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

TUTOR

GRADO Y NOMBRE

2021

FACULTAD DE CIENCIAS
UNAM

Índice general

Índice de figuras

Índice de Diagramas

CAPÍTULO 1

Un poco de memoir

1.1. Lo básico

La clase `tesisFC` tiene como base a *memoir*. La motivación es poder configurar de manera fácil los ascpectos 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`. Memoir tiene definidos muchos más estilos de salida por defecto, estos pueden verse en http://www.ebookation.com/wp-content/uploads/2010/03/memoirchapstyles.pdf

Modifique el tamaño y la posición de la caja de texto para obtener una linea de texto

suficientemente grande como evitar tantos cortes de palabra, pero no ta grande como para que sea difícil pasar de una lína 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 que el margen de la orilla (el opuesto a la espina). Lo mismo sucede con los margenes superior e inferior, donde el superior es más chico que el inferior. Tomamos en cuenta estos detalles en la formación de este documento.

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

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

abstract	chngpage	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.

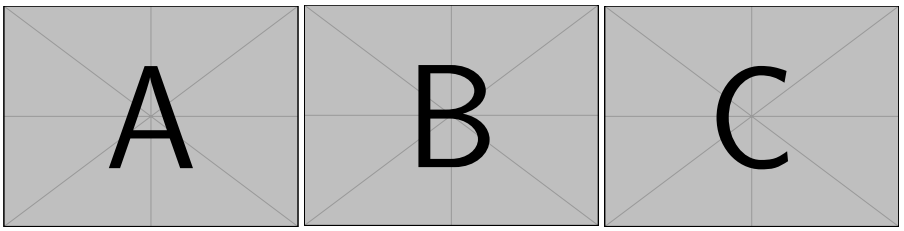
Memoir puede crear índices de manera relativamente sencilla, en este ejemplo haremos uno de términos o alfabético y uno de nombres. Para esto se escribió en el preámbulo

```
\makeindex
\makeindex[names]
```

luego una entrada en el índice de términos se crea con el comando `\index{término}` término. Este índice puede crear subtérminos de la siguiente manera subtérmino y uno más subsubtérmino (ver código fuente).

Ahora una entrada para el índice de nombres Grothendieck. Nota que en este índice se específico que pertenece al índice de nombres con el parámetro opcional `[names]`.

Al final del archivo principal se puede ver cómo se imprimieron.



(a) Primera (b) Segunda (c) Tercera

Figura 1.1: Muchas figuras

1.2. 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 2.1 hay una forma de hacerlas usando el comando `\subcaption{...}` para poner los subtítulos correspondientes. También es posible hacer como en el siguiente ejemplo 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 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.

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 nuevo 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.

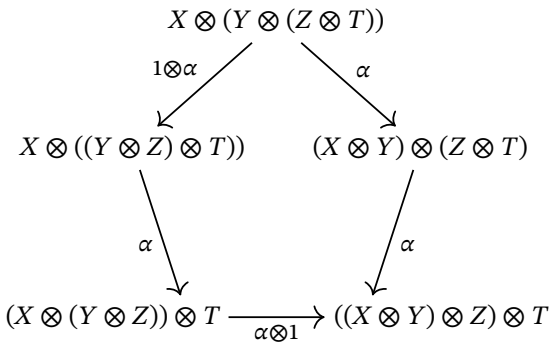


Diagrama 1: ¡Pentagonator!

1.2: Un circu-
l 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.

