



UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

---

---

FACULTAD DE CIENCIAS

TÍTULO DEL TRABAJO

T E S I S

QUE PARA OBTENER EL TÍTULO DE:

GRADO A OBTENER

P R E S E N T A :

NOMBRE DEL ALUMNO

TUTORA

GRADO Y NOMBRE



FACULTAD DE CIENCIAS  
UNAM

2021



## Hoja de datos del jurado

1. Datos del alumno

ApellidoP

ApellidoM

Nombres

teléfono

Universidad Nacional

Autónoma de México

Facultad de Ciencias

Carrera

Cuenta

2. Datos del tutor

Grado

Nombres

ApellidoP

ApellidoM

3. Datos del sinodal 1

Grado

Nombres

ApellidoP

ApellidoM

4. Datos del sinodal 2

Grado

Nombres

ApellidoP

ApellidoM

5. Datos del sinodal 3

Grado

Nombres

ApellidoP

ApellidoM

6. Datos del sinodal 4

Grado

Nombres

ApellidoP

ApellidoM

7. Datos del trabajo escrito

Título

Subtítulo

número de páginas

año



---

# Índice general

<b>1</b>	<b>Lo mínimo necesario</b>	<b>1</b>
1.1.	Estructura . . . . .	1
1.2.	El archivo principal . . . . .	2
<b>2</b>	<b>Un poco de memoir</b>	<b>3</b>
2.1.	Lo básico . . . . .	3
2.2.	Un poco más . . . . .	4
<b>3</b>	<b>Fuentes</b>	<b>7</b>
<b>4</b>	<b>Paquetes en la clase</b>	<b>9</b>
4.1.	amsmath y mathtools . . . . .	9
4.2.	amsthm . . . . .	10
4.3.	babel y polyglossia . . . . .	11
4.4.	microtype . . . . .	12
4.5.	siunitx . . . . .	13
4.6.	csquotes . . . . .	13
4.7.	biblatex . . . . .	13
<b>5</b>	<b>Glosarios</b>	<b>17</b>
<b>6</b>	<b>Compilación y arara</b>	<b>19</b>
	<b>Índice de términos</b>	<b>23</b>
	<b>Lista de símbolos</b>	<b>25</b>
	<b>Acrónimos</b>	<b>27</b>
	<b>Lista de Nombres</b>	<b>29</b>
	<b>Referencias</b>	<b>31</b>
	<b>Bibliografía</b>	<b>33</b>



## Lo mínimo necesario

Para usar la clase `tesisFC` sólo se necesitan los archivos:

- `tesisFC.cls`,
- `fc.pdf`,
- `unam.pdf`,
- `hojaS.tex`

El resto de los archivos en este repositorio son para hacer ejemplos, tanto de la estructura de un documento “grande” como el uso de algunos paquetes. Además, se incluye un archivo, `plantilla.tex`, que puede servir para usarse en la creación de una tesis.

### 1.1. Estructura

No escribir toda la tesis en un sólo archivo ya que editar y encontrar cosas puede ser complicado. Para hacer una tesis se recomienda una estructura similar a la siguiente:

```
carpetaTesis
├── tesisFC.cls
├── fc.pdf
├── unam.pdf
├── hojaS.tex
├── plantilla.tex
├── archivoCap1.tex
├── archivoCapn.tex
├── archivoAp1.tex
├── archivoApn.tex
├── archivoRef.bib
└── posiblemente algunos más para glosarios
```

## 1.2. El archivo principal

En este repositorio hay un archivo llamado `guia.tex` que funciona como el archivo principal de una tesis. También hay un archivo `plantilla.tex` el cual se puede usar como archivo principal y tiene la cualidad de “estar vacío”. Así, será más fácil de usar que `guia.tex`.

En el archivo `plantilla.tex` se ha dividido el preámbulo. Este tiene una sección para paquetes, otra para lo necesario para hacer una bibliografía con `biblatex`, una para la información que irá en la portada y una más para la definición de comandos, funciones y demás cosas útiles.

Luego el cuerpo del documento en el archivo `plantilla.tex` incluye lo necesario para crear la portada, la hoja de sinodales, la tabla de contenidos y la bibliografía.

Se recomienda escribir el contenido de cada capítulo en un archivo diferente y luego incluirlo en `plantilla.tex` usando el comando `include`, siguiendo la estructura lógica del documento. Por ejemplo,

```
...
\begin{document}
...
\mainmatter
\include{introduccion.tex}
\include{loprincipal.tex}
...
\appendix
\include{algosecundario.tex}
...
\end{document}
```

Finalmente este archivo tiene dos comandos para crear una bibliografía, que se complementan con lo que están en el preámbulo. El primero de ellos es

```
\printbibliography[title={Referencias},category=cited]
```

Su función es imprimir todas aquellas referencias que sí fueron citadas en el texto, bajo el nombre de “Referencias”. Luego el comando

```
\printbibliography[title={Bibliografía},notcategory=cited]
```

imprime las referencias que no fueron citadas bajo el nombre de “Bibliografía”.

La parte de la bibliografía se hizo de esa manera ya que en el instructivo de tesis digital de la facultad de ciencias se especifica esta división de las referencias.



## Un poco de memoir

### 2.1. Lo básico

La clase `tesisFC` tiene como base a *memoir*. La motivación es poder configurar de manera fácil los aspectos del documento sin tener que usar paquetería adicional. Así, cuando el usuario cargue un paquete es poco probable que cause alguna incompatibilidad.

*Memoir* es una clase configurable, es decir, no hace falta cargar paquetes adicionales para hacer un diseño completo de cómo se verá nuestro documento. Además puede servir tanto para *book* como para *article* la opción por defecto es una salida estilo *book*, para cambiar al diseño estilo *article* basta poner la opción `article` como opción a la clase. A diferencia de las clases estándar donde hacer un cambio de *book* a *article* seguramente causará problemas (por ejemplo `\chapter{...}` no está definido en *article*), en *memoir* no existe ese problema. Otra ventaja inmediata es que el ambiente `abstract` estará definido para *book*.

La tabla de contenidos esta formada por el comando `\tableofcontents*`. Esta versión con estrella es única de *memoir* y su utilidad es que no aparezca una entrada para la tabla de contenidos en la tabla de contenidos.

Otra ventaja de *memoir* es que no hace falta redefinir `\cleardoublepage` para que las páginas pares “vacías” antes de un nuevo capítulo estén en blanco. La clase ya lo hace por defecto. Además, la clase tiene las opciones `openright` para que los capítulos empiecen en las paginas impares, `openleft` para que empiecen en las pares y `openany` para que empiecen en la siguiente página sin importar su paridad.

*Memoir* tiene predefinidos muchos estilos de capítulo, en este documento se está usando `madsen`. Además tiene definidos muchos más estilos de salida por defecto, estos pueden verse en <http://www.ebookation.com/wp-content/uploads/2010/03/memoirchapstyles.pdf>

Modificamos el tamaño y la posición de la caja de texto para obtener una línea de texto suficientemente grande como evitar tantos cortes de palabra, pero no tan grande como para que sea difícil pasar de una línea a otra.

