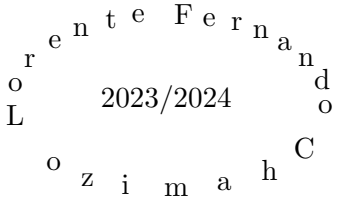
# LaTeX en la UAM

## en diez lecciones

Fernando Chamizo Lorente

Curso 2023/24



# Prefacio

Estas notas corresponden al curso 2023/24 de una asignatura transversal bajo el sobrecargado nombre "*Composición de textos científicos con editores de libre distribución (LaTeX)*" en la que tradicionalmente se matriculan estudiantes del Grado de Matemáticas que desean aprender a manejar LaTeX para escribir su Trabajo de Fin de Grado. El plan es cubrir aquí todos los aspectos que necesitaría un estudiante en tal situación.

Veremos también en la última sección el uso de la herramienta BibTeX que, aunque estrictamente no es LaTeX, es realmente muy útil para la organización de la bibliografía y últimamente se ha vuelto bastante común.

A pesar de que no hay una fuente unificada de todo este documento, que ha sido creado concatenando ficheros PDF, las fuentes `.tex` de cada lección por separado están accesibles en `https://matematicas.uam.es/~fernando.chamizo/asignaturas/2324latex/2324latex.html`.

Madrid 8 de enero de 2024

Fernando Chamizo

# Tabla de contenidos

# Comenzamos

**Lección 1**

## 1. ¿Qué es LaTeX?

En los años 80 el famoso informático D. Knuth creó el sistema tipográfico TeX y LaTeX surgió de la mano de L. Lamport para facilitar su uso. En pocas palabras, sobre el papel LaTeX es TeX con nuevos comandos y entornos predefinidos y en principio uno podría reproducir el aspecto visual de cualquier documento LaTeX empleando TeX. En la práctica, muy pocos usuarios hoy en día sabrían desenvolverse con soltura utilizando solo TeX porque es demasiado primitivo, hay demasiadas cosas básicas que requerirían conocimientos profundos y una cantidad nada desdeñable de código. Ciertamente el hijo ha superado al padre.

Al principio, LaTeX era un poco rígido, si no te gustaba la forma en que se había diseñado el formato de los títulos de secciones o cosas similares la única forma era aguantarse o aprender mucho del funcionamiento interno, lo que era casi como volver a TeX. Una comunidad creciente de desarrolladores, una documentación muy extensa y la posibilidad de cargar paquetes para las cosas más peregrinas, han hecho que esto sea cosa del pasado.

Una pregunta natural es para qué quiero un LaTeX si ya tengo en mi ordenador algo como Word con dos ventajas fundamentales:

1. Está universalmente extendido. Casi todo ciudadano del siglo XXI con un ordenador podrá abrir un fichero `.docx` o similar.

2. El uso básico es prácticamente trivial. Cualquiera puede abrir un documento y ponerse a escribir directamente. Es la excusa para incluir en los currículos el eufemismo "conocimientos a nivel de usuario".

Incluso si uno odiara este producto, en muchos ámbitos uno "debe" saber algo de Word. Hasta la administración pública u organismos dependientes de ella fuerzan a veces a presentar documentación en Word sin alternativas, lo cual es llamativo siendo un *software* de pago.

En comparación, posiblemente pocos de los que acceden a nuestras redes sociales podrán compilar un fichero `.tex` y la prueba de que no es trivial es que exista este curso en una universidad como la UAM que aparece en los *rankings* de centros de prestigio. Lo primero queda rebajado con que siempre

podremos crear un fichero PDF, que sí es universal, de hecho no difundir el fichero fuente nos da cierta manera de demostrar nuestra autoría.

La gran ventaja de LaTeX es obvia para cualquiera que haya pasado por el infierno de utilizar el editor de fórmulas de Word. Una vez que uno tiene práctica con LaTeX todo es muchísimo más rápido y queda muchísimo más atractivo. La gran mayoría de las revistas de matemáticas y física, y seguramente de otras áreas científicas, ya solo admiten artículos en LaTeX.

En el corazón de TeX y por tanto de LaTeX hay un complejo algoritmo para la división de palabras en líneas y párrafos. Gracias a ello, incluso en un texto sin fórmulas el resultado es más profesional, aunque quizá sea difícil que convenzas a algunos de ello.

