat{e}_r$.
- **Funciones vectoriales de muchas variables** o **Campos vectoriales**: Dominio $\mathbb{R}^n$, rango $\mathbb{R}^3$. Ejemplo: $\vec{F}(r,\theta,z) : -k(r\cos\theta, r\sin\theta, z)$.

Nota

***t* como variable genérica de las funciones.** En todos los textos de matemáticas (incluso en los de física) se acostumbra usar x como el nombre preferido para representar, de forma genérica, la variable independiente de las funciones. En lo sucesivo cambiaremos esta convención al llamar t a la variable independiente genérica. La razón no puede ser más sencilla: en la mecánica t es el nombre que damos a la variable independiente por excelencia, el tiempo, de modo que muchas de las fórmulas que desarrollaremos en este capítulo, se trasladaran simbólicamente casi sin modificación a la mecánica.
Es obvio que la elección de la letra con la que representamos la variable independiente, no modifica en nada las definiciones y teoremas que veremos en esta sección, de modo que esperamos esta elección no moleste a los más conservadores ni confunda a quienes han estudiado ampliamente estos temas en otros textos.

Algoritmos para funciones

Hay dos maneras de definir una función en `Python`: 1) como una rutina o 2) como una función `lambda`.

Como una rutina, una función en `Python` puede recibir como "argumentos" de entrada no solo las variables de la función sino también argumentos opcionales.

La siguiente función, por ejemplo, permite calcular el valor de la energía potencial de un sistema físico usando la función de varias variables $U(\vec{r}) = kr^n$ (siendo $\vec{r} : x, y, z$ el vector posición y r su magnitud.)

(Algoritmo 1.1)

```
def U(x,y,z,k=1,n=-1):
    r=(x**2+y**2+z**2)**0.5
    return k*r**n
```

```
U(1.0,2.0,0.0) con k = 1 y n = -1 (valores por defecto) = 0.4472135954999579
U(1.0,2.0,0.0) con k = 6.67e-11 y n = -2 = 1.334e-11
```

Nota

Argumentos obligatorios y argumentos opcionales. Toda rutina en `Python` puede tener unos argumentos obligatorios (que llamaremos variables) o unos opcionales.
Las variables son en estricto sentido una `tupla` de valores, por ejemplo `x,y,z` en la función U en el Alg. (1.1).
Los argumentos opcionales son, por otro lado, un `diccionario` de valores, que no es otra cosa que una lista de valores identificados con un nombre (también llamdo clave o `key`). En la función U en el Alg. (**??**) los argumentos opcionales son `k=1,n=-1`.
En `Python` las variables y las opciones de una rutina pueden representarse usando los objetos especiales `*variables` y `**opciones`. El uso de estos objetos especiales no es muy común, pero en ciertas situaciones puede ser bastante útil.
Una forma alternativa de la rutina para U en el Alg. (**??**) es:

```
def U(*variables,**opciones):
  x,y,z=*variables
  r=(x**2+y**2+z**2)**0.5
  return opciones["k"]*r**n
```

que se puede invocar usando:

```
var=1.0,2.0
opc=dict(k=1,n=-2)
U(*var,**opc)
```

No parece muy práctico, pero como veremos puede ser muy útil en ciertas situaciones especiales.

Las funciones `lambda` se usan para representar funciones muy abreviadas y no tienen argumentos distintos de las variables de las que dependen.

Así, por ejemplo, el siguiente algoritmo define una función `lambda`, `U_x`, basada en la función U del Alg. (**??**), que depende solo de la variable x cuando y asumes constante los valores de y y z (`U_x` será util para calcular más abajo la derivadas parcial de U respecto a x):

(Algoritmo 1.2)

```
y=1.0
z=1.0
k=1
n=-2
U_x=lambda x:U(x,y,z,k,n)
```

```
Ux(0.0) = 0.49999999999999994
```

1.1.4. Derivadas

La derivada de una función de variable real es en sí misma una función definida por el límite:

$$\frac{\mathrm{d}f}{\mathrm{d}t} \equiv \lim_{\Delta t \to 0} \frac{f(t) - f(t + \Delta t)}{\Delta t} \tag{1.16}$$

Si el límite no existe decimos que la función no es derivable en t.

Nota

Notación de la derivada. A lo largo de la historia la manera como se ha representado la función derivada ha cambiado. Existen al menos tres notaciones comunes:

- La **notación de Leibniz**, $\mathrm{d}f/\mathrm{dt}$, $\mathrm{d}^2f/\mathrm{d}t^2$. En esta notación la derivada se representa como si fuera la razón entre dos cantidades, pero no es así ¡mucho cuidado! Usaremos la notación de Leibniz especialmente para representar la derivada de funciones que se escriben de forma explícita, así por ejemplo:

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{1}{2}t^2\right)$$

- La **notación de Newton**, $\dot{f}$, $\ddot{f}$. Esta será la forma que usaremos para denotar a lo largo del libro las derivadas respecto del tiempo (o el tiempo propio en relatividad).
- La **notación de Lagrange**, f', f'', que no usaremos en este libro pero que pueden encontrar en muchos textos clásicos de mecánica celeste o analítica.
- La **notación de Euler**, $\mathrm{D}f$, D^2f, es la notación menos común, pero puede aparecer en el contexto de la mecánica de fluídos.

La definición de derivada de las funciones de variable real como un límite, se extiende por analogía a campos escalares, funciones vectoriales o campos vectoriales. Para las funciones que dependen de varias variables, sin embargo, se usa una notación y un nombre diferente: **derivada parcial**. La derivada parcial de un campo escalar se define como:

$$\frac{\partial f}{\partial q_k} = \lim_{\Delta q_k \to 0} \frac{f(q_1, q_2, \ldots, q_k + \Delta q_k, \ldots, q_N) - f(q_1, q_2, \ldots, q_k, \ldots, q_N)}{\Delta q_k}$$

La derivada parcial se calcula de la misma manera que la derivada de una variable, con la salvedad de que al hacerlo se asume que todas las demás variables de la función son constantes.

En muchas partes en este libro, y por economía usaremos la notación de Euler para las derivadas parciales, a saber:

$$\partial_x f \equiv \frac{\partial f}{\partial x}$$

En esta notación una derivada parcial múltiple se escribirá como:

$$\partial_{xyz} f(x,y,z) \equiv \frac{\partial^3 f}{\partial x \partial y \partial z}$$

A pesar de que la derivada parcial tiene una definición *numérica* análoga a la de la derivada total, existe una sútil diferencia entre ambas.

Imagine que tenemos una **variable independiente** t y definimos, a partir de ella, una nueva variable u que es función de t (variable dependiente).

¿Cómo podemos calcular la derivada de una función de la nueva variable $f(u)$ respecto de la variable independiente t?

Teorema 1.1

Regla de la Cadena. Dada una función compuesta $f(u(t))$, la derivada de f respecto a t es:

$$\frac{\mathrm{d}f}{\mathrm{d}t} = \frac{\mathrm{d}f}{\mathrm{d}u}\frac{\mathrm{d}u}{\mathrm{d}t}$$

Decimos que la función f depende *implícitamente* de la variable independiente t. En este sentido la regla de la cadena es una regla de *derivación implícita*.