Si notan como está formado un libro, el bloque de texto no está centrado en la página. Comúnmente el margen de la espina (donde se juntan las páginas de un libro) es la mitad del margen de la orilla (el opuesto a la espina). Lo mismo sucede con los márgenes superior e inferior, donde el superior es más chico que el inferior. Tomamos en cuenta estos detalles en la formación de este documento.

También cambiamos cabeceras y pies, esta vez fue mínimo pero aún así es suficiente para ver cómo cambiar cabeceras y pies (ver archivo `.cls`).

Mencionamos de nuevo que *memoir* es una clase configurable y tiene de manera nativa capacidades mayores o iguales a las de los siguientes paquetes:

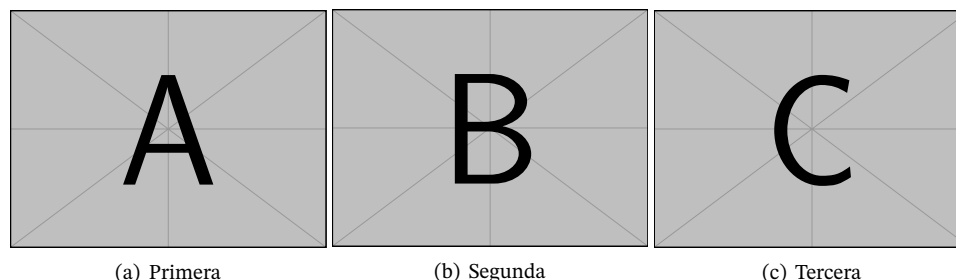


Figura 2.1: Muchas figuras

abstract	chnpage	index	nextpage	shortvrb	tocbibind
appendix	enumerate	makeidx	parskip	showidx	tocloft
booktabs	epigraph	moreverb	patchcmd	titleref	verbatim
ccaption	framed	needspace	setspace	titling	verse
chngcntr	ifmtarg	newfile			

También tiene las capacidades, aunque con comandos diferentes, de los siguientes paquetes

crop, fancyhdr, geometry, sidecap, subfigure, titlesec.

Por último, la clase carga los siguientes paquetes

array, dcolumn, delarray, etex, iftex, tabularx, textcase (con la opción `overload`)

Así que no hace falta cargar ninguno de estos, de esta manera es más fácil tener compatibilidad de paquetes.

## 2.2. Un poco más

Primero veamos cómo hacer subfiguras con memoir, es decir, sin usar el paquete `subfigure` o `subcaption`. Primero necesitamos escribir

```
\newsubfloat{figure}
```

en el preámbulo. Este comando activará todo lo necesario para hacer subfiguras, por ejemplo en la figura 3.1 hay una forma de hacerlas usando el comando `\subcaption{...}` para poner los subtítulos correspondientes. También es posible hacer como en el ejemplo de la figura 2.1. Nota que no hay archivos aparte para las figuras que usamos. Estas ya están instaladas con nuestra distribución y podemos usarlas como lo hicimos. Además como su nombre lo indica, un *float* está flotando en la página y  $\text{\LaTeX}$  decidirá cual es el mejor lugar para ponerlo. Otro ejemplo de flotante es una tabla. Las opciones para la ubicación de flotantes son las siguientes.

**h** trata de poner el flotante donde fue creado.

**t** lo pone en la parte superior de la página.

**b** lo pone en la parte inferior de la página.

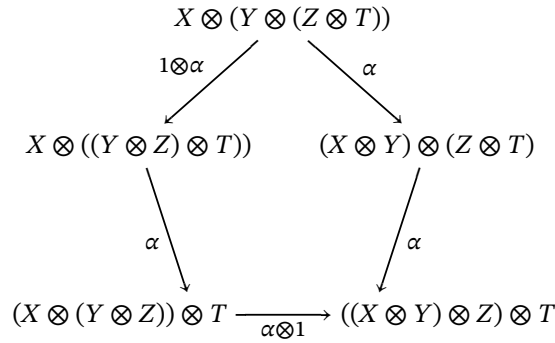


Diagrama 1: ¡Pentagonator!

**p** lo pone en una página especial para flotantes.

**h!** se esfuerza más en poner el flotante donde fue creado.

La sintaxis para especificar la posición es `\begin{figure}[...]`.

También es posible crear nuevos flotantes, nosotros hemos hecho un flotante para diagramas en el preámbulo. Así podemos poner diagramas en una versión similar a la de figura. También es posible aprovechar el espacio del margen como apoyo para las figuras. Esto es recomendable sólo en el caso cuando haya suficiente espacio en el margen. Por ejemplo, en la versión para imprimir no hay espacio en el margen, así que, con la opción `imp` se redefinió el ambiente `marginfigure` para que se convierta en un `figure`.

**Nota.** En la versión para imprimir la tesis como un libro el margen se reduce para permitir una caja de texto más grande, así las cosas que estén en el margen podrían verse muy “apretadas”. Por esta razón se quitaron automáticamente todas las figuras del margen.

Por último en el código de este documento se puede ver que todas las figuras tienen inmediatamente después del inicio del ambiente el comando `\centering`, debe ser este y no usar el ambiente `center` para evitar espacios verticales no deseados. Además, como esto se hizo en cada figura pudo haberse configurado en el preámbulo con el siguiente comando

`\setfloatadjustment{figure}{\centering}`

que también se puede usar en tablas, figuras al margen y los flotantes que se hayan creado.



Figura 2.2: Un círculo en el margen



## Fuentes

Para las fuentes del documento he escogido los paquetes `fontspec` y `unicode-math` en el caso de compilar con `Lua $\TeX$` . Si la compilación es con `pdf $\TeX$`  se usará el paquete `fontenc`.

La ventaja de `fontspec` y `unicode-math` es que posible cambiar fuentes de manera fácil en partes del documento, pero deberá ser compilado con `Xe $\TeX$`  o con `Lua $\TeX$` , el último siendo el método recomendado. Con esto se podrán usar caracteres unicode en cualquier parte de texto, por ejemplo en una ecuación centrada (para que aparezcan las ecuaciones compila con `Lua $\TeX$` .)

$$\forall \varepsilon > 0 \exists \delta > 0 (|x - y| < \delta \Rightarrow |f(x) - f(y)| < \varepsilon)$$

o en una línea de texto  $\forall \varepsilon > 0 \exists \delta > 0 (|x - y| < \delta \Rightarrow |f(x) - f(y)| < \varepsilon)$ . Las letras griegas, por ejemplo, no necesitan estar en modo matemático para funcionar  $\alpha\beta\gamma\delta\varphi\psi$  (ver código fuente ignorando los condicionales para cada tipo de compilación).