1.3. Me falta desarrollar mejor

Memoir puede crear glosarios de una forma muy similar a los índices. Se puede crear, por ejemplo, una lista de acrónimos y una de símbolos para ejemplificar cómo se hacen. En el preámbulo se escribe

```
\makeglossary[acro]
\makeglossary[simb]
```

Luego un acrónimo `\glossary[acro]{HTML}{ HyperText Markup Language}` y un símbolo `\glossary[simb]{\(c\)}{Una constante importante}`. Por último, en el lugar del documento donde vayan a ir los glosarios se escribe

```
\clearpage %para crear una página nueva \printglossary[acro] \clearpage
\printglossary[simb]
```

Para cambiar el nombre del glosario, de la misma manera que en los índices, se usa

```
\renewcommand{\glossaryname}{Un glosario}
```

La dificultad que encuentro en la creación de glosarios es la compilación. Para esto también se usará *makeindex*, pero si se intenta correr este programa en alguno de los archivos para crear glosarios, por ejemplo

```
makeindex acro.glo
```

se generaran muchos errores. Entonces, para lograr hacer estos glosarios se necesita un archivo de configuración `basic.gst` que encontrarás en los archivos de este proyecto y con esto la cadena de compilación que se usa para los glosarios es

```
...
makeindex -s basic.gst -o acro.gls acro.glo
makeindex -s basic.gst -o simb.gls simb.glo
lualatex MiDocumento.tex
```

como no estoy seguro como llamar un archivo de configuración externo en `arara` tuve que correr los dos `makeindex` en la terminal.

Acabo de hacer una prueba en overleaf y tristemente no soporta esta construcción de glosarios. Por lo tanto sugeriré el paquete `glossaries` o su versión extendida `glossaries-extra`. La versión extendida tiene muchas capacidades con el método `bib2gls` pero tampoco es soportado por overleaf. Por lo que para hacer glosarios en overleaf hay que usar la versión simple de `glossaries` que debería ser suficiente para una tesis.

CAPÍTULO 2

Fuentes

Para las fuentes del documento he escogido los paquetes `fontspec` y `unicode-math`. Con estos será posible cambiar fuentes de manera fácil en partes del documento pero deberá ser compilado con XeLaTeX o con LuaLaTeX, el último siendo el método recomendado. Con esto se podrán usar caracteres unicode en cualquier parte de texto, por ejemplo en una ecuación centrada

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

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).

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 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 1 Lorem ipsum dolor sit amet, consectetuer 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 $\forall \varepsilon > 0 \exists \delta > 0(\ldots)$. 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.

(a) Suma de vectores (b) Resta de vectores

Figura 2.1: Operaciones con vectores

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 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 y tablas, también se podría cambiar el tamaño de la fuente o poner alguna palabra en específico.

Otra ventaja del manejo de fuentes de XeLaTeX y LuaLaTeX es que las fuentes instaladas en nuestra 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 pdfLaTeX 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 XeLaTeX y LuaLaTeX basta ver las fuentes disponibles en overleaf https://www.overleaf.

com/latex/examples/fontspec-all-the-fonts/hjrpnxhrrtxc. Seguramente en tu sistema habrá un conjunto de fuentes disponibles grande. Junto con los archivos de este documento encontrarás un script de `lua` para ver cuáles son las fuentes disponibles en tu sistema (aún no sé cómo hacer que el nombre de la fuente se imprima en la fuente correspondiente).

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 otra 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 topos 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) `\symbf{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.

CAPÍTULO 3

Paquetes en la clase

3.1. amsmath y mathtools

Aunque en el título de la sección se menciona `amsmath` en el archivo principal 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. 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*. Ya están definidos los operadores más comunes, pero si se quisiera definir uno nuevo se escribe en el preámbulo `\DeclareMathOperator{\id}{Idem}` para definir los idempotentes de un anillo, por ejemplo `\id(R)` genera $\mathrm{Idem}(R)$.

Además de esto están los ambientes matemáticos como `align`, `gather`, `cases`, etc.