Las diferencias más obvias de LaTeX y Word, y lo que hace que la curva de aprendizaje del primero tenga una pendiente inicial alta, son que en LaTeX uno utiliza comandos y no ve el resultado hasta que compila. Esto requiere para empezar cierta memorización de comandos y alguna habilidad para interpretar los posibles errores. Por ejemplo en LaTeX usaremos el comando `\textbf{Texto}` para poner la palabra `Texto` en negrita y si se nos olvida la segunda llave es posible que nos diga que se niega a compilar con un aviso del tipo `File ended while scanning use of \textbf` que en este caso es medianamente orientativo, pero otras veces no tanto.

## 2. Instalar o no instalar

Ya sabemos que LaTeX requiere disponer TeX. Es muy conveniente tener también un editor especialmente adaptado que complete comandos o nos señale errores obvios. La docencia de esta asignatura tiene lugar en un aula que dispone de ordenadores bajo el sistema operativo Linux (Debian) y con el editor, o entorno de desarrollo, TeXstudio para LaTeX.

Sin embargo más de una vez querremos trabajar en casa o en cualquier otro lugar con nuestro portátil. Si estamos escribiendo nuestro TFG, sería un poco ridículo depender del horario del aula. Hay dos posibilidades: instalar en nuestro ordenador una distribución de TeX y un buen editor adaptado o usar LaTeX online o, como parece que se dice más ahora, *en la nube*.

La primera posibilidad tiene los inconvenientes de que lleva un rato y ocupa una cantidad de memoria que no es despreciable para los viejos que tenemos la mentalidad informática rácana (formada con el ZX Spectrum y el Amstrad CPC 6128). Por otro lado, si uno va a usar LaTeX a menudo, por ejemplo para escribir su TFG, es lo personalmente me parece más adecuado y así uno no depende de la conexión de red o de los posibles fallos en la plataforma donde tengan los ficheros. De nuevo quizá solo sean paranoias ligadas a un inicio en internet con Mosaic.

Si prefieres no instalar nada o esperar a probar un poco hasta que te decidas, la opción que actualmente tiene mayor difusión es Overleaf. En

`https://www.overleaf.com/` uno puede registrarse gratuitamente y utilizar LaTeX de forma muy similar a como si lo tuvieras instalado. Tengo una cuenta de Overleaf que apenas uso y prácticamente no me han dado nunca lata con mensajes, más allá de que me notificaron al principio alguna novedad.

En mi opinión es más conveniente instalar porque los editores son mejores y uno puede configurar lo que quiera, pero todo el curso se puede seguir con Overleaf sin instalar nada. Mi única duda es si algún paquete que uno quiera usar en una tarea voluntaria puede faltar en Overleaf, yo creo que no, pero no estoy totalmente seguro. En cualquier caso solo afectaría a trabajos voluntarios.

## 2.1. Instalar

Las dos distribuciones más conocidas de TeX son `texlive` y `MiKTeX`. La segunda, hasta donde sé, no está disponible para Linux. Hay tantas versiones de los sistema operativos y mi experiencia es tan sesgada que seguramente te pueda brindar poca ayuda. Lo mejor que puedes hacer es buscar unas instrucciones de instalación recientes en la red. Por ejemplo siguiendo los enlaces `https://www.latex-project.org/get/`. En el blog de J.R. Berrendero hay un enlace a una guía, un poco antigua, para instalar MiKTeX con TeXmaker.

En breve, mi experiencia es que en Linux solo he tenido que marcar algo en la lista del gestor de paquetes y en Windows ejecutar un instalador. Lleva tiempo, pero en casi todos los casos ha resultado trivial. En Windows el editor se debe instalar después de que se ha instalado la distribución TeX. Los cuatro editores más famosos son:

1. Kile (Linux)
2. TeXmaker (posiblemente el más usado)
3. TeXstudio (un proyecto que procede de TeXmaker)
4. TeXnicCenter (nunca lo he usado)

Mi preferido es Kile. En teoría es posible usarlo también en Windows, pero si quieres hacerlo hay que instalar algunas cosas para que simulen cierto entorno de escritorio de Linux y posiblemente no merezca la pena. Estos editores se instalan como cualquier otro programa. Al menos Kile no es muy pesado, ocupa poco en memoria.