La fuente que fue escogida para este texto es *Stix Two*. Esta fuente tiene como objetivo servir como un estándar para la preparación, publicación e impresión de textos científicos. Es impulsada y usada por la *AMS* (matemáticas), *ACS* (química), *AIP* y *APS* (física), *IEEE* (ingeniería) y Elsevier. Por lo tanto, tiene un conjunto de símbolos sumamente extenso.

Para nuestros ejemplos, con `Lua $\TeX$` , se cambiara el tipo de fuente sans por GFS Neohellenic. No es recomendable usar muchos tipos de fuente en un documento, pero como esto es un ejemplo haremos dicho cambio. Como distintas fuentes tienen distintos tamaños, al combinar dos de ellas es muy posible crear una inconsistencia en los tamaños. Para evitar esto use la opción de `fontspec` (ver preámbulo) `Scale=MatchUppercase`. Este paquete puede hacer muchas modificaciones a los atributos de una fuente y no veremos más de esos atributos. En el preámbulo está la definición de un ambiente donde la fuente se cambia:

**Este texto está en sans** `\textsf{I}` Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris.

$$\prod_{i \in I} A_i \neq \emptyset$$

Como el objetivo fue cambiar la fuente sans el cambio también se hará al usar `\textsf{...}` y `\sffamily ...` como podemos ver a continuación cambio de fuente  $\forall \varepsilon > 0 \exists \delta > 0 (...)$ . Como puede verse en el ejemplo lo que hace cambiar la fuente matemática es el comando `\mathversion{sansmath}` de `unicode-math`.

Un ejemplo de dónde se hizo este tipo de cambios es en las *Lecturas de Física de Feynman*. En este texto las figuras llevan un tipo de fuente diferente a la del cuerpo.

Como no está pensado para que así sea la salida de todas las figuras no se hizo la definición en el preámbulo. Para dicho cambio se debe escribir `\captionnamefont{\sffamily}` para



Figura 3.1: Operaciones con vectores

cambiar el tipo de fuente de la palabra “Figura” y su número. Para cambiar el delimitador, que en este caso son dos puntos se usa `\captiondelim{: }`. Para cambiar el tipo de fuente del título de las figuras se usa `\captiontitlefont{\sffamily}`. Con esto es fácil hacer una configuración de la salida de los títulos de figuras, tablas y demás flotantes. También se podría cambiar el tamaño de la fuente o poner alguna palabra en específico.

Otra ventaja del manejo de fuentes de Xe $\LaTeX$  y Lua $\LaTeX$  es que las fuentes instaladas en nuestro sistema estarán disponibles para su uso en documentos de  $\LaTeX$ . Por ejemplo si quisiéramos que nuestro documento estuviera escrito en Arial simplemente hay que escribir

```
\setmainfont{Arial}
```

contrario a la forma en la que hace pdf $\LaTeX$  donde habría que usar el código de la fuente y en muchos casos es difícil encontrar dicho código. Además que las fuentes disponibles de esa forma son relativamente pocas. Para dar una idea de la disponibilidad de fuentes con Xe $\LaTeX$  y Lua $\LaTeX$  basta ver las fuentes disponibles en overleaf <https://www.overleaf.com/latex/examples/fonts-spec-all-the-fonts/hjrpnxhrrtxc>. Seguramente en tu sistema habrá un conjunto de fuentes disponibles grande.

Un comentario acerca de `unicode-math` es que permite usar una fuente específica para cada alfabeto. El ejemplo en este texto es cambiar la fuente script, que tanto en *Stix two* como muchas otras fuentes es igual a la caligráfica. Con `unicode-math` es suficiente cargar una fuente con el alfabeto que nos guste, sólo para el rango que la queremos. Aún cuando en un principio no es necesario tener tantos alfabetos matemáticos diferentes, en teoría de categorías he visto una notación que aunque no es estándar (creo que aún no hay ningún estándar) parece dar coherencia. Las categorías pequeñas y localmente pequeñas se denotan con letras en negritas **A**, **Con**,... categorías más grandes se denotan con letras caligráficas  $\mathcal{X}$ ,  $\mathcal{Y}$ , etc. y categorías especiales como topós se denotan con letras script como  $\mathcal{E}$ .

En el código se puede notar que se han usado diferentes sintaxis para los alfabetos, que `unicode-math` se encargará de mapear al símbolo correcto. Por ejemplo, las formas de obtener la letra “A” en negritas es como símbolo definido por un alfabeto `\mbfA`, como comando de `unicode-math` (es la forma recomendada) `\sympf{A}` o como en la forma “tradicional” `\mathbf{A}`.

Algo que hay que notar es que el soporte de rango y versiones en `unicode-math` es aún experimental, por lo que para que funcione como en este ejemplo se debe escribir primero la versión `\setmathfont{GFS Neohellenic Math}[version=sansmath]` y luego el rango `\setmathfont{XITS Math}[range=scr]`. De lo contrario GFS Neohellenic reescribiría el rango y no lograríamos lo que se quería mostrar.

## Paquetes en la clase

### 4.1. `amsmath` y `mathtools`

Aunque en el título de la sección se menciona `amsmath` en la clase sólo se carga el paquete `mathtools`. Esto se debe a que `mathtools` carga al paquete `amsmath`, así lo podemos pensar como una extensión. De manera más precisa `mathtools` es resultado de corregir algunos bugs de `amsmath` y añadir algunas otras funciones.

Un ejemplo específico de estos paquetes es la creación de operadores. En el modo matemático de  $\text{\LaTeX}$  una letra representa una variable y dos variables juntas representa la multiplicación de estas. De esta manera  $xya$  representa la multiplicación de las variables  $x$ ,  $y$  y  $a$ , mientras que  $xya$  representa al operador  $xy$  aplicado a la variable  $a$ . En otras palabras, un cambio en el alfabeto debería implicar un cambio en el significado. Por esta razón deberíamos escribir operadores de una forma especial y no sólo escribiendo su nombre en modo matemático.

Por ejemplo, para escribir el supremo del conjunto  $A$  se debe escribir `\sup A` y su resultado es  $\sup A$ , de donde es claro que los operadores están en letras *upright* (redondas en español). Ya están definidos los operadores más comunes, pero si se quisiera definir uno nuevo se escribe en el preámbulo `\DeclareMathOperator{\idem}{Idem}` para definir, por ejemplo, los idempotentes de un anillo, así `\idem(R)` genera  $\text{Idem}(R)$ .

Además `amsmath` define ambientes matemáticos como `align`, `gather`, `cases`, `equation`, `array`, las variantes de `matrix`, etc. y `mathtools` corrige algunos errores en ellos, por ejemplo en `gather` o añade algunas opciones, por ejemplo en los ambientes de matriz se puede especificar la alineación de las columnas de la misma manera que en `array`.