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 que símbolo "cierra". La deferencia de este comando se puede apreciar con el código `|\sum_{i=1}^{n}a_1|`, `\lvert\sum_{i=1}^{n}a_1\rvert`, `\abs{\sum_{i=1}^{n}a_1}` y `\abs*{\sum_{i=1}^{n}a_1}`, que su salida es, respectivamente,

$$|\sum_{i=1}^{n} a_1| \quad |\sum_{i=1}^{n} a_1| \quad |\sum_{i=1}^{n} a_1| \quad \left|\sum_{i=1}^{n} a_1\right|$$

También podemos hacer conjuntos con delimitadores y la línea de "tal que" que

crezcan correctamente (ver su definición en `ejemplo.tex`)

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

3.2. amsthm

Este paquete provee mejoras útiles para las definiciones de teoremas (LaTeX puede crea teoremas sin necesidad de paquetes) y define un ambiente de demostración (esto no lo hace LaTeX). Con este paquete se pueden usar y definir estilos de teoremas de forma fácil. En la calse se definieron los siguinetes ambientes:

```
definicion   teorema      corolario
 lema        proposicion  observacion
```

Con la salida esperada del nombre del ambiente.

Definición 3.1. Una función $f : X \to Y$ es continua si para cualquier abierto $V \subseteq Y$ se tiene que $f^{-1}(V) \subseteq X$ es abierto.

Teorema 3.2 (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. □

Se modificó el estilo del ambiente demostración para que su salida sea similara a la de los resultados —Una demostración es tan importante como el enunciado—. Nuestra redefinición del ambiente de demostración tiene lo necesario para comportarse bien con el símbolo □. Es decir, cuando se termina una demostración con una ecuación u otro tipo de ambiente habrá un espacio vertical indeseado 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 3.3. *Un resultado importante.*

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

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

□

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

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

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 3.2" 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 $f: D \to R$ 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 demostración como un nuevo ambiente. El ejemplo 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`.

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.

3.3. babel

Para el soporte de idiomas elegimos `babel` 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 $\pi = 3.141592\ldots$

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 máxA o lím 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 XeLaTeX o LuaLaTeX es `polyglossia`. Este paquete se creó cuando `babel` había dejado de tener mantenimiento con el objetivo de simplificar su trabajo en estos motores. Si se elige usar `polyglossia` este paquete también tiene una variante `mexico`, 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.

De esta forma podríamos usar `polyglossia` con la siguiente configuración

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

Tanto `babel` como `polyglossia` son buenas opciones para el manejo de idiomas y es difícil elegir uno sobre el otro. En este caso elegí `babel` por ser más conocido (lleva mucho más tiempo existiendo) y por ser el más estable (`polyglossia` está en desarrollo, aunque lo he usado en la mayoría de mis documentos y no he tenido problemas con la configuración como la de arriba).

3.4. microtype

Este paquete habilita los aspectos micro-tipográficos del documento. Algunos de ellos ya son hechos por TeX como justificación y la separación silábica (cortes de palabra). En general las opciones que se le pasaron a este paquete sirven para calcular la expansión de letras, palabras y permitir que algunos caracteres, como el guión de un corte de palabra, 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`. El resultado es un pdf que luce tipográficamente mejor.

3.5. siunitx

Este paquete tiene como objetivo implementar la escritura de cantidades físicas de acuerdo con las reglas del sistema internacional de medidas[1]. 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 separase 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
`\ang{1;2;3}`	$1°2'3''$
`\si{\gram\per\cubic\centi\metre}`	$\mathrm{g\,cm^{-3}}$
`\si{kg.m/s^2}`	$\mathrm{kg\,m/s^2}$
`\SI{.23e7}{\candela}`	$0.23 \times 10^7\,\mathrm{cd}$
`\SI[mode=text]{1.23}{J.mol^{-1}.K^{-1}}`	$1.23\,\mathrm{J\,mol^{-1}\,K^{-1}}$
`\SI[per-mode=symbol]{1.99}[\$]{\per\kilogram}`	$1.99/kg
`\SI[per-mode=fraction]{1,345}{\coulomb\per\mole}`	$1.345\,\frac{\mathrm{C}}{\mathrm{mol}}$

3.6. biblatex

En principio no se eligió 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`.

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 el siguiente link 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.