Usando la notación de Newton la expresión anterior se escribirá de forma abreviada:

$$\dot{f}(t) = \dot{u}\frac{\mathrm{d}f}{\mathrm{d}u}$$

¿Qué pasa en el caso en el que f depende de varias variables dependientes, por ejemplo $f(q_1(t), q_2(t), \ldots, q_N(t)) \equiv f(\{q_i(t)\}_N)$?

En este caso la regla de la cadena se puede generalizar como:

$$\frac{\mathrm{d}f}{\mathrm{d}t} = \frac{\partial f}{\partial q_1}\frac{\mathrm{d}q_1}{\mathrm{d}t} + \frac{\partial f}{\partial q_2}\frac{\mathrm{d}q_2}{\mathrm{d}t} + \ldots \frac{\partial f}{\partial q_N}\frac{\mathrm{d}q_N}{\mathrm{d}t} = \sum_i \frac{\partial f}{\partial q_i}\frac{\mathrm{d}q_i}{\mathrm{d}t} = \sum_i \dot{q}_i \partial_{q_i} f$$

Ahora bien: ¿existirá, en este caso, la derivada parcial de f respecto de t?

La respuesta a esta pregunta, ilustra, justamente, la diferencia sutil entre la derivada ordinaria o *derivada total* $\mathrm{d}/\mathrm{d}t$ y la derivada parcial $\partial/\partial t$.

Hay dos situaciones posibles:

- Si la función f *no depende* explícitamente de t, es decir si la variable t no aparece en la fórmula de f, entonces $\partial f/\partial t = 0$. Este resultado es *independiente* de que f dependa implícitamente de t a través de otras variables dependientes.

  Ejemplo: si $f(q,t) = q^2$, entonces: $\partial f/\partial t = 0$ aunque, por regla de la cadena, $\mathrm{d}f/\mathrm{d}t = 2q\dot{q}$.

- Si la fórmula de la función f contiene la variable t, entonces su derivada parcial puede ser distinta de cero (dependiendo de la forma funcional de f).

BORRADOR

Ejemplo: Si $f(q,t) = q^2 + \sin t$, entonces: $\partial f/\partial t = \cos t$ y $\mathrm{d}f/\mathrm{d}t = 2q\dot{q} + \cos t$.

En este sentido la derivada parcial es como un "operador semántico", es decir un operador sobre las "letras" que aparecen en la fórmula de la función.

Teniendo en cuenta esta propiedad, la forma más general de la regla de la cadena, para una función de varias variables (campo escalar o vectorial) será:

$$\frac{\mathrm{d}}{\mathrm{d}t} f(\{q_i\}, t) = \sum_i \dot{q}_i \partial_{q_i} f + \frac{\partial f}{\partial t} \tag{1.17}$$

1.1.5. Funciones homogéneas

Existe un interesante conjunto de funciones para las cuáles hay una relación no trivial entre su derivada y el valor de la función misma. Se conocen como **funciones homogéneas**:

Definición 1.1

Funciones homogéneas. Una función general $f(\{q_i\})$ se llama homogénea si frente a una operación de escalado de sus variables (multiplicación por un escalar), la función *escala* también. En términos matemáticos:

$$f(\{\lambda q_i\}) = \lambda^k f(\{q_i\})$$

donde λ es un número real y k se conoce como el **orden** de la función.

Las funciones homogéneas son, generalmente polinomios y funciones racionales. Así por ejemplo $f(x,y) = x^2/a^2 + y^2/b^2$, con a y b constantes, y que representa la ecuación algebraica de una elipse, es una función homogénea de grado $k = 2$. De otro lado $f(x) = x^3y^2 + y^5$ es homogénea de grado $k = 5$.

Las funciones homogéneas más interesantes para nosotros en este libro son del tipo $f(\vec{r}) = kr^n$ que son homogéneas de grado $k = n$ (ver problemas al final del capítulo.)

Como mencionamos desde el principio, las derivadas de las funciones homogéneas tienen una propiedad muy importante:

Teorema 1.2

Teorema de funciones homogeneas de de Euler. Si una función $f(\{q_i\}_N)$ es homogénea de grado k, entonces:

$$\sum_i q_i \frac{\partial f}{\partial q_i} = kf$$

Para funciones homogéneas definidas en el espacio de tres dimensiones, el teorema de Euler se puede escribir como:

$$\vec{r} \cdot \vec{\nabla} f = kf$$

1.1.6. Derivada vectorial

Para funciones de varias variables (especialmente aquellas con dominio en el espacio coordenado $\mathbb{R}^3$) se definen generalizaciones vectoriales de la derivada que tienen motivaciones e interpretaciones geométricas específicas.

El *operador diferencial vectorial* básico se conoce como el **gradiente**. Denotado comúnmente como $\vec{\nabla}$, en coordenadas cartesianas se define explícitamente como:

$$\vec{\nabla} f(x,y,z) = \frac{\partial f}{\partial x}\hat{e}_x + \frac{\partial f}{\partial y}\hat{e}_y + \frac{\partial f}{\partial z}\hat{e}_z \tag{1.18}$$

El operador gradiente en el sistema de coordenadas cilíndrico (con la notación definida anteriormente) esta dado por:

$$\vec{\nabla} f(r,\theta,z) = \frac{\partial f}{\partial r}\hat{e}_r + \frac{1}{r}\frac{\partial f}{\partial \theta}\hat{e}_\theta + \frac{\partial f}{\partial z}\hat{e}_z \tag{1.19}$$

Donde el factor $1/h_\theta \equiv 1/r$ se conoce como *factor de escala*.

Por su parte en coordenadas esféricas (con la notación definida anteriormente):

$$\vec{\nabla} f(r,\theta,\phi) = \frac{\partial f}{\partial r}\hat{e}_r + \frac{1}{r}\frac{\partial f}{\partial \theta}\hat{e}_\theta + \frac{1}{r\cos\phi}\frac{\partial f}{\partial \phi}\hat{e}_\phi \tag{1.20}$$

En este caso se ha introducido un nuevo factor de escala: $h_\phi \equiv r\cos\phi$.

Nota

Una notación para el gradiente. Como lo hicimos con la derivada parcial, a lo largo de este libro, abreviaremos el gradiente usando la notación especial:

$$\partial_{\vec{r}} f \equiv \frac{\partial f}{\partial \vec{r}} \equiv \vec{\nabla} f$$

Aunque no es una notación muy rigurosa, permite abreviar expresiones que de otra manera serían muy elaboradas. Así por ejemplo, la regla de la cadena (Ec. 1.17) para funciones definidas en el espacio coordenado, se puede escribir de forma compacta como:

$$\dot{f}(x,y,z,t) = \partial_{\vec{r}} f \cdot \dot{\vec{r}} + \partial_t f \tag{1.21}$$

Existen otros operadores vectoriales (laplaciano, divergencia, rotacional) sobre los que no profundizaremos aquí por no ser de mucha utilidad práctica en la mecánica celeste (al menos no al nivel de este libro.)

Algoritmos para la derivada

Existen diversos algoritmos para calcular la derivada de una función en una o varias variables. En este libro, en donde sea necesario, nos apoyaremos de la biblioteca científica `scipy` y su rutina `derivative` que permite calcular, numéricamente, derivadas de cualquier orden.