No he incluido en la lista anterior LyX que parece que cuenta con sus adeptos porque tengo el prejuicio de que los editores WYSIWYM solo te retrasan en cuanto tienes un poco de práctica. Nunca he conocido a nadie que escriba rápido en LaTeX y use editores de ese tipo.

### 2.2. No instalar

Si quieres utilizar LaTeX online a través de Overleaf, debes registrarte en `https://www.overleaf.com/`. Una vez que lo hayas hecho cuando entres ya podrás usar LaTeX.

A veces he visto vender la ventaja de Overleaf frente a instalar LaTeX porque permite trabajar con diferentes versiones y coautores de forma automática. En este curso no vamos a utilizar esta funcionalidad porque el trabajo de cada estudiante debe ser personal. Por otro lado, en lo poco que la he usado, no he visto grandes diferencias a intercambiar ficheros por *email*. Supongo que con muchos coautores la cosa cambiaría. Automáticamente guarda las versiones y el historial de los accesos de cada autor.

Aunque no te registres en Overleaf podrás disfrutar de muchos ejemplos y documentación acerca de LaTeX que ponen a disposición de cualquiera. El acceso a algunas plantillas está restringido a usuarios registrados.

## 3. Mi primer fichero LaTeX

Como es lógico cada editor tiene su propia estructura y atajos, pero lo básico es bastante intuitivo. Configurar todos los detalles requiere bucear un poco en la documentación. Vamos a ver tres situaciones representativas.

### 3.1. Con Kile

Dentro de `File`, o con el atajo, pinchamos en `New` donde nos aparece una lista de posibles plantillas. Elegimos por ejemplo `Article`. Podemos rellenar datos de esta plantilla o borrar casi todo hasta reducirla a

```
\documentclass[a4paper,10pt]{article}
\usepackage[utf8]{inputenc}

\begin{document}

\end{document}
```

Lo que escribamos entre `\begin{document}` y `\end{document}` será el texto. Para compilarlo y generar un fichero DVI el atajo de teclado es `Alt`+2 y para verlo `Alt`+3. Si lo que queremos es un PDF, los análogos son `Alt`+6 y `Alt`+7.

### 3.2. Con Overleaf

Al entrar verás un botón con “Create first project” o una lista vacía de proyectos y un botón con “New Project”. Pinchando en él tienes las opciones *Blank Project*, *Example Project* y *Upload Project* cuyo nombre

es suficientemente explicativo. Eligiendo la primera opción nos pedirá un nombre y tendremos un editor donde aparece la fuente LaTeX a la izquierda y el resultado en PDF a la derecha. Después de hacer los cambios que queramos podemos compilar con el botón de "Recompile". El fichero PDF resultante se puede descargar a través del icono correspondiente. Por cierto, el que está al lado es para ver el fichero `.log` describiendo todo el transcurso de la compilación. El `stdout file` es la versión breve que se ve en consola en otros editores.

### 3.3. Con TeXmaker y TeXstudio

Si abrimos un fichero nuevo estará en blanco. Lo mejor es usar el asistente donde podemos escoger algunos parámetros para hacer una plantilla. No te preocupes si ahora no te imaginas el efecto de casi ninguno de ellos, aceptando la opción por defecto tendrás una plantilla razonable.

Esencialmente la compilación, al menos en las versiones de TeXmaker que yo conocía, va como con Kile, pero hay que sustituir `Alt+n` por `Fn`. Por ejemplo, para compilar en PDF se usa la tecla especial de función F6. Por alguna razón en la versión de los ordenadores del aula la tecla para compilar es F5. En cualquier caso, si uno duda, siempre hay un menú con pestañas y seguramente algunos iconos que harán difícil que te pierdas.

Al abrir el PDF con acrobat reader, al menos con mis versiones, no deja compilar de nuevo porque dice que el fichero está ocupado. Si no tenemos una vista por defecto que nos muestre el PDF, yo lo he solucionado utilizando el visor TexWork, incluido en MiKTeX para abrir el PDF.