[1] https://www.bipm.org/en/measurement-units/

`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 haré 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 hice 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 capitulo, 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{*}`.

Por último es recomendable usar el paquete `csquotes` junto con `biblatex` para tener algunas facilidades en citas y ajustar las comillas al idioma que se quiera. En este documento se uso junto con la opción `autostyle` que carga el estilo de comillas del idioma que se haya pasado a babel, es este caso español. Como puede verse en la bibliografía en México no usamos las comillas de México, más bien usamos las comillas inglesas. Para hacer el cambio a comillas inglesas se debe sustituir `autostyle` por `style=american`

Ahora algunos ejemplos de bibliografía. Primero todas las entradas donde Lawvere es autor, en el archivo `.bib` use el campo `keywords` para lograr esto se usó el comando

```
\printbibliography[keyword=Lawvere,heading=subbibliography,title=Lawvere]
```

Usamos `heading=subbibliography` para que el título de la bibliografía aparezca un nivel más bajo que "el nivel principal" en este caso el nivel principal es capítulo, así que el título será impreso como una sección sin número.

Para imprimir una bibliografía con todos los libros que están en `refs.bib` hacemos lo siguiente

```
\printbibliography[type=book,heading=subbibliography,title=Libros]
```

Como puse el ejemplo con muchos tipos de bibliografía y al final pondré toda la bibliografía usaré el estilo bibliográfico `alphabetic`. Otros estilos son `authoryear-icomp`, `numeric` (este es común en matemáticas) `author-title`, etc. Además de estos y las variantes ya definidas para apa, chicago, mla, etc. Es mucho más fácil definir o modificar un estilo usando `biblatex` (ya que se hace con comandos de LaTeX comunes, al estilo `\renewcommand`) que usando `natbib` o `cite` donde se requiere editar/crear un archivo `.bst` con un lenguaje, en principio, muy diferente a LaTeX.

En el ejemplo de bibliografías múltiples con el estilo `numeric` sería ideal usar la opción `resetnumbers` para que cada bibliografía empiece en [1], pero esto creará inconsistencias en la numeración de la bibliografía que pondré al final del documento.