Una función de `mathtools` que no tiene `amsmath` es la creación de delimitadores que se pueden ajustar al tamaño del contenido, por ejemplo para hacer un delimitador para el valor absoluto se escribe `\DeclarePairedDelimiter\abs{\lvert}{\rvert}` este tiene un argumento más que el operador ya que hay que decir qué símbolo “abre” y qué símbolo “cierra”. No es necesario que concuerde un símbolo que abre con su pareja de cierre, como los paréntesis. Por ejemplo se podría definir un delimitador para intervalos abiertos por la izquierda y cerrados por la derecha. La deferencia de estos delimitadores se puede apreciar con su versión con estrella y con el código

$$\begin{array}{llll}
 |\sum_{i=1}^n a_i| & \left| \sum_{i=1}^n a_i \right|, & \lvert \sum_{i=1}^n a_i \rvert & \left| \sum_{i=1}^n a_i \right| \\
 \abs{\sum_{i=1}^n a_i} & \left| \sum_{i=1}^n a_i \right|, & \abs*{\sum_{i=1}^n a_i} & \left| \sum_{i=1}^n a_i \right|
 \end{array}$$

Más aún, no se debe usar las barras comunes para el valor absoluto como en  $|x|$  ya que no tienen el contexto adecuado. Cuando se definen como delimitadores, `mathtools` les agrega un `\mathopen{}` y `\mathclose{}` como corresponde. La función de los dos comandos anteriores

se puede apreciar en el siguiente ejemplo

```
|x|      |x|    |-x|      |-x|
\abs{x}  |x|    \abs{-x}  |-x|,
```

donde claramente el espaciado en `|-x|` es pésimo. El mal espaciado se debe a que  $\text{\LaTeX}$  reconoce a `-` como un operador binario y da un poco de espacio entre los símbolos a su derecha e izquierda. En cambio, cuando se le ha agregado `\mathopen{}` al delimitador, como sucede en nuestra definición de `\abs`,  $\text{\LaTeX}$  ahora toma a `-` como operador unario y no agrega espacio a la izquierda.

`\DeclarePairedDelimiter` tiene una versión extendida para que pueda tomar argumentos, como `\newcommand{ } [ ] { }`. Esta versión extendida se escribe `\DeclarePairedDelimiterX` y con ella podemos hacer, por ejemplo, conjuntos con delimitadores y la línea de “tal que” que crezcan correctamente (ver su definición en `ejemplo.tex`)

$$\left\{ x \in X \left| \frac{\sqrt{x}}{x^2 + 1} > 1 \right. \right\}$$

## 4.2. amsthm

Este paquete provee mejoras útiles para las definiciones de teoremas ( $\text{\LaTeX}$  puede crear teoremas sin necesidad de paquetes) y define un ambiente de demostración (esto no lo hace  $\text{\LaTeX}$ ). Con este paquete se pueden usar y definir estilos de teoremas de forma fácil. En la clase se definieron los siguientes ambientes:

<code>definicion</code>	<code>teorema</code>	<code>corolario</code>
<code>lema</code>	<code>proposicion</code>	<code>observacion</code>

Con la salida esperada del nombre del ambiente. Por ejemplo:

**Definición 4.1.** Una función  $f : X \rightarrow Y$  es continua si para cualquier abierto  $V \subseteq Y$  se tiene que  $f^{-1}(V) \subseteq X$  es abierto.

**Teorema 4.2** (Fermat). La ecuación  $x^n + y^n = z^n$  con  $n \geq 3$  no tiene soluciones no triviales en  $\mathbb{Z}$ .

**Demostración.** He descubierto una demostración maravillosa de esto, que este margen es demasiado estrecho para contener.  $\square$

Se modificó el estilo del ambiente demostración para que su salida sea similar a la de los resultados, una demostración es tan importante como el enunciado. Nuestra redefinición del ambiente de demostración se basa en la definición de `amsthm` así tiene lo necesario para comportarse bien con el símbolo  $\square$ . Es decir, cuando se termina una demostración con una ecuación u otro tipo de ambiente habrá un espacio vertical no deseado entre el final del texto y el símbolo de fin de la demostración. Este espacio se evita con el comando `\qedhere`, pero si el ambiente no cuenta con la definición correcta seguirá apareciendo este espacio vertical. Un ejemplo de cómo funciona `\qedhere`, para mostrar la diferencia se han hecho dos demostraciones iguales

**Teorema 4.3.** *Un resultado importante.*

**Demostración.** Se sigue de la siguiente ecuación (sin usar `\qedhere`)

$$\sum_{n \geq 1} \frac{1}{n} = -\frac{1}{12}.$$



□

¿Es otra demostración? Se sigue de la siguiente ecuación (usando `\qedhere`)

$$\sum_{n \geq 1} \frac{1}{n} = -\frac{1}{12}.$$

□

Cada “teorema” nuevo crea un contador. En este ejemplo la cuenta de teoremas se reiniciará al iniciar un nuevo capítulo, este es el efecto del comando opcional `[chapter]` en la definición de `teorema`. Además, el contador de las definiciones será el mismo que el de los teoremas (notar que aparece definición 4.1, teorema 4.2), esto lo hace el parámetro opcional `[teorema]` en la definición de `definicion`. Esta cuenta de teoremas sirve para hacer referencia a resultados o definiciones en el futuro. Al teorema le pusimos una etiqueta con el comando `\label{teo:fermat}` que luego se puede hacer referencia con `\ref{teo:fermat}`. Una buena práctica es separar la referencia de la palabra anterior con un espacio irrompible `~`, de esta forma cuando escribimos “por el teorema 4.2”  $\text{\LaTeX}$  no podrá romper un renglón dejando la palabra “teorema” en un renglón y el número “4.2” en otro.

Nota que el estilo `plain` pone el cuerpo del teorema en itálicas mientras que el estilo `definition` no, como puede verse en el enunciado de la definición y de los teoremas.

En el preámbulo de `ejemplo.tex` está un ejemplo de cómo crear un estilo nuevo de “teorema”. En este se creó un ambiente para los axiomas usando un estilo diferente que llevará una cuenta independiente de las definiciones anteriores, pues no aparece la opción `[teorema]` en su definición.

**Axioma.** Para toda  $f : D \rightarrow R$  existe una y sólo una  $b \in R$  tal que para cualquier  $d \in D$

$$f(d) = f(0) + d \cdot b.$$

Adicionalmente, se podría hacer las definiciones de estos ambientes de `teorema` y `demonstración` como un nuevo ambiente. El ejemplo, compilando con  $\text{\Lua\TeX}$ , con tipo de letra sans en 1 tiene lo necesario para crear un ambiente y contador con las características como las de los ambientes de `amsthm`.

Además, junto con `amsthm` se podría usar el paquete `thmtools` que permite hacer estilos de teoremas, repetirlos, hacer una lista de teoremas, entre otros. La documentación de `thmtools` es fácil de leer ya que consta de ejemplos, así es posible ver la salida de los comandos que define.

Finalmente, otro paquete común para el manejo de teoremas y demostraciones es `ntheorem`. Cada uno tiene sus ventajas y desventajas y no es fácil elegir uno sobre otro. En este documento se eligió `amsthm` porque es (posiblemente) el más común.