El siguiente algoritmo ilustra el uso de `derivative` y sus opciones:

```
def f(t):
    from math import sin
    return sin(t)/t

#Valor de la variable independiente donde queremos la derivada
t=2.0

from scipy.misc import derivative

#Primera derivada usando un dx=0.01 y 3 puntos
dfdt=derivative(f,t,dx=1e-2,n=1,order=3)

#Segunda derivada en t
d2fdt2=derivative(f,t,dx=1e-2,n=2,order=5)
```

```
dfdt : Numérica = -0.4353938258295498, Exacta = -0.43539777497999166
d2fdt2 : Numérica = -0.019250938436687903, Exacta = -0.01925093843284925
```

Usando `derivative` es posible diseñar funciones para calcular derivadas parciales e incluso gradientes (para los cuáles no existen funciones en la biblioteca `scipy`). Así por ejemplo:

```
def f(x,y,z):
    from math import sin
    return sin(x*y*z)/(x*y*z)

def partial_derivative_x(f,x,y,z,**opciones):
    f_solo_x=lambda x:f(x,y,z)
    dfdx=derivative(f_solo_x,x,**opciones)
    return dfdx

x=1.0
y=2.0
z=3.0
dfdx=partial_derivative_x(f,x,y,z,dx=0.01)
```

```
dfdx: Numérica = 1.0061803563982654, Exacta = 1.006739536350187
```

Nótese como usamos aquí la función `lambda f_solo_x`, de la manera que lo hicimos en el Algorítmo (1.2) para conseguir el resultado deseado.

1.1.7. Integrales

Se llama **antiderivada** de una función de variable real $f(t)$, a la función $F(t)$ cuya derivada es igual a la función original:

$$\dot{F}(t) = f(t)$$

O en notación *integral*:

$$F(t) \equiv \int f(t)\, \mathrm{d}t$$

A $F(t)$ o equivalentemente $\int f(t)\ \mathrm{d}t$ se la llama también la **integral indefinida** de $f(t)$.

La antiderivada permite calcular la **cuadratura de una función**, que no es otra cosa que el área encerrada por la curva en el plano cartesiano definido por la variable independiente y los valores de la función:

Teorema 1.3

Fórmula de Newton-Leibniz. [a]. Dada una función $f(t)$ que tiene antiderivada $F(t)$ definida en el intervalo $[a, b]$, el área o cuadratura de la función en el mismo intervalo esta dado por:

$$\int_a^b f(t)\ \mathrm{d}t = F(b) - F(a)$$

A la cantidad $\int_a^b f(t)\ \mathrm{d}t$ se la llama **integral definida** de $f(t)$.

[a] A esta fórmula se la llama a menudo *segundo teorema fundamental del cálculo*

En términos de la integral definida podemos definir una nueva función:

$$I(t) = \int_a^t f(\tau)\ \mathrm{d}\tau$$

Nótese que para ser rigorusos hemos cambiado el nombre de la "variable de integración" τ para no confundirla con el límite superior de la integral t.

Esta nueva función tiene una importante propiedad:

Teorema 1.4

Teorema fundamental del cálculo. Dada una función $f(t)$ integrable en el intervalo $[a, b]$, si definimos la función $I(t) = \int_a^t f(\tau)\ \mathrm{d}\tau$, entonces:

$$\frac{\mathrm{d}I}{\mathrm{d}t} = f(t)$$

o bien,

$$\frac{\mathrm{d}}{\mathrm{d}t}\int_a^t f(\tau)\ \mathrm{d}\tau = f(t) \tag{1.22}$$

Es interesante anotar que aunque la antiderivada $F(t)$ y la función $I(t)$ tienen la misma derivada en t, es decir $\mathrm{d}F/\mathrm{d}t = \mathrm{d}I/\mathrm{d}t = f(t)$, no son necesariamente la misma función. Considere, por ejemplo, el hecho elemental de que $I(a) = 0$ (por definición) mientras que $F(a)$ podría ser cualquier número (incluyendo cero por supuesto.)

1.1.8. Integrales vectoriales

Una extrapolación del concepto de integral a funciones de varias variables (campos escalares y campos vectoriales) conduce a algunas operaciones integra-

les de gran importancia en la física. Para los propósitos de lo que veremos en este libro, son de particular interés las integrales del tipo:

$$\int \vec{F} \cdot \mathrm{d}\vec{r},$$

que se define sobre todos los valores de $\vec{r}$ de una curva en el espacio coordenado. A esta integral se la conoce como **integral de línea**. Si la trayectoria es cerrada, escribiremos:

$$\oint \vec{F} \cdot \mathrm{d}\vec{r},$$

que no se diferencia (matemáticamente) en nada de una integral de línea. A esta integral la llamaremos **circulación** del campo vectorial $\vec{F}$.

Otro tipo de integral vectorial de interés es:

$$\int_{\Sigma} \vec{F} \cdot \mathrm{d}\vec{S}$$

Donde $\mathrm{d}\vec{S}$ tiene dirección normal a la superficie Σ (formada por el lugar geométrico de todos los puntos que la definen) y magnitud igual al área de una fracción infinitesimal de la superficie.

Teorema 1.5

teorema de Stokes. Si $\vec{F}(\vec{r})$ es un campo vectorial diferenciable en todos los puntos del espacio, entonces:

$$\oint \vec{F} \cdot \mathrm{d}\vec{r} = \int_{\Sigma} (\vec{\nabla} \times \vec{F}) \cdot \mathrm{d}\vec{S}$$

Donde Σ es cualquier superficie que tenga como frontera la trayectoria sobre la que se define la circulación.

Un importante corolario del teorema de Stokes es el siguiente:

Teorema 1.6

Corolario del teorema de Stokes. Si el campo vectorial $\vec{F}(\vec{r})$ tiene circulación nula:

$$\oint \vec{F} \cdot \mathrm{d}\vec{r} = 0$$

Entonces existe un campo escalar $U(\vec{r})$ tal que:

$$\vec{F} = \vec{\nabla} U$$

Llamamos a U la función *potencial* de $\vec{F}$.

Algoritmos para la integral

El cálculo numérico de integrales es una basta área del análisis numérico. En cada lenguaje de programación es posible encontrar bibliotecas completas con rutinas para el cálculo de aproximaciones numéricas de integrales definidas e integrales vectoriales.

Para los propósitos de este libro, usaremos la rutina `quad` de la biblioteca `SciPy` para calcular numéricamente integrales definidas de funciones de variable real.

En el algoritmo provisto a continuación, calculamos, por ejemplo, el trabajo $W \equiv \int F(x)\ \mathrm{d}x$ sobre una partícula que se mueve en una dimensiónm sometida a una fuerza del tipo $F(x) = -kx$, asumiendo que $k = 0{,}1$ y que la partícula se desplaza entre $x = 1{,}0$ y $x = 5{,}0$:

```
#El integrando debe definirse como una rutina
def F(x,k=1):
    return -k*x

from scipy.integrate import quad
k=0.1
x0=1.0
x1=5.0
integral=quad(F,x0,x1,args=(k,))
```

```
Integral: Numérica = (-1.2, 1.3322676295501878e-14), Exacta = -1.2
```

Nótese que los argumentos opcionales del integrando se pasan como la `tupla` `args` que en este caso, dado que la función solo depende de un parámetro opcional, se escribe de forma poco intuitiva como `args=(k,)` donde la coma final es oblogatoria.