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
```

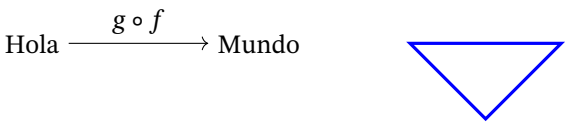
CAPÍTULO 4

Tikz

Tikz es un paquete muy amplio y es bastante difícil describir todas sus capacidades. Por ejemplo el manual, que hace precisamente eso, es de mas de 1000 páginas. Por esta razón sólo veremos lo más básico de este paquete

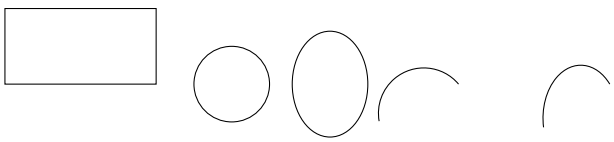
4.1. Nodos y líneas

Tikz crea un plano cartesiano en la página. Así para poner contenido se especifica una coordenada o bien una recta entre dos puntos del plano, como en el siguiente ejemplo.



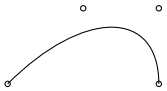
La sintaxis `\node (x) at (0,0) {Hola};` crea un nodo con el contenido "Hola" en la posición $(0, 0)$ y le pone una etiqueta "(x)" para futuras referencias. En la flecha de (x) a (y) se puso un nodo sobre la flecha con el comando `\draw[->] (x) --node[above] {\(g\circ f\)} (y);`. Las opciones de posición son `above`, `below`, `left` y `right`. Para mover de posición el nodo se puede usar `near start`, `near end`, `center` (la opción por defecto) o una más granular `pos=x`, donde `x` es un número entre 0 y 1.

Algunas figuras ya están definidas en la base de tikz. Algunas de ellas son:



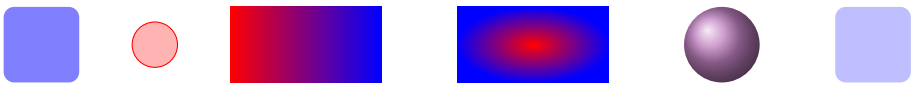
Hay muchas más en la librería `shapes` o `shapes.geometric`.

También se pueden hacer curvas "jalando" una línea en dos puntos

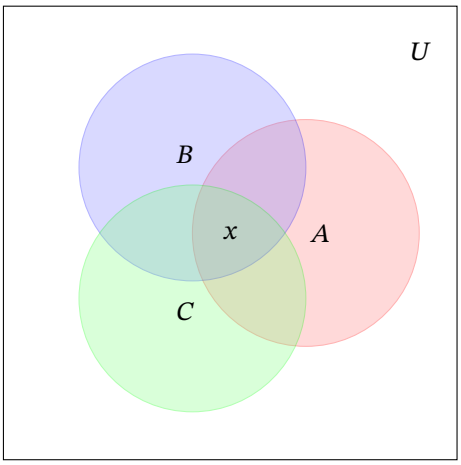


donde los puntos de arriba sólo fueron puestos para hacer referencia de qué puntos se "jalo" la recta.

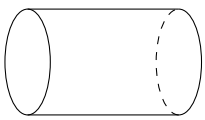
Colorear figuras también es fácil, además de que se pueden hacer muchos estilos de coloreado.



Una aplicación de la opción `opacity` es hacer diagramas de Venn. Para esto voy a usar coordenadas polares. Estas usan la sintaxis (ángulo:longitud).

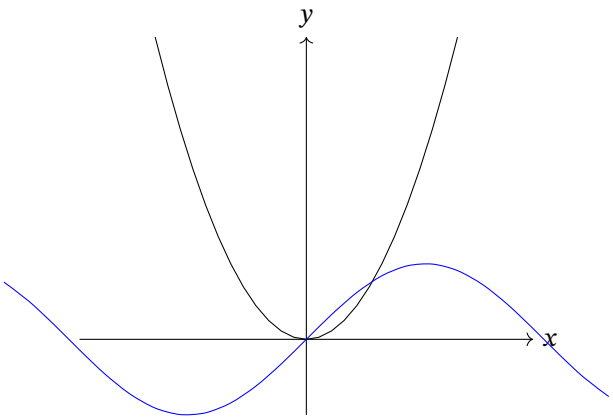


Una aplicación de `arc` puede ser dibujar un cilindro



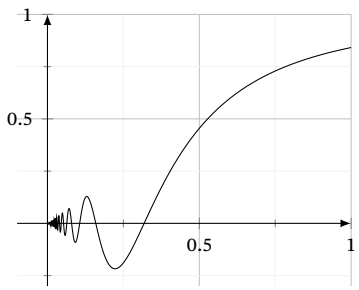
A pesar de que esto es lo más básico de Tikz se pueden tener muchas posibilidades combinando estas reglas básicas entre ellas.