### 4.3. babel y polyglossia

Para el soporte de idiomas elegimos `babel` ya que tiene mayor soporte que otros paquetes similares, como `polyglossia`.

`babel` se carga con las opciones `spanish` y `mexico`. La opción `mexico` hace una localización más similar a la que se usa en México, como su nombre lo indica. Entre las cosas que hace podemos mencionar que cambia el nombre “cuadro” por “tabla”, prioriza comillas y usa el punto decimal en lugar de la coma como puede verse en la compilación con  $\pi = 3.141592 \dots$  Además se cargó la opción `es-noindentfirst` para evitar el sangrado en el primer párrafo después de un

título. Por último se deshabilitaron los shorthands o taquigrafías para evitar algunos conflictos con la sintaxis de otros paquetes, por ejemplo las comillas en `xy-pic` o en `tikzcd`.

Al usar el idioma español `babel` se encargará de traducir todo, por ejemplo la palabra “capítulo” o “figura” como se puede ver en el documento. También traduce y acentúa los operadores, por ejemplo  $\max A$  o  $\lim f$ . Pero en algunos casos se decidió crear un comando nuevo como en el caso del seno:

$$\sin(\alpha) \neq \text{sen}(\alpha)$$

Otra opción para el manejo de idiomas en  $\text{Xe}\text{L}\text{A}\text{T}\text{E}\text{X}$  o  $\text{Lua}\text{L}\text{A}\text{T}\text{E}\text{X}$  es `polyglossia`. Este paquete se creó cuando `babel` había dejado de tener mantenimiento y uno de sus objetivos es simplificar su trabajo en estos motores. Este paquete también tiene una variante `mexico` (no compatible con `biber`), además de que se puede elegir un poco más el comportamiento de los operadores con las opciones:

`accented` para acentuar los operadores como `babel`.

`spaced` para dar un espacio corto entre el operador y el objeto al que se le aplica, de nuevo, como `babel`.

`all` para hacer las dos anteriores más la localización de `\sin \tan \sinh` y `\tanh`.

`none` para no hacer ninguna localización.

Con  $\text{Lua}\text{L}\text{A}\text{T}\text{E}\text{X}$  se podría cargar `polyglossia` con la siguiente configuración

```
\usepackage{polyglossia}
\setdefaultlanguage[spanishoperators=all]{spanish}
```

De nuevo, para evitar el sangrado del primer párrafo después de un título se debería añadir el comando

```
\PolyglossiaSetup{spanish}{indentfirst=false}
```

Tanto `babel` como `polyglossia` son buenas opciones para el manejo de idiomas y ambos tienen ventajas y desventajas en algunos aspectos sobre el otro. Aún así elegimos usar `babel` por ser más conocido y tener un mejor soporte.

#### 4.4. microtype

Este paquete habilita los aspectos micro-tipográficos del documento. Algunos de ellos ya son hechos por  $\text{T}\text{E}\text{X}$  como justificación y la separación silábica (cortes de palabra). En general las opciones que se le pasaron a `microtype` sirven para calcular la expansión de letras, palabras y permitir que algunos caracteres cortos, como el guión de un corte de palabra o el punto, salgan del margen. Algunas de estas mejoras se pueden hacer con `fontspec`, por ejemplo poniendo la opción `LetterSpacing` al cargar una fuente, pero hay que calcular las cosas manualmente. En cambio `microtype` toma todas estas decisiones por nosotros y lo hace de acuerdo al idioma que se la haya pasado a `babel` o `polyglossia`. El resultado es un pdf que luce tipográficamente mejor.

## 4.5. siunitx

Este paquete tiene como objetivo implementar la escritura de cantidades físicas de acuerdo con las reglas del sistema internacional de medidas.<sup>1</sup> Aunque hay un desacuerdo con la forma de espaciar las cantidades y las unidades debido a la mala traducción del francés (idioma de la versión oficial del manual del sistema internacional de medidas) al inglés (idioma de donde se basaron para la creación del paquete). En el francés dice que deberían separarse con un espacio y en el inglés dice que se separan con un *thin space*. Así, el espaciado no es el correcto pero el error puede ser productivo ya que un menor espaciado crea una relación más estrecha entre cantidad y unidad. Algunos ejemplos de este paquete están en la siguiente tabla:

Comando	Resultado
<code>\ang{1;2;3}</code>	$1^{\circ}2'3''$
<code>\unit{\gram\per\cubic\centi\metre}</code>	$\text{g cm}^{-3}$
<code>\unit{kg.m/s^2}</code>	$\text{kg m/s}^2$
<code>\qty{.23e7}{\candela}</code>	$0.23 \times 10^7 \text{ cd}$
<code>\qty[mode=text]{1.23}{J.mol^{-1}.K^{-1}}</code>	$1.23 \text{ J mol}^{-1} \text{ K}^{-1}$
<code>\qty[per-mode = symbol]{1.99}{\per\kilogram}</code>	$1.99/\text{kg}$
<code>\qty[per-mode=fraction]{1,345}{\coulomb\per\mole}</code>	$1.345 \frac{\text{C}}{\text{mol}}$

## 4.6. csquotes

Este paquete no fue cargado por defecto en la clase pero es recomendado para la bibliografía, sobretodo si se usa `biblatex`, como en nuestro ejemplo.

Este paquete provee algunas facilidades en citas y ajustar las comillas al idioma que se quiera. En este documento se uso junto con la opción `style=mexican` que carga el estilo de comillas del idioma español con la variante para México. También define comandos para poner comillas y hacer citas, por ejemplo `\enquote{comillas}` resulta en “comillas”. Este comando se puede anidar para poner las comillas correctas dentro de otras comillas, por ejemplo

```
\enquote{Lorem ipsum \enquote{dolor} sit amet}
```

resulta en “Lorem ipsum ‘dolor’ sit amet”. Puede ser útil escribir comillas mediante un comando, ya que muchas veces no se usan las comillas correctas de  $\text{\LaTeX}$ , estas son ``...'. Además, da facilidades para poner citas textuales con el comando `\textquote[cita][puntuación]{texto}`. También tiene un comando para citar un bloque de texto `\blockquote[cita][puntuación]{texto}`. Todas estas funciones tienen una versión para citar en un idioma extranjero, este idioma también debe ser cargado en `babel` (o `polyglossia`).

## 4.7. biblatex

En principio no se cargo por defecto ningún paquete para crear la bibliografía del documento, pero el que nos parece recomendable es `biblatex`. Hay otros paquetes para formar la bibliografía de un texto, por ejemplo `natbib` o `cite`.

<sup>1</sup><https://www.bipm.org/en/measurement-units/>

Mientras que `natbib` y `cite` son muy parecidos, sólo cambian algunas capacidades y `natbib` puede hacer técnicamente todo lo que hace `cite` más algunas cuantas cosas; `biblatex` es muy diferente a estos dos paquetes.