### ¿Dónde están mis ficheros?

Para resolver las tareas del curso tendrás que enviar ficheros `.tex` y ficheros `.pdf`. Una pregunta evidente es dónde se han almacenado.

Con los editores instalados, posiblemente hayas empleado un fichero anterior de uno de tus directorios y todos los ficheros estarán allí. Si lo has creado de nuevo, puede que estén en un directorio general por defecto dentro de tu sistema. Lo más fácil es que utilices la pestaña `File > Save as` de tu editor y lo guardes de nuevo donde quieras, compilándolo otra vez para que se genere el PDF. En raras ocasiones, sobre todo según avance el curso, te parecerá que TeXmaker y TeXstudio son un poco sordos y hay que compilar dos veces. Volveremos sobre ello, simplemente estate precavido.

Con Overleaf, más arriba se ha dicho cómo descargar el PDF. Para la fuente LaTeX, el `.tex`, las cosas son menos intuitivas. En la página que ves nada más entrar, estará la lista de tus proyectos. Pincha en la nubecita con una flecha, a la derecha del proyecto deseado bajo la etiqueta "Actions", y te descargarás un `.zip`. Descomprímelo para obtener el `.tex` y cámbiale el nombre si fuera necesario, porque si no has hecho nada se llamará `main.tex`.

## 4. Dos posibles problemas

En nuestro primer fichero escrito en LaTeX hay dos cosas que inicialmente nos pueden dar dolores de cabeza. Lo mismo con tu editor o con el texto que has escrito, no te percatas de ello, pero es importante que no te sorprendan cuando cambies de ordenador o recibas un fichero de otra persona.

La primera no es algo de LaTeX sino un problema informático general, que los viejos usuarios de Linux conocen bien. Diferentes sistemas utilizan diferentes codificaciones, esto es, diferentes maneras de asignar bytes a los caracteres. Por ejemplo que `O` (la letra o mayúscula) se represente por `4F` en hexadecimal es bastante universal, pero cómo indicar `Ó` (el mismo símbolo acentuado) es algo que habitualmente difiere en Linux o Mac y Windows, lo cual nos puede machacar todos los acentos de un texto. Además dentro de cada uno de estos sistemas operativos uno puede tener editores que trabajen con una u otra codificación. Dentro de un fichero LaTeX, la codificación se puede especificar con el paquete `inputenc` (*input encoding*) que apareció en la cabecera generada con Kile y debe coincidir con la del texto y la del editor que manejemos. Dos posibilidades son:

`\usepackage[utf8]{inputenc}` y `\usepackage[latin1]{inputenc}`

que corresponden a las dos codificaciones principales usadas en gran parte del mundo o al menos en la que nos resulta más cercana.

La segunda cosa que nos puede dar disgustos es que incluso con la codificación adecuada, los acentos podrían todavía dar problemas o los comandos no responden en el lenguaje que nos gustaría. Por ejemplo, el comando `\today` escribe la fecha de hoy y lo hará en inglés aunque nuestro texto esté en castellano. Evidentemente, LaTeX no puede adivinar en qué idioma queremos escribir y de hecho en el origen de los tiempos solo se podía hacer en inglés y los acentos se lograban con comandos. Afortunadamente, existe el paquete `babel` que admite muchísimos idiomas, entre los que se encuentra, por supuesto, el español. Ya veremos más sobre la internacionalización. Ahora basta con saber que seguramente nos guste incluir muy arriba en nuestra cabecera, antes de la codificación, `\usepackage[spanish]{babel}`. Por ejemplo, un posible código fuente mínimo para escribir en español a partir de lo que nos da como plantilla Overleaf quitando todas los cosas que no usamos es:

```
\documentclass{article}
\usepackage[spanish]{babel}
\usepackage[utf8]{inputenc}

\begin{document}
\end{document}
```

Recuerda que el texto va entre `\begin{document}` y `\end{document}`. Si quieres prueba el efecto de los comandos `\today` y `\LaTeX`. Seguramente no funcionarán como tú esperabas cuando están dentro de una frase, por ejemplo cuando escribes:

```
\begin{document}
Aprenderé \LaTeX muy
rápido.
\end{document}
```

Si no quieres esperar a otra clase para saber la solución, enciérralos entre llaves o escribe `{}` tras ellos. También te sorprenderá que no te haga mucho caso en los espaciamientos entre palabras, ni respete los fines de línea (como en el ejemplo anterior) o las líneas en blanco. Esto que parece una desventaja, es en realidad una de las grandes fortalezas del corazón de LaTeX, un maravilloso algoritmo que pone los espacios adecuados para que el resultado sea armonioso.