El resultado de la turina `quad` es una `tupla` con dos números: el valor de la integral y el error estimado de la misma. Como vemos, en el ejemplo arriba, la integral es prácticamente exacta.

Nota

Cuadraturas Gaussianas. El método usado por `quad` para calcular la integral se conoce como *cuadraturas gaussianas* y aproxima la integral como una serie de pocos términos del valor de la función definido en algunos puntos específicos [4]. Las cuadraturas gaussianas permiten calcular la integral de funciones polinómicas de forma *exacta*. Esta es la razón por la cuál la integral en el ejemplo dado aquí, es idéntica al valor esperado.

1.1.9. Ecuaciones diferenciales

Encontrar la antiderivada de una función (ver Sección 1.1.7), se puede formular, de forma general, como el problema de encontrar una función $F(t)$ tal que:

$$\frac{\mathrm{d}F(t)}{\mathrm{d}t} = f(t) \tag{1.23}$$

La solución a este problema es, por definición:

$$F(t) = \int f(t)\, \mathrm{d}t$$

La integral indefinida en el lado derecho de la anterior ecuación y los métodos numéricos o exactos (analíticos) para obtenerla (no cubiertos en este corto resumen) representan unas de las herramientas matemáticas más útiles de la física.

Pero, existen situaciones en las que el cálculo de una antiderivada no se reduce simplemente a una integral indefinida. Considere por ejemplo el siguiente problema:

$$\frac{\mathrm{d}^2 F(t)}{\mathrm{d}t^2} = -kF(t) \tag{1.24}$$

que, en palabras, se formularía como: encontrar la función cuya segunda derivada es proporcional (k se supone constante) al negativo de ella misma.

Ambas, las Ecs. (1.23) y (1.24) se conocen como **ecuaciones diferenciales**.

Las ecuaciones diferenciales se clasifican según:

- Su **orden**. El orden de una ecuación diferencial es igual al máximo orden de la derivada de la función objetivo (antiderivada) que aparece en la ecuación. La ecuación diferencial básica (1.23) es, por ejemplo, de *primer* orden porque solo involucra la *primera* derivada de la función $F(t)$. Por su parte, la ecuación diferencial (1.24) es una ecuación diferencial de segundo orden.

- Su **linealidad**. Una ecuación diferencial que solo depende de primeras potencias de la función y sus derivadas se dice que es lineal. En caso contrario tenemos una *ecuación diferencial no lineal*. Las ecuaciones (1.23) y (1.24) son lineales, pero la siguiente ecuación diferencial de primer orden, no lo es:

  $$\frac{\mathrm{d}F(t)}{\mathrm{d}t} = \frac{h}{F(t)},$$

  donde h es una constante.

- El **número de variables independientes**. Una ecuación diferencial en la que la función depende de una sola variable real se conoce como una **ecuación diferencial ordinaria** (*ODE* por la sigla en inglés de *ordinary differential equation*). Si, por otro lado, la función es un campo escalar o vectorial y la ecuación diferencial se expresa en términos de derivadas parciales (y totales) hablamos de una **ecuación diferencial parcial** (*PDE* por sus siglas en inglés).

- El **número de funciones** o **variables dependientes**. Es posible que un problema implique encontrar más de una antiderivada. En ese caso hablamos de un **sistema de ecuaciones diferenciales**. Un caso común de sistemas de ecuaciones diferenciales se produce cuando queremos encontrar la antiderivada de una función vectorial (cada componente de una función vectorial es una función en sí misma). El caso más importante en la física de un sistema de ecuaciones diferenciales es la *ecuación de movimiento de una partícula* (que exploraremos a fondo en la **??**):

BORRADOR

$$\frac{\mathrm{d}^2\vec{r}(t)}{\mathrm{d}t^2} = \vec{a}$$

Esta ecuación es una forma abrevida de escribir el sistema de ecuaciones diferenciales:

$$\begin{aligned}\frac{\mathrm{d}^2x(t)}{\mathrm{d}t^2} &= a_x\\ \frac{\mathrm{d}^2y(t)}{\mathrm{d}t^2} &= a_y\\ \frac{\mathrm{d}^2z(t)}{\mathrm{d}t^2} &= a_z\end{aligned} \tag{1.25}$$

Como las cantidades a_x, a_y y a_z pueden ser a su vez funciones del tiempo, de las funciones x, y, z y de sus derivadas, se habla, además, de un **sistema de ecuaciones diferenciales acopladas**.

- Las **condiciones que deben proveerse para resolverla**. La solución abstracta de una ecuación diferencial, es decir, el problema de encontrar la antiderivada general, es el equivalente a la integral indefinida. Las integrales definidas, por su lado, equivalen en la teoría de ecuaciones diferenciales a los que se conocen como **problemas de valor incial** (*IVP* por el acrónomimo en inglés de *initial value problem*.) En este tipo de problemas la solución a la ecuación diferencial consiste en encontrar el valor de la función para cualquier valor de la variable independiente una vez se ha provisto el valor de la función (o funciones) y de sus derivadas, en un valor específico o inicial de la variable independiente. Así por ejemplo:

  $$\frac{\mathrm{d}^2\vec{r}(t)}{\mathrm{d}t^2} = \vec{a}, \text{ con } \vec{r}(0) : (0,0,0) \text{ y } \dot{\vec{r}}(0) : (1,0,0)$$

  es un IVP.

  Por otro lado un **problema de condiciones de frontera** (*BVP* por la sigla en inglés de *boundary value problem*) es aquel en el que el valor de la función dependiente (no de sus derivadas necesariamente) se provee para varios valores de la variable independiente. Así por ejemplo:

  $$\frac{\mathrm{d}^2F(t)}{\mathrm{d}t^2} = -F(t), \text{ con } F(0) = 0 \text{ y } F(\pi/2) = 1{,}0,$$

  es un BVP.

Como se intuye fácilmente, la dificultad en la solución a una ecuación diferencial, como sucede también con las ecuaciones algebraicas, puede aumentar con su orden. Sin embargo, usando variables auxiliares, siempre es posible escribir una

BORRADOR

ecuación diferencial de orden M como un sistema de M ecuaciones diferenciales de primer orden.

Por ejemplo, si en la Ec. (1.24) llamamos $G(t) \equiv \mathrm{d}F(t)/\mathrm{d}t$, esa ecuación diferencial de segundo orden se puede escribir como el sistema de ecuaciones diferenciales:

$$\begin{array}{rcl} \mathrm{d}F/\mathrm{d}t & = & G \\ \mathrm{d}G/\mathrm{d}t & = & -kF \end{array} \tag{1.26}$$

A esta sistema de ecuaciones, lo llamamos el *sistema de ecuaciones diferenciales reducido*.

La reducción del orden será un método muy utilizado en este libro para abordar la solución a las ecuaciones diferenciales de la mecánica celeste y analítica.

Algoritmos para la solución de ODE

La solución aproximada de ecuaciones diferenciales es una de las áreas de mayor interés en el análisis numérico. Sus beneficios prácticos se extienden desde la física teórica y la economía hasta la climatología y la simulación del vuelo de aviones y vehículos espaciales. A lo largo de los últimos 350 años (y en paralelo con la evolución de la mecánica), se han desarrollado métodos numéricos para aproximar la solución de todos los tipos de ecuaciones diferenciales que hemos mencionado hasta aquí.