Como este es un texto “local” en el sentido que su código fuente no debe cumplir los requerimientos de ningún *journal* o editorial, entonces es posible usar los paquetes que cada quien considere necesario. La motivación principal para hacer este ejemplo con `biblatex` está en la siguiente liga <https://tex.stackexchange.com/a/25702/140456>, donde se explican las ventajas de `biber` sobre `bibtex`.

La construcción de la base de datos de referencias es básicamente la misma para cualquier paquete de los anteriores. Hay herramientas gráficas útiles para hacer esto como JabRef o Zotero.

El paquete `biblatex` puede manejar muchos tipos de entrada. En el manual se describe el uso de los siguientes

article	dataset	reference	conference	jurisdiction
book	manual	mvreference	masterthesis	legislation
mvbook	misc	inreference	pdhthesis	legal
inbook	online	report	techreport	letter
bookinbook	patent	set	www	movie
suppbook	periodical	software	artwork	music
booklet	suppperiodical	thesis	audio	performance
collection	proceedings	unpublished	bibnote	review
mvcollection	mvproceedings	xdata	commentary	standard
incollection	inproceedings	custom[a-f]	image	video
suppcollection				

Son demasiados para dar una descripción breve, pero en el manual de `biblatex` no sólo se describe para qué se usa cada una sino que también algunos campos recomendados para algunas de ellas.

La lista de campos que se pueden usar en las entradas es demasiado larga para escribirla aquí. De nuevo, en el manual se dan los campos disponibles junto con una descripción breve.

También tiene una lista grade de formas de citación, las más comunes son `\cite{...}`, `\parencite{...}`, `\textcite{...}` y `\footcite{...}`. El estilo de las entradas bibliográficas y de la citación se hacen mediante opciones del paquete. Como la lista de opciones también es amplia haremos referencia a la siguiente liga <http://tug.ctan.org/info/biblatex-cheatsheet/biblatex-cheatsheet.pdf> para tener una guía rápida de `biblatex`.

En este ejemplo hicimos un archivo `refs.bib` como base de datos de la bibliografía. Para “cargarlo” hay que usar el comando `\addbibresource{refs.bib}`. Este comando sólo acepta un archivo, pero se pueden cargar más escribiendo todo el comando con cada archivo `.bib` que se quiera usar. Para imprimir la bibliografía se pone el comando `\printbibliography` donde se quiera tener la bibliografía. Por ejemplo se puede imprimir una bibliografía por capítulo, por tipo (artículo, libro, etc.), por alguna palabra clave, etc.

En principio sólo imprime las entradas que fueron citadas en el texto, si se quiere imprimir una entrada que no fue citada se usa el comando `\nocite{label1,label2,...}` o si se quiere imprimir todas las entradas del archivo `.bib` se usa `\nocite{*}`.

Como en el instructivo de tesis digital de la facultad de ciencias se pide hacer dos “bibliografías”, entonces se ha incluido `\nocite{*}` en el preámbulo.

El paquete `biblatex` tiene muchas capacidades para hacer e imprimir bibliografías. Por ejemplo podría imprimir sólo los libros en una bibliografía y los artículos en otra. También es

posible hacer una bibliografía por capítulo y muchas otras cosas más.

La facultad requiere una “bibliografía” donde estén todas las referencias que sí fueron citadas en el texto, que vaya con el título “Referencias”, y otra con todas aquellas que no fueron citadas, bajo el nombre “Bibliografía”. Como este es el requerimiento de la facultad, sólo incluimos un ejemplo de cómo hacer eso y nada más.

Primero se hizo una categoría para biblatex y se incluyeron en ella todas las referencias citadas. Para que la bibliografía con las referencias no citadas no aparezca vacía se necesita usar `\nocite{*}`. Así, en el preámbulo del documento aparecen los comandos

```
\DeclareBibliographyCategory{cited}
\AtEveryCitekey{\addtocategory{cited}{\thefield{entrykey}}}
\addbibresource{refs.bib}
\nocite{*}
```

Luego, al final del documento se incluyen los comandos para imprimir las dos bibliografías. Estos son

```
\printbibliography[title={Referencias},category=cited]
\printbibliography[title={Bibliografía},notcategory=cited]
```

Para ver cómo funciona todo esto en el pdf llamado `guia.pdf` citamos algunas referencias, [[Law07](#); [JT11](#); [Joh13](#); [Mac98](#)].

Una nota final es que el proceso de compilación ahora debe ser el siguiente

```
lualatex MiDocumento.tex
biber MiDocumento
lualatex MiDocumento.tex
lualatex MiDocumento.tex
```



## Glosarios

Algo que no es tan común en las tesis, pero podría ser útil para facilitar la lectura y encontrar rápidamente qué significan ciertos términos, es la creación de glosarios. Hay muchas formas y muchos paquetes para hacer glosarios en  $\text{\LaTeX}$ . En este ejemplo hemos elegido el paquete `glossaries-extra` para la formación de glosarios. Las razones son que tiene opciones para hacer multiples glosarios y de todo tipo de manera fácil, se puede extender a cosas más complejas como glosarios conjuntos de libros de dos o más volúmenes (esto requiere `glossaries-extra` con `bib2gls`) y se puede hacer una versión más simple de glosarios (con `glossaries`).

El paquete `glossaries-extra` es uno de los que se debe cargar después de `hyperref`. Así que debería usarse en el orden el que aparece en el archivo `ejemplo.tex`. Además, puede causar conflictos con los comandos internos de *memoir* por lo que se deben cargar de la siguiente manera:

```
\let\printindex\relax%para evitar conflictos con memoir
\usepackage[symbols,index,abbreviations,automake]{glossaries-extra}
```

Las opciones son para tener índices de símbolos, terminos, de acrónimos (o abreviaturas) de manera simple y algo técnico en la construcción. En este ejemplo, además de los tres anteriores, haremos una “lista de nombres”. Primero debemos crear un nuevo glosario con el siguiente comando

```
\newglossary*{nom}{Lista de Nombres}
```

que consta de un tipo, `nom` en nuestro caso, y un título. Luego debemos dar una instrucción para crear los archivos necesarios para los glosarios

```
\makeglossaries
```

Finalmente, cargamos las entradas definidas en el archivo `gloss.tex` (donde podrás ver ejemplos de cómo crear entradas para un glosario) con el siguiente comando

```
\loadglsentries{gloss}
```

Ahora podemos usar las entradas del glosario con el comando `\gls{...}` o alguna de sus variantes. Una común es `\Gls{...}` donde pone la primer letra de la palabra que sustituye al comando en mayúscula. Otra común es `\glspl{...}` en donde la sutitución se pone en plural. En general el plural sólo pone una “s” al final, pero esto se puede cambiar con la opción `plural=...` de cada entrada.

Para completar este ejemplo escribimos algunos símbolos que pueden estar en modo texto *F*, *t*, *x* o en modo matemático *v*, *a*.

Los acrónimos cambian cuando se escribe la primera vez **support vector machine (SVM)** que cuando se escribe después **SVM**. Como es posible que uno quiera imprimir la versión completa más adelante en el texto existe el comando `\glstrfull{...}` que al usarlo con nuestro acrónimo aparece así: **support vector machine (SVM)**.