Una cosa más, sobre todo para los que hacéis el TFG en matemáticas. Si lo visto te sabe a poco y quieres empezar a compilar ficheros con una cabecera bastante complicada, por ejemplo la plantilla que te ofrecen para el TFG, lo mismo ves errores muy raros. Es posible que provengan de paquetes de los que no dispones. El remedio expeditivo es instalarlos. En el caso de la plantilla del TFG para matemáticos hay algún paquete un poco inusual y, en general, innecesario (a no ser que hagas cosas avanzadas). Si ese fuera el caso, simplemente precede el `\usepackage` correspondiente por un %, lo que hará que el compilador se olvide de esa línea.

## 5. Un vistazo más a la cabecera

La cabecera de un documento es lo que hay antes de `\begin{document}`. En nuestros primeros ejemplos, podemos usar algo tan breve como

```
\documentclass{article}
\usepackage[utf8]{inputenc}
```

De hecho esto es lo que pone Overleaf como dos primeras líneas al crear un fichero nuevo. La primera indica que se toman los parámetros por defecto de un formato llamado `article`. Con otros editores se especifican algunos de estos parámetros entre corchetes. Por ejemplo

```
\documentclass[a4paper,10pt]{article}
```

que indica que el formato del papel es A4 y que las fuentes son de tamaño diez puntos. Podemos jugar a cambiar `10pt` por `11pt` o `12pt`, o `a4paper` por `letterpaper` (los folios usados en EE.UU.) y ver los resultados.

La segunda línea indica que la codificación es UTF8 con un parámetro que se pasa al paquete `inputenc`. Parámetros aparte, un paquete es algo que se carga con `\usepackage{...}` y que extiende de cierta forma las capacidades de LaTeX. Hay algunos muy específicos y otros que hacen cosas muy tontas. Por ejemplo, incluyendo `\usepackage{lipsum}` en la cabecera dispondremos de la instrucción `\lipsum` que genera el típico texto *Lorem ipsum* usado en pruebas de imprenta, con las variantes `\lipsum[n-m]` que dan los párrafos entre `n` y `m`. Por ejemplo `\lipsum[11-11]`, que se puede abreviar con `\lipsum[11]`, produce:

---

Sed feugiat. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Ut pellentesque augue sed urna. Vestibulum diam eros, fringilla et, consectetuer eu, nonummy id, sapien. Nullam at lectus. In sagittis ultrices mauris. Curabitur malesuada erat sit amet massa. Fusce blandit. Aliquam erat volutpat. Aliquam euismod. Aenean vel lectus. Nunc imperdiet justo nec dolor.

---

En matemáticas casi todo el mundo carga tres paquetes de la AMS (*American Mathematical Society*) incluso si no los usa y así lo haremos nosotros. Nuestras cabeceras incluirán

```
\usepackage{amsmath}
\usepackage{amsfonts}
\usepackage{amssymb}
```

En la práctica, casi ningún matemático tenemos mucha idea de qué comandos dejarían de funcionar si no los cargásemos y la realidad es que en textos simples no necesitaríamos ninguno. Es más una costumbre y una precaución. Un físico o un químico seguro que hará lo mismo con otros paquetes.

# Principios del modo matemático

**Lección 2**

## 1 Dos tipos de fórmulas

Las fórmulas en un texto matemático pueden aparecer dentro de una línea *inline* o resaltadas y centradas *displayed*. Las primeras en LaTeX se encierran entre `$` y las segundas entre `\[` y `\]`. Por ejemplo la fórmula $x + y = 2$ se ha escrito como `$x+y=2$` mientras que `\[ x+y=2 \]` (que se suele escribir en tres líneas en el código fuente para que la fórmula sea más visible) daría lugar a

$$x + y = 2$$

Los espacios en las fórmulas matemáticas son en general irrelevantes, así que `\[ x +y = 2 \]` daría el mismo resultado. Cabe preguntarse en el primer caso por qué no tecleamos la fórmula como texto, sin `$`, el resultado sería entonces x+y=2 que es más feo y posiblemente incoherente con el tipo de letra de otras fórmulas que involucren $x$ e $y$. Esto se aplica también a fórmulas mínimas que solo contienen un carácter. Seguro que te resultaría raro leer en un libro algo de la forma:

Despejamos la x de

$$2x - 4 = 0.$$

Lo correcto habría sido escribir "`Despejamos la $x$ de`" para que se preserve el tipo de letra.