Otra función de Tikz es graficar funciones. En el siguiente ejemplo graficamos una parábola y el seno. En las funciones trigonométricas se debe especificar como se hacen los cálculos: radianes, grados, etc. Para obtener el seno como queremos se escribe `sin((\y)r)`.



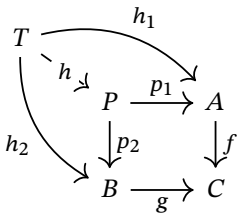
Esta forma simple es bastante efectiva y su única posible desventaja es que se debe tener cuidado con las dimensiones de la grafica para que no haya algunas muy grandes y otras muy chicas, generando una inconsistencia indeseada.

Tikz es la capa superior de `pgf` así los métodos de Tikz serán muy similares a los de `pgf`. Por esta razón otro método para gráficar es usar el paquete `pgfplots` que usará el motor de `pgf` para hacer gráficas. Es importante notar que se escribió `\pgfplotsset{width=6cm,compat=1.17}` en el preámbulo. La parte de `width` es para que todas las gráficas tengan el mismo tamaño y así tener uniformidad en el texto. La parte de `compat` es la versión del motor que se usará, diferentes versiones podrían generar diferentes salidas con el mismo código. Para usar la versión más actual siempre se escribe `compat=newest`, aunque esta opción no es recomendada ya que es como trabajar con una versión beta de pgf y como se mencionó antes la salida del mismo código podría cambiar.



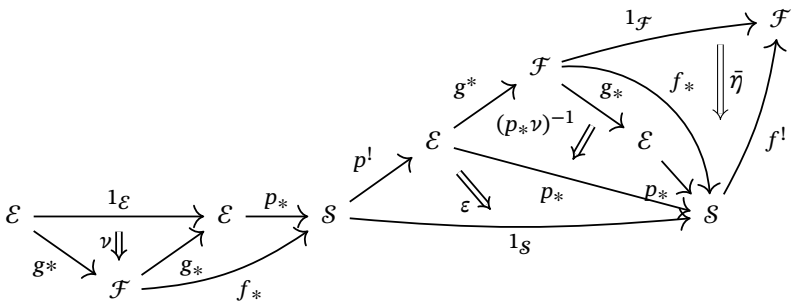
4.2. Diagramas conmutativos

Una aplicación que seguramente tendrá mayor uso en la carrera de matemáticas es dibujar diagramas conmutativos. En la sección 4.1 ya hicimos una flecha entre dos objetos con un nombre sobre dicha flecha. Con la misma técnica se pueden hacer diagramas conmutativos. Aquí veremos como se hacen usando `\usetikzlibrary{cd}` o equivalentemente `\usepackage{tikz-cd}`.



Hay otros paquetes para hacer diagramas conmutativos, uno que aún es muy común es `xy-pic`. También tiene una versión de su sintaxis similar a la de `tikzpicture` de arriba y una versión de matriz como la de `tikzcd`. A pesar de ser común en estos días creo que el paquete que debería usarse es `tikz` ya que `xy-pic` tiene cosas que están mal medidas y tiene muchos años que no tiene mantenimiento. Tiene tantos años sin mantenimiento que se quedo sin soportar la compilación con LuaLaTeX, por lo que no es posible mostrar aquí un ejemplo, pero basta hacer un monomorfismo o una inclusión, sobretodo vertical, para ver que quedan demasiado cerca del dominio.