No hay mucho que decir en los nombres así que sólo lo usamos **Gauss, J. Carl F.** y ya.

Por último algunas entradas para el índice de términos **cardinal** y se puede revisar el archivo `gloss.tex` para ver cómo fue definido el plural para que lo podamos usar con `\glsp1{...}` y su salida sea **cardinales**.

El último comentario respecto a los glosarios es que requieren un paso extra para la compilación. Éstos requieren la siguiente cadena

```
lualatex ejemplo.tex
makeglossaries ejemplo
lualatex ejemplo.tex
```



## Compilación y arara

Como este documento puede ser compilado con Lua<sub>La</sub>T<sub>E</sub>X o pdf<sub>La</sub>T<sub>E</sub>X y tiene paquetes diferentes para cada una de las anteriores, entonces el pdf de salida depende de qué tipo de compilación se usó. Se recomienda que si compila con un método y luego el otro sea haga una compilación “nueva”, es decir, que antes de cambiar de método se eliminen los archivos auxiliares para evitar posibles conflictos.

En este momento la compilación de nuestro ejemplo se ha vuelto algo complicada y por tanto puede que tome algo de tiempo. Por ejemplo, para generar la tabla de contenidos es necesaria la siguiente cadena

```
lualatex ejemplo.tex
lualatex ejemplo.tex
```

En la sección 4.7 ya se había mencionado cuál es la cadena de compilación para obtener la bibliografía. También vimos en el capítulo 5 que los glosarios requieren un paso más de compilación.

Si juntamos todos los pasos necesarios para obtener un pdf a partir de nuestros archivos, entonces obtenemos algo así:

```
lualatex ejemplo.tex
biber ejemplo
makeglossaries ejemplo
lualatex ejemplo.tex
lualatex ejemplo.tex
```

Como claramente tiene más pasos puede que haga el proceso algo lento. Para hacer más rápida la compilación hay que evitar los pasos innecesarios. Esto es, una vez que se ha compilado la primera vez se han creado los archivos necesarios para componer bibliografía, glosarios, tabla de contenidos y otros. Si no hay modificaciones a la bibliografía o glosarios, no es necesario correr los pasos intermedios de arriba, ni tampoco correr tantas veces `lualatex`. Para hacer esta compilación condicional escribimos en las primeras líneas del archivo principal algo como lo siguiente

```
% arara: ...
```

se usan para compilar con `arara` ya que se pueden agregar condicionales de manera sencilla. Al menos mucho más simple que crear un `latexmk` o un `makefile`. Aunque seguramente su instalación local sí debe incluir `arara`, desgraciadamente `overleaf` no incluye esta opción e ignorará los *magic comments* del principio del archivo. (Overleaf usa un `latexmk` para compilar, este método se asegura de correr los pasos necesarios en cada compilación que hacemos.)

En el archivo `ejemplo.tex` se pueden ver los *magic comments* al inicio del documento, estos son

```
% !TEX root = ejemplo.tex

% arara: lualatex: { draft: yes } unless changed(toFile('ejemplo.aux'))
% arara: biber if changed (toFile('refs.bib')) || found('log', 'Citation')
% arara: --> || found ('log', 'Please \\(re\\)run Biber')
% arara: makeglossaries if missing('gls')
% arara: --> || changed('glo') || changed(toFile('gloss.tex'))
% arara: lualatex: { synctex: yes } until !found('log', '\\(?R|r)e\\)?run (to get|LaTeX)')
```

El primero le dice a mi editor de texto cual es el archivo principal. Después están las instrucciones de arara.

La primera instrucción con arara hace que compile con lualatex el archivo `ejemplo.tex` con el modo draft. Con esto sólo se crean los archivos auxiliares necesarios para generar el documento. Al no crear un documento completo en esta compilación se gana un poco de tiempo de compilación. Además sólo se hará esta compilación si no existe el archivo `ejemplo.aux`, ya que de existir es muy probable que ya haya habido una compilación previa y existan los archivos necesarios para una compilación completa.

La segunda instrucción es una compilación condicional de biber. En este caso sólo correrá biber si cambia el archivo `refs.bib` o si en el log aparece la palabra `Citation` o cuando en este mismo archivo se encuentran las palabras `rerun Biber`. Hay que notar que `arara: -->` se usa como una continuación del comando anterior. En nuestro caso biber usa dos líneas, lo mismo que `makeglossaries`.

La tercera instrucción es la compilación condicional de `makeglossaries`. Ahora sólo correrá `makeglossaries` cuando no exista un archivo `gls`, haya cambiado el archivo `glo` o haya cambiado el archivo con nuestras entradas de los glosarios, `gloss.tex`.

Finalmente, compila el archivo `ejemplo.tex` con la opción `synctex`, para que se comuniquen el editor y el lector de pdf, y correrá lualatex tantas veces como sea necesario, es decir, cuando en el log ya no diga que hace falta correr de nuevo lualatex para generar alguna cita bibliográfica, entrada del índice o alguna referencia.

Ahora nuestro archivo se compila con el comando

```
arara ejemplo
```

Para mostrar cómo mejoró el tiempo de compilación la segunda vez ponemos capturas de pantalla de la compilación de este documento, ver figuras. Además muchos editores de L<sup>A</sup>T<sub>E</sub>X pueden configurarse para compilar con arara. Por ejemplo, en Atom (el editor que uso) con el paquete `atom-latex` se configura como *custom toolchain* y el toolchain es:

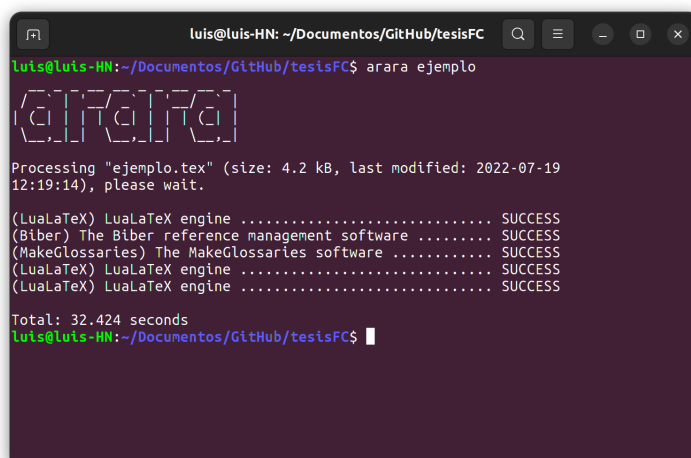
```
arara %DOC -v
```

Para configurarlo en TeXMaker hay que seguir las instrucciones en <https://tex.stackexchange.com/a/107995>.

En TeXWorks: <https://tex.stackexchange.com/a/98795>.

En TeXShop: <https://tex.stackexchange.com/a/175673>.