En este libro, sin embargo, nos concentraremos en la solución de sistemas ecuaciones diferenciales ordinarias con valores iniciales o *IVP*.

Los métodos numéricos generales, desarrollados para resolver este tipo de problemas (ver [4] para detalles sobre los métodos y algoritmos explícitos), suponen que la ecuación o sistema de ecuaciones diferenciales que queremos resolver puede escribirse como un sistema reducido de ecuaciones diferenciales de primer orden de la forma:

$$\{\dot{Y}_i = f_i(\{Y_k\}, t)\}_{i=0,1,\ldots,M} \tag{1.27}$$

Donde Y_i ($i = 0, 1, 2, \ldots, M-1$) es el conjunto de funciones auxiliares que reemplaza a las funciones dependientes y sus derivadas de orden inferior y f_i son las función que proveen el valor de la primera derivada de la variable auxiliar Y_i.

Así por ejemplo, para resolver la ecuación diferencial (1.24), que ya habíamos reducido como las ecuaciones (1.26), las variables auxiliares y sus derivadas serían:

$$\begin{array}{rcl} Y_0 = F & , & f_0(t, Y_0, Y_1) = Y_1 \\ Y_1 = G & , & f_1(t, Y_0, Y_1) = -kY_0 \end{array} \tag{1.28}$$

Con esta identificación, el problema original puede escribirse, de forma general como la Ec. (1.27).

En este libro usaremos la rutina `odeint` de la biblioteca científica `SciPy` (ver *Nota* abajo) para integrar numéricamente sistemas de ecuaciones diferenciales de la forma reducida. El lector puede leer la documentación completa de `odeint`[4] para conocer los detalles de su aplicación.

[4] https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.odeint.html

BORRADOR

El primer paso para usar `odeint` es implementar las ecuaciones reducidas como una rutina. En nuestro ejemplo (Ec. 1.28) la rutina sería:

```
def ode_simple(Y,t,k=1):
    f=[0,0]
    f[0]=Y[1]
    f[1]=-k*Y[0]
    return f
```

Los primeros dos argumentos de esta rutina (ver Sección 1.1.3), es decir `Y` (que contiene una lista de los valores instantáneos de las variables auxiliares Y_i) y `t` (el tiempo en el que las variables auxiliares tienen ese valor) deben estar, estrictamente en ese orden. Otros podrían encontrar más natural poner de primero el tiempo, pero `odeint` esta diseñado para trabajar con rutinas con este *prototipo* particular. Además de estos argumentos obligatorios, la rutina puede tener cualquier otro argumento opcional. En este caso aprovechamos esta libertad para proveer el valor de la constante `k`, que aparece en la ecuación diferencial, y para el cual hemos asumido un valor por defecto `k=1` (naturalmente el usuario de la rutina podrá especificar un valor distinto cuando la llame.)

Para resolver este conjunto de ecuaciones diferenciales debemos, además de la rutina anterior, proveer:

1. Valores específicos para los parámetros de la ecuación diferencial (en este caso la constante k),
2. Una lista de condiciones iniciales, es decir de los valores iniciales de las variables auxiliares $\{Y_i(t = t_0)\}$
3. un conjunto de valores del tiempo (incluyendo el tiempo inicial t_0) para los cuales deseamos predecir el valor de la antiderivada (función o funciones dependientes.)

El siguiente algoritmo prepara estos insumos para `odeint` en nuestro ejemplo particular:

```
from numpy import array

k=1.5

Yos=array([1.0,0.0])

ts=array([0.0,1.0,2.0,3.0,4.0,5.0])
```

Nótese que para las condiciones iniciales y los valores de tiempo (que son aquí arbitrarios, el lector podría escoger unos completamente diferentes) hemos escogido usar arreglos de `NumPy` (`array`) en lugar de listas planas (ver Sección 1.1.1). Aunque esto no es obligatorio, más adelante hará más fácil la manipulación matemática de estas variables.

BORRADOR

Nota

El plural en los algoritmos. Preste atención a la convención que usaremos en lo sucesivo de usar la letra `s` como sufijo del nombre de algunos arreglos y matrices (p.e. `Yos`, `ts`). En lo sucesivo (a no ser que se indique lo contrario) `t` denotará un valor individual de la variable, pero `ts` será un arreglo de valores de `t`.

La solución numérica al cojunto de ecuaciones diferenciales implementados en la rutina `ode_simple` se obtiene, finalmente, invocando `odeint`:

```
from scipy.integrate import odeint
Ys=odeint(ode_simple,Yos,ts,args=(k,))
```

```
Solución, Ys =
[[ 1.          0.        ]
 [ 0.33918602 -1.15214115]
 [-0.76990562 -0.78158038]
 [-0.86146852  0.6219388 ]
 [ 0.18550948  1.20348632]
 [ 0.987313    0.1944726 ]]
```

Las filas de la matriz solución `Ys`, contienen el valor de las variables auxiliares $\{Y_i\}$ en cada uno de los tiempos provistos. Las columnas, naturalmente, corresponden a los valores instantáneos de cada una de esas variables auxiliares. Así, la componente `Ys[0,0]` corresponde al valor de Y_0 (es decir el valor de la función F de nuestro ejemplo) en t_0 (condición inicial).

También es posible extraer tajadas de la matriz. Así, `Ys[:,1]` (que podría leerse como *el segundo valor de cualquier fila* o simplemente *la columna 1*), corresponde al valor de la función auxiliar G en cada uno de los tiempos de integración (recuerde que $G = Y_1$, ver la identificación en la Ec. 1.28).

Usando la matriz de solución `Ys` es posible, finalmente, hacer el gráfico de la función $F(t)$ que estabamos buscando:

(Algoritmo 1.3)

```
import matplotlib.pyplot as plt

#Extraemos los valores de la función F
Fs=Ys[:,0]

plt.figure();
plt.plot(ts,Fs,marker='o',linewidth=0);

#--hide--
plt.xlabel("$t$");
plt.ylabel("$F(t)$");
```

ver Figura 1.3

Naturalmente la resolución de este gráfico es bastante pobre porque hemos pedido al algoritmo encontrar únicamente los valores de $F(t)$ en 5 valores del tiempo

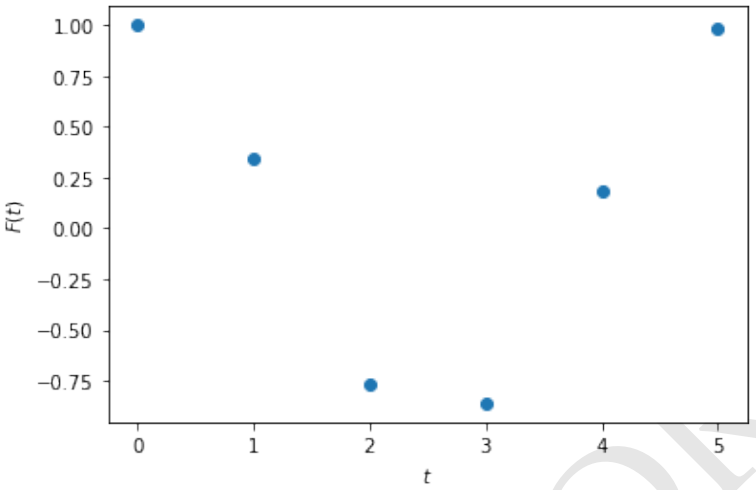


Figura 1.3: *Figura correspondiente al código 1.3. Solución aproximada de la ecuación diferencial* $d^2\mathrm{F}/dt^2 = -kF$ *con k=1.5.*

(arreglo `ts`.) Si se incrementa el número de componentes de este vector el resultado será mucho más cercano al que esperamos de una función.