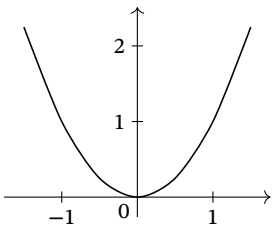
Esta forma de hacer diagramas por medio de matrices podría parecer limitada respecto a la de coordenadas con `tikzpicture`, pero los métodos de tikz son suficientemente robustos como para hacer el siguiente diagrama con matrices (este usa la librería `calc` para mover fácilmente la flecha $\bar{\eta}$):



4.3. Visualización de datos

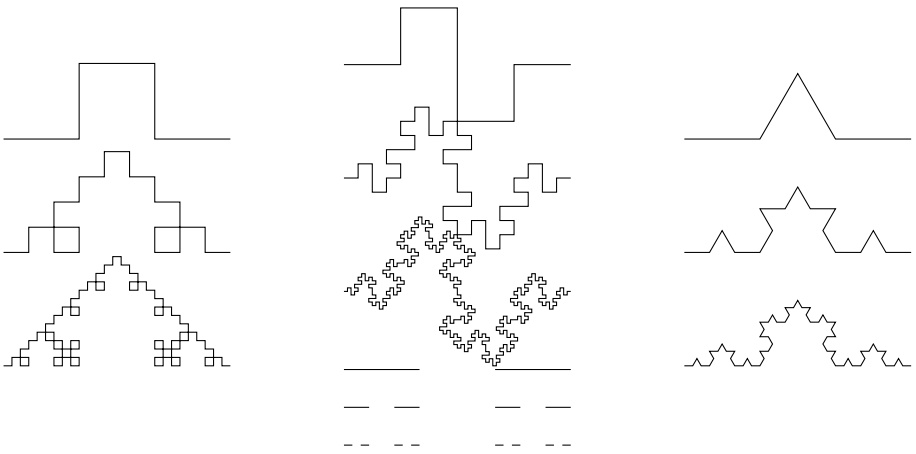
No he tenido la necesidad de trabajar visualizando información de una base de datos, así que no tengo un ejemplo útil de esto. Sin embargo, en el manual de tikz/pgf el capítulo 80 se encarga de esto. Afortunadamente este es un manual muy bien escrito y con muchos ejemplos, así que una mirada rápida dará los conceptos básicos para hacer conjuntos de datos dentro del documento o usar conjuntos de datos de un archivo `.csv`, `.dat`,...

Aún así va un intento de ejemplos. Para esto voy a usar las librerías `datavisualization` y `datavisualization.formats.functions`. Un ejemplo del manual:

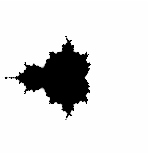


4.4. Por diversión, Fractales

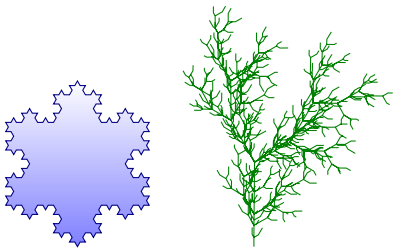
Con `\usetikzlibrary{decorations.fractals}`



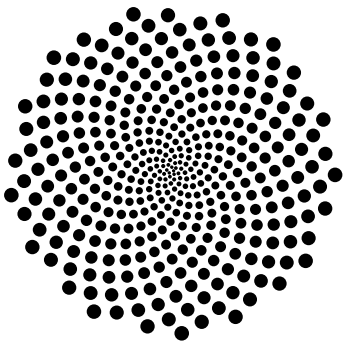
Con `\usetikzlibrary{shadings}` tenemos un Mandelbrot, si hacen esto en overleaf hay que cambiar el visor de pdf para que sí aparezca el dibujo. Esto se hace desde el menú en la parte superior izquierda.



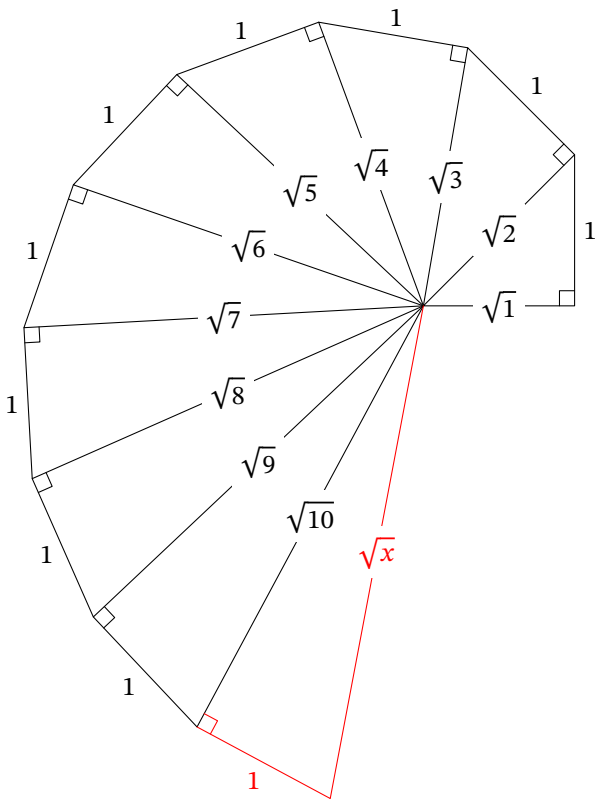
Con `\usetikzlibrary{lindenmayersystems}`



Girasol de https://www.texample.net



Uno más usando la librería `math`


1
1
1
1
1
1
1
1
1
1
$\sqrt{1}$
$\sqrt{2}$
$\sqrt{3}$
$\sqrt{4}$
$\sqrt{5}$
$\sqrt{6}$
$\sqrt{7}$
$\sqrt{8}$
$\sqrt{9}$
$\sqrt{10}$
$\sqrt{x}$
1

CAPÍTULO 5

Compilación y arara

En este momento la compilación 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 MiDocumento.tex
lualatex MiDocumento.tex
```