Un último comentario acerca del tiempo de compilación es que en el proceso de revisión y correcciones de la tesis, por ejemplo al revisar un capítulo específico donde no se requiera estar



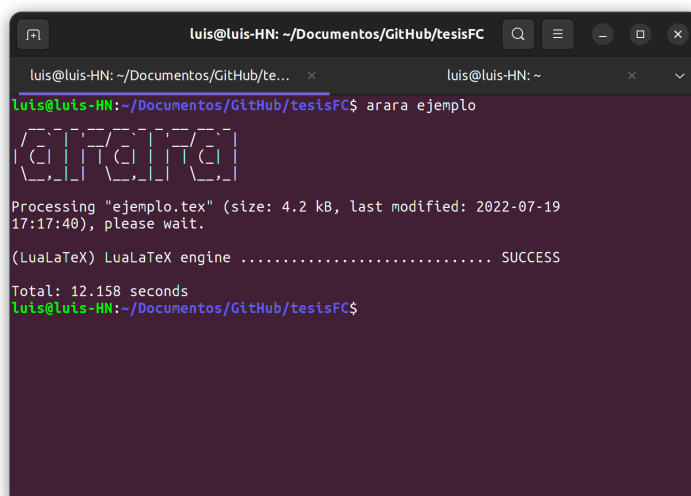
```
luis@luis-HN: ~/Documentos/GitHub/tesisFC
luis@luis-HN:~/Documentos/GitHub/tesisFC$ arara ejemplo

Processing "ejemplo.tex" (size: 4.2 kB, last modified: 2022-07-19
12:19:14), please wait.

(LuaLaTeX) LuaLaTeX engine ..... SUCCESS
(Biber) The Biber reference management software ..... SUCCESS
(MakeGlossaries) The MakeGlossaries software ..... SUCCESS
(LuaLaTeX) LuaLaTeX engine ..... SUCCESS
(LuaLaTeX) LuaLaTeX engine ..... SUCCESS

Total: 32.424 seconds
luis@luis-HN:~/Documentos/GitHub/tesisFC$
```

Figura 6.1: Primera compilación



```
luis@luis-HN: ~/Documentos/GitHub/tesisFC
luis@luis-HN:~/Documentos/GitHub/tesisFC$ arara ejemplo

Processing "ejemplo.tex" (size: 4.2 kB, last modified: 2022-07-19
17:17:40), please wait.

(LuaLaTeX) LuaLaTeX engine ..... SUCCESS

Total: 12.158 seconds
luis@luis-HN:~/Documentos/GitHub/tesisFC$
```

Figura 6.2: Segunda compilación

viendo los otros, se puede escribir en el preámbulo `\includeonly{CapituloEnRevision}`. Una vez que ya haya sido compilado el documento completo el comando anterior hará que sólo se compile el capítulo en revisión con la paginación y las referencias a otros capítulos correctas. Esto podría mejorar mucho el tiempo de compilación.



---

## Índice de términos

**cardinal** Un número que podría ser muuuy grande 18



---

## Lista de símbolos

$a$  aceleración 17

$F$  fuerza 17

$t$  tiempo 17

$v$  velocidad 17

$x$  posición 17





---

## Acrónimos

**SVM** support vector machine 18



---

## Lista de Nombres

Gauss, J. Carl F. 18



---

## Referencias

- [Joh13] P. T. Johnstone. *Topos Theory*. Dover Publications, 2013.
- [JT11] A. Joyal y M. Tierney. *Notes on simplicial homotopy theory*. <http://mat.uab.cat/kock/crm/hocat/advanced-course/Quadern47.pdf>. 2011.
- [Law07] F. W. Lawvere. “Axiomatic Cohesion”. En: *Theory and Applications of Categories* 19.3 (2007), págs. 41-49.
- [Mac98] S. Mac Lane. *Categories for the Working Mathematician*. Springer, 1998.



---

## Bibliografía

- [BJ01] F. Borceux y G. Janelidze. *Galois Theories*. Cambridge, UK: Cambridge University Press, 2001.
- [Bor02] F. Borceux. *Handbook of Categorical Algebra 1–3*. Cambridge, UK: Cambridge University Press, 2002.
- [CLW93] A. Carboni, S. Lack y R. Walters. “Introduction to extensive and distributive categories”. En: *Journal of Pure and Applied Algebra* 84 (1993), págs. 145-158.
- [GZ67] P. Gabriel y M. Zisman. *Calculus of Fractions and Homotopy Theory*. Belin: Springer-Verlag, 1967.
- [Isb76] J. R. Isbell. “Pulling paths and canonical sheaves of paths”. En: *Notices Amer. Math. Soc.* 1976, pág. 1.
- [Joh02] P. T. Johnstone. *Sketches of an Elephant: A Topos Theory Compendium*. Oxford University Press, 2002.
- [Joh11] P. T. Johnstone. “Remarks on Punctual Local Connectedness”. En: *Theory and Applications of Categories* 25.3 (2011), págs. 51-63.
- [Joh79] P. T. Johnstone. “On a Topological Topos”. En: *Proc. London Math. Soc.* s3-38.2 (1979), págs. 237-271.
- [Joh92] P. T. Johnstone. *Stone Spaces*. Cambridge, UK: Cambridge University Press, 1992.
- [Law05] F. W. Lawvere. “Categories of Spaces may not be Generalized Spaces as Exemplified by Directed Graphs”. En: *Reprints in Theory and Applications of Categories* 9 (2005), págs. 1-7.
- [Law75] F. W. Lawvere. “Variable setes, topoi and étendu”. En: *Notices American Mathematical Society* 22.A675 (1975).
- [Law82] F. W. Lawvere. “Introduction”. En: *Categories in Continuum Physics*. Lectures Notes in Mathematics. Berlin Heidelberg New York Tokyo: Springer-Verlag, 1982, págs. 1-16.
- [Law97] F. W. Lawvere. *Toposes of Laws of Motion*. Transcript from video. 1997.
- [LM15] F. W. Lawvere y M. Menni. “Internal Choice Holds in the Discrete Part of any Cohesive Topos Stisfying Stable Connected Codiscretness”. En: *Theory and Applications of Categories* 30.26 (2015), págs. 909-932.
- [LR03] F. W. Lawvere y R. Rosebrugh. *Sets for Mathematics*. Cambridge University Press, 2003.
- [Men14] M. Menni. “Continuous Cohesion over sets”. En: *Theory and Applications of Categories* 29.20 (2014), págs. 542-568.

- [MM17] F. Marmolejo y M. Menni. “On the relation between continuous and combinatorial”. En: *Journal of Homotopy and Related Structures* 12 (2017), págs. 379-412.
- [MM92] S. Mac Lane e I. Moerdijk. *Sheaves in Geometry and Logic*. Springer, 1992.
- [Nol74] W. Noll. *The Foundations of Mechanics and Thermodynamics*. Berlin Heidelberg New York: Springer-Verlag, 1974.
- [Tru91] C. A. Truesdell. *A First Course in Rational Continuum Mechanics*. Vol. 1–3. Academic Press, Inc., 1991.