Nota

Los algoritmos detrás de `odeint`. La rutina `odeint` es un *empaque* en `Python` (*wrap* en inglés) de un complejo y robusto paquete de rutinas conocido como `ODEPACK`[a]. Desarrollado por el *Center for Applied Scientific Computing* del *Lawrence Livermore National Laboratory*, las rutinas de `ODEPACK` están escritas en lenguaje `FORTRAN77` (`Python` se usa únicamente para pasar los parámetros al paquete y para recuperar las salidas; ese es justamente el sentido del nombre "empaque") y han sido probadas y perfeccionadas durante varias décadas en distintas aplicaciones científicas y de ingeniería [3].
Existen otras rutinas en el paquete `SciPy` para resolver ecuaciones diferenciales con condiciones inicales (IVP). Por ejemplo `ode` y `solve_ivp` pueden usarse también (esta última es, por ejemplo, la recomendada por los desarrolladores de `SciPy`). Sin embargo, estas otras rutinas tienen una *interface* un poco más complicada. Así por ejemplo, para integrar la e.d.m. del ejemplo visto aquí, usando `solve_ivp`, el código **mínimo** en `Python` sería:

```
from scipy.integrate import solve_ivp
solucion=solve_ivp(fun=lambda t,Y:ode_simple(Y,t,k),
                   t_span=[ts[0],ts[-1]],y0=Yos,t_eval=ts)
```

Como puede apreciarse la complejidad del código supera con creces la de aquel que usamos para invocar `odeint`. A esto se suma el hecho de

Nota (Cont.)

que la solución, que en el caso de `odeint` es una matriz `Ys` fácil de interpretar, en el caso de `solve_ivp` es en realidad un *objeto* cuyo *atributo* `solucion.y` contiene la solución que buscamos. Y finalmente, pero no menos importante: para el tipo de ecuaciones diferenciales que usaremos en este libro `solve_ivp` es casi dos veces más lento que `odeint`. El lector sin embargo puede explorar esas otras alternativas, especialmente si quiere, por ejemplo, comparar distintos métodos de solución (a diferencia de `odeint`, `solve_ivp` escoger el método de solución.)

[a] https://computing.llnl.gov/casc/odepack

1.1.10. Funcionales y cálculo de variaciones

Un tema poco cubierto en los textos básicos de cálculo, pero de gran utilidad en la mecánica, es el denominado **cálculo de variaciones**. Si bien en esta sección de "repaso" no pretendemos ofrecer una introducción detallada a esta importante área del análisis matemático, es necesario presentar aquí algunos resultados básicos que serán de utilidad para el resto del libro.

Si el cálculo infinitesimal, que repasamos en las secciones anteriores, trata sobre la variación continua de funciones de variable real, el cálculo de variaciones se ocupa de la variación de los que se conocen como **funcionales**.

En términos informales, un funcional es una "función de funciones", es decir, una regla de correspondencia entre el conjunto de las funciones y el de los números reales.

Un ejemplo, muy interesante e ilustrativo de un funcional, es la integral definida de una función de variable real:

$$I[f] = \int_a^b f(t)\, \mathrm{d}t$$

La notación $I[f]$, en lugar de $I(b)$ como lo usamos en Sección 1.1.7, trata de poner en evidencia el hecho de que lo que nos interesa aquí no es el valor mismo de la integral definida, sino cómo el valor de esta cantidad cambia si modificamos la función f. En la Figura 1.4 se muestra la interpretación gráfica de la integral definida. Sabemos que el área bajo la curva, el valor de nuestro funcional, dependerá de si usamos la función $f(t)$ o $f_0(t)$.

De la misma manera en la que se puede estudiar el efecto que un cambio muy pequeño Δt en el valor de la variable independiente t tiene en una función de variable real $f(t)$, como lo hicimos por ejemplo para definir la derivada (ver Sección 1.1.4), en el cálculo variacional es posible estudiar el efecto que un cambio pequeño δf de una función f tiene en el funcional $I[f]$

Para hacerlo debemos primero definir otra función η que sirve de "plantilla" para el cambio. Al cambio en f se lo llama **variación** y se escribe como:

$$\delta f \equiv \epsilon \eta \tag{1.29}$$

Una ilustración del concepto de *variación* se muestra en la Figura 1.4. Allí reconocemos una importante propiedad de la función de plantilla $\eta(t)$ y es que vale

BORRADOR

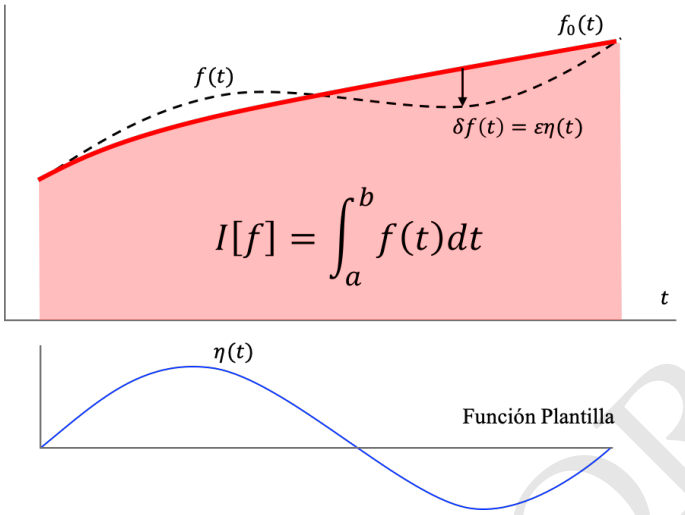


Figura 1.4: *El área bajo una curva es un funcional, en tanto depende de la función que represente la curva, $f(t)$ o $f_0(t)$ Se conoce como una variación δf a la diferencia entre dos funciones cercanas, parametrizada a través de un número real ϵ y una función plantilla (panel inferior.) En términos de variaciones el valor de cualquier función vecina a una función de referencia f_0 se puede calcular, en un intrevalo de interés, como $f(t) = f_0(t) + \epsilon\eta(t)$.*

cero en los extremos del intervalo considerado $[a, b]$.

El cálculo variacional surgió originalmente para resolver problemas prácticos en física, tales como hallar las funciones que máximan o minimizan (extremos) funcionales de alguna utilidad.

Así por ejemplo, considere la siguiente pregunta: ¿cuál es la curva más corta que conecta dos puntos en el plano de euclidiano?

Para responder a esta pregunta debemos primero construir el funcional "distancia a lo largo de una curva", también llamado, longitud de arco ([2]):

$$I[f] = \int_a^b \sqrt{1 + \left|\frac{\mathrm{d}f}{\mathrm{d}t}\right|^2}\, \mathrm{d}t \tag{1.30}$$

Queremos encontrar la función f_0 tal que $I[f_0]$ tenga el mínimo valor entre todas las posibles funciones f.