En la sección 3.3 ya se había mencionado cuál es la cadena de compilación para obtener la bibliografía. Finalmente, para generar los índices la compilación debe ser:

```
lualatex MiDocumento.tex
makeindex MiDocumento.idx
lualatex MiDocumento.tex
```

Para ser más precisos se debería correr `makeindex` en cada índice que se haya creado. En nuestro documento debería ser en el documento principal, como arriba, para formar al índice alfabético y en `names.idx` para formar el índice de nombres.

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

```
lualatex MiDocumento.tex
biber MiDocumento
makeindex MiDocumento.idx
lualatex MiDocumento.tex
lualatex MiDocumento.tex
```

donde, de nuevo, `makeindex` se debe correr en cada índice que se haya creado. 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, índices, tabla de contenidos y otros. Si no hay modificaciones a la bibliografía o índices, 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 siguiene

```
% 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.

Con estos comandos sólo compilará la bibliografía cuando sea necesario, la primera vez y cuando se modifiqué la bibliografía. De la misma forma sólo compilara el índice alfabético cuando haga falta. En la primera compilación de `lualatex` sólo creara los archivos auxiliares, ganando así un poco de tiempo y las siguientes veces que corra `lualatex` activará la opción `synctex` para que su editor y visor se comuniquen.

Ahora se compila con el comando

```
arara MiDocumento
```

Para mostrar cómo mejoró el tiempo de compilación la segunda vez pongo capturas de pantalla de la compilación de este documento. Además muchos editores de LaTeX 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.

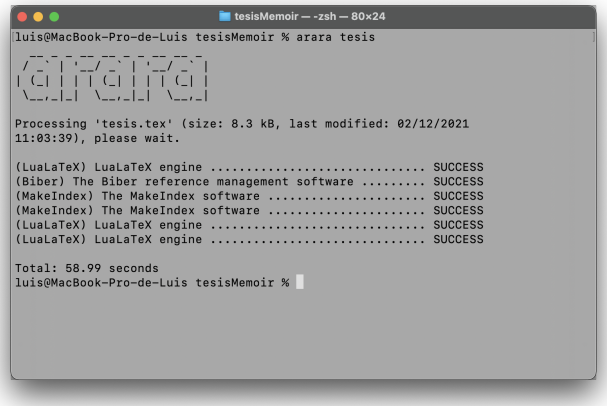


Figura 5.1: Primera compilación

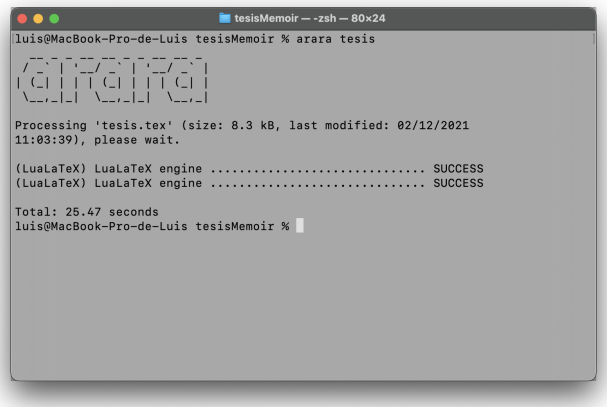


Figura 5.2: Segunda compilación