Para encontrar la función que minimiza este funcional debemos, como se acostumbra en el cálculo ([1]), derivar el funcional respecto a la cantidad que parametriza la variación: ϵ.

Escribamos el funcional de forma más general, en términos de una función cercana al mínimo escrita como $f = f_0 + \epsilon\eta$:

$$I[f] = \int_a^b L(f(t), \dot{f}(t), t)\, \mathrm{d}t \tag{1.31}$$

Nótese que hemos escrito el integrando como una función general L que depende del valor de la función $f(t)$, de su derivada $\dot{f}(t)$ y de la variable indepen-

BORRADOR

diente t. Implícitamente, el funcional depende también del parámetro ϵ dado que $f = f_0 + \epsilon\eta$.

Si derivamos el funcional respecto de ϵ, obtenemos:

$$\frac{\mathrm{d}I[f]}{\mathrm{d}\epsilon} = \int_a^b \frac{\mathrm{d}}{\mathrm{d}\epsilon} L(f(t), \dot{f}(t), t)\ \mathrm{d}t$$

Aplicando la regla de la cadena, la integral del lado derecho nos queda:

$$\frac{\mathrm{d}I[f]}{\mathrm{d}\epsilon} = \int_a^b \left(\frac{\partial L}{\partial f}\frac{\mathrm{d}f}{\mathrm{d}\epsilon} + \frac{\partial L}{\partial \dot{f}}\frac{\mathrm{d}\dot{f}}{\mathrm{d}\epsilon} \right)\ \mathrm{d}t$$

Como $f(t) = f_0(t) + \epsilon\eta(t)$, entonces $\mathrm{d}f/\mathrm{d}\epsilon = \eta$, mientras que $\mathrm{d}\dot{f}/\mathrm{d}\epsilon = \dot{\eta}$. Así la integral anterior se desarrolla como:

$$\frac{\mathrm{d}I[f]}{\mathrm{d}\epsilon} = \int_a^b \left(\frac{\partial L}{\partial f}\eta + \frac{\partial L}{\partial \dot{f}}\dot{\eta} \right)\ \mathrm{d}t \tag{1.32}$$

El término $\int_a^b (\partial L/\partial \dot{f})\dot{\eta}\ \mathrm{d}t$ se puede integrar por partes, si se hace $u = \partial L/\partial \dot{f}$ y $\mathrm{d}v = \dot{\eta}\ \mathrm{d}t$:

$$\int_a^b \frac{\partial L}{\partial \dot{f}}\dot{\eta}\ \mathrm{d}t = \frac{\partial L}{\partial \dot{f}}\eta\bigg|_a^b - \int_a^b \frac{\mathrm{d}}{\mathrm{d}x}\left(\frac{\partial L}{\partial \dot{f}}\right)\eta\ \mathrm{d}t$$

El primer término del lado derecho de la ecuación anterior es cero, en tanto, por definición $\eta(a) = \eta(b) = 0$.

Reemplazando en la Ec. (1.32), la derivada del funcional respecto de epsilon queda finalmente:

$$\frac{\mathrm{d}I[f]}{\mathrm{d}\epsilon} = \int_a^b \eta(x)\left(\frac{\partial L}{\partial f} - \frac{\mathrm{d}}{\mathrm{d}x}\frac{\partial L}{\partial \dot{f}} \right)\ \mathrm{d}x$$

Para que $I[f]$ sea mínima en $f = f_0$ su derivada $\mathrm{d}I[f]/\mathrm{d}\epsilon$ debe ser cero en $\epsilon = 0$. Esto equivale a la *ecuación integral*:

$$\int_a^b \eta(x)\left(\frac{\partial L}{\partial f} - \frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial L}{\partial \dot{f}} \right)\ \mathrm{d}t = 0 \tag{1.33}$$

que lamentablemente no es muy útil para resolver nuestro problema original. Para acercarnos a la solución necesitamos de un poderoso teorema:

Teorema 1.7

Lema fundamental del cálculo de variacions. Si una función continua $f(t)$ en el intervalo abierto (a, b) satisface la igualdad:

$$\int_a^b f(t)h(t)\ \mathrm{d}t = 0$$

para toda función $h(t)$ continuamente diferenciable (todas sus derivadas son continuas) y con *soporte compacto* (acotada), entonces $f(t) = 0$.

BORRADOR

De acuerdo con este teorema, y suponiendo que $\eta(t)$ es continuamente diferenciable y acotada, la función entre paréntesis la ecuación integral (1.33) es:

$$\frac{\partial L}{\partial f} - \frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial L}{\partial \dot{f}} = 0 \tag{1.34}$$

Esta ecuación es una versión particular (para funciones de una sola variable) de la que se conoce en la historia como la **ecuación de Euler-Lagrange** y que será de importancia central en este libro.

Volviendo a nuestro problema original, es decir, encontrar la curva con la menor longitud entre dos puntos, y reconociendo que:

$$L(f(t), \dot{f}(t), t) = \sqrt{1 + |\dot{f}(t)|^2},$$

Entonces $\partial L/\partial f = 0$ (no aparece el símbolo f en la fórmula de L) y $\partial L/\partial \dot{f} = \dot{f}/\sqrt{1 + |\dot{f}(t)|^2}$. De allí, la ecuación de Euler-Lagrange (1.34) en este problema se convierte en:

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\dot{f}}{\sqrt{1 + |\dot{f}(t)|^2}}\right) = 0$$

Esta ecuación significa que el término entre paréntesis es constante. Después de un poco de algebra, la expresión resultante, se puede integrar para obtener:

$$f(t) = At + B,$$

donde A, B son constantes.

La respuesta final a la pregunta original es ahora clara: la curva más corta entre dos puntos en el plano euclidiano es una línea recta.

Algoritmos en el cálculo variacional

Si el cálculo variacional es poco común en los textos básicos de cálculo infinitesimal, los algoritmos relacionados con él son aún más escasos en los textos de análisis numérico.

Dada la importancia del cálculo variacional en la mecánica nos detendremos un momento aquí para explorar desde la algoritmia, al menos la solución al problema de cálculo variacional que expusimos en la sección anterior: el cálculo de la curva más corta entre dos puntos en el plano euclidiano.

Para ello escribamos primero la rutina que servirá en nuestro caso como funcional (y que implementa la Ec. 1.30):

(Algoritmo 1.4)

```
def funcional_integral(f0,eta,epsilon,a,b,**opciones_de_f0):

    #Definimos las función con su variación
    f=lambda t:f0(t,**opciones_de_f0)+epsilon*eta(t)

    #La derivada de f la calculamos con derivative
    from scipy.misc import derivative
```

BORRADOR

```
    dfdt=lambda t:derivative(f,t,0.01)

    #Este es el integrando del funcional
    from numpy import sqrt
    L=lambda t:sqrt(1+abs(dfdt(t))**2)

    #El funcional es la integral definida del integrando
    from scipy.integrate import quad
    integral=quad(L,a,b)
    longitud=integral[0]

    return longitud
```

Nótese que un *funcional* en el lenguaje de la algoritmia es una rutina que recibe como parámetros otras rutina (en este caso `f0` y `eta`) y devuelve un valor numérico (en este caso `longitud`.)

La rutina en el Alg. (1.4), si bien parece compleja, recoje todos los elementos que hemos aprendido en esta sección: los parametros opcionales de una rutina expresados como `**opciones_de_f0` y que vimos en una nota de la Sección 1.1.3, las funciones `lambda` que vimos en la misma sección, la derivada numérica calculada usando `derivative` que conocimos en la Sección 1.1.4 y la integral por cuadraturas usando `quad` de la Sección 1.1.7.

Más importante aún es el hecho que esta rutina puede usarse para cualquier funcional que se exprese como una integral definida de la forma de la Ec. (1.31). Para adaptarla a otras situaciones, simplemente se debe cambiar la función `L`. En la sección de problemas al final de este capítulo se pone a prueba esta rutina en otros contextos.

Supongamos ahora que queremos calcular la curva más corta que une los puntos del plano cartesiano $(0,0)$ y $(\pi,1)$ (es decir $a=0$ y $b=\pi$). Para ello proponemos una función de referencia $f_0(t)=(t/\pi)^n$. Esta función para por ambos puntos para todo n. Como función de plantilla $\eta(t)$, que debe ser una función acotada de acuerdo al lema fundamental del cálculo de variaciones, usaremos la función trigonométrica seno (que cumple la condición $\eta(a)=\sin 0=0$ y $\eta(b)=\sin\pi=0$).

El siguiente algoritmo implementa estas elecciones:

```
#Intevalo entre los puntos
from numpy import pi
a=0
b=pi

#Funcion de referencia
def curva(t,n=1):
    return (t/pi)**n

#Función plantilla
from numpy import sin
eta=sin
```

Para ilustrar el uso de la rutina en el Alg. (1.4), calculemos la longitud de arco para el caso en el que $n=2$ y $\epsilon=0{,}5$:

```
n=2
If=funcional_integral(curva,eta,0.5,a,b,n=n)
```

```
I[f] = 3.337162809417341
```

Para encontrar la trayectoria más corta entre los puntos seleccionados, debemos minimizar una función del tipo `longitud_arco(epsilon)` que llame a la rutina `funcional_integral`, pero que solo dependa de la variable que queremos minimizar, es decir de `epsilon`. Para ello podemos definir la función `lambda`:

```
longitud_arco=lambda epsilon:funcional_integral(curva,eta,epsilon,
                                                a,b,n=n)
```

La minimización, finalmente, se consigue usando la rutina `minimize` del paquete `SciPy`, capaz de encontar el mínimo de funciones escalares con un número arbitario de variables. Lo único que necesita `minimize` para lograr su cometido es que le pasemos una rutina que tenga un solo parametro, en nuestro caso `longitud_arco` y un valor de prueba para la variable independiente (en nuestro caso usaremos $\epsilon = 0$):

```
from scipy.optimize import minimize
solucion=minimize(longitud_arco,0.0)
```

```
Resultado de la minimización:
      fun: 3.2975722013512403
 hess_inv: array([[0.73687233]])
      jac: array([1.1920929e-06])
  message: 'Optimization terminated successfully.'
     nfev: 12
      nit: 3
     njev: 4
   status: 0
  success: True
        x: array([0.25801323])
```

Nótese que el resultado de la rutina `minimize` es un *objeto* entre cuyos atributos se encuentra el valor de la variable independiente `x` que hace mínima la función de nuestro interés, en este caso `longitud_de_arco`.

Puesto en términos de nuestro problema el resultado anterior indica que para curvas del tipo $f_0(t) = (t/\pi)^2$, que sufren variaciones con una función plantilla $\eta(t) = \sin t$, la curva de mínima longitud entre el punto $(0,0)$ y el punto $(0,\pi)$, corresponde a una variación con $\epsilon = 0{,}258$.

Hagamos un gráfico de la función resultante y de su comparación con la solución analítica:

(Algoritmo 1.5)

```
import matplotlib.pyplot as plt
plt.figure()

from numpy import linspace,pi
ts=linspace(0,pi)
```

BORRADOR

```
#Valor de epsilon proveniende de la minimización
epsilon=solucion.x[0]

plt.plot(ts,curva(ts,n=n),'r.',
         label=f"Curva de referencia")
plt.plot(ts,curva(ts,n=n)+epsilon*eta(ts),'b-',
         label=f"Curva variada con $\epsilon$={epsilon:g}")
plt.plot(ts,curva(ts,n=1),'k--',
         label=f"Línea recta")

plt.legend();

#--hide--
plt.xlabel("t");
```

ver Figura 1.5

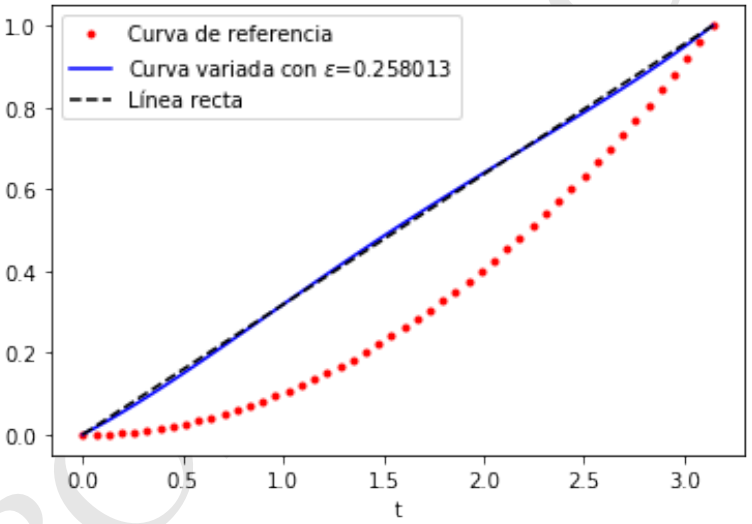


Figura 1.5: *Figura correspondiente al código 1.5. La curva continua indica una aproximación numérica al camino más corto entre los puntos $(0,0)$ y $(0,\pi)$ del plano euclidiano, encontrada al minimizar el funcional longitud de arco y usando como función de prueba $f_0 = (t/\pi)^n$ (linea punteada) y como función plantilla $\epsilon(t) = \sin t$. El valor de ϵ que corresponde a la solución se muestra en la etiqueta. Para comparación se muestra (linea rayada) la solución exacta, que corresponde a una línea recta.*

1.1.11. Gráficos interactivos

Para ver los gráficos interactivos use a las libretas de `Jupyter` que que están disponibles en la versión electrónica del libro.

Bibliografía

[1] T. M. APOSTOL, *CALCULUS volume I One-Variable Calculus, with an Introduction to Linear Algebra, Blaisdell Publishing Company*, John Wiley & Sons, 1967.

[2] T. M. APOSTOL, *Calculus, Volume II: Multi-Variable Calculus and Linear Algebra, with Applications to Differential Equations and Probability*, John Wiley & Sons, 1969.

[3] A. C. HINDMARSH, *Odepack, a systematized collection of ode solvers*, Scientific computing, (1983), pp. 55–64.

[4] W. H. PRESS, S. A. TEUKOLSKY, W. T. VETTERLING, AND B. P. FLANNERY, *Numerical recipes 3rd edition: The art of scientific computing*, Cambridge university press, 2007.

BORRADOR