Si examinas fuentes escritas por otras personas, a veces verás `$$...$$` en lugar de `\[...\]` porque en TeX, que es la base de LaTeX, es así. A pesar de que funciona y es más fácil de teclear, es preferible evitarlo porque en ciertas situaciones muy particulares no gestiona bien centrados y espacios (para complicar más las cosas, en principio `$...$` también es TeX y su reflejo en LaTeX debería ser `\(...\)`, pero prácticamente nadie lo usa y, hasta donde yo sé, en este caso no hay ninguna diferencia).

Una de las peculiaridades de las expresiones matemáticas es que a menudo contienen elementos que no respetan la altura natural de la línea. Los casos más simples son los subíndices y superíndices que en LaTeX se indican respectivamente con `_` y `^`. Cuando estos tengan más de un carácter o comando, los agruparemos con llaves. Por ejemplo, $E = mc^2$ es `$E=mc^2$` y

$E = mc^{1+1}$ es `$E=mc^{1+1}$`. Esto lleva a la duda de cómo teclear una llave si queremos que aparezca de verdad, basta poner \ delante. Así tenemos

$\{x_n\}$ ↔ `\{x_n\}` y $\{x_n\}_{n=1}^{\infty}$ ↔ `\{x_n\}_{n=1}^\infty`

Además hemos aprendido que el comando `\infty` da lugar a $\infty$.

La regla general es que en LaTeX los símbolos matemáticos se reemplazan por sus nombres en inglés o por abreviaturas suyas precedidos de la barra hacia atrás \ y, en general, esta misma filosofía se aplica a los comandos LaTeX aunque no correspondan a símbolos matemáticos (como vimos con `\LaTeX` y `\today`). Es también un hecho general que los espacios tras un comando no se tienen en cuenta, por ello se usa `{\LaTeX}` o `\LaTeX{}` si queremos aislar el resultado del comando `\LaTeX` de la siguiente palabra.

Ejemplos bastante comunes de comandos matemáticos son los que dan lugar a las letras griegas, para las que se usan el nombre completo. Si se desean en mayúsculas y no existen en el alfabeto latino, se pone la primera letra en mayúsculas. Así

```
\[
 \alpha^\beta,
 \qquad
 \Omega^{\lambda^2},
 \qquad
 \theta(\pi)
\]
```

da lugar a

$$\alpha^{\beta}, \qquad \Omega^{\lambda^2}, \qquad \theta(\pi)$$

y además hemos aprendido que `\qquad` introduce cierta separación entre fórmulas. En la línea de lo explicado, no existe `\Alpha` porque la letra $\alpha$ mayúscula es similar a la del alfabeto latino. Las letras griegas $\epsilon$, $\theta$, $\rho$ y $\phi$ admiten variantes tipográficas usadas históricamente. Se indican precediendo el nombre con `var`. Por ejemplo, con `\varepsilon` se obtiene $\varepsilon$, que es la tipografía preferida en libros de cálculo.

Las funciones trigonométricas básicas son

`\sin`, `\cos`, `\tan`, etc.

Si utilizamos el paquete de idioma `babel` con la opción `spanish` tendremos la traducción `\sen`. Más adelante en el curso veremos cómo definir `\sen` u otros comandos nosotros mismos. Otras funciones son

`\log`, `\exp`, `\min`, `\max`, `\inf`, `\sup`

y seguro que puedes imaginar más ejemplos y comprobarlos en un manual.

De nuevo surge la pregunta de para qué necesitamos escribirlo de esta forma si podríamos teclearlo como texto. Seguramente lo siguiente responda a la pregunta:

| | | | |
|---|---|---|---|
| `$cos 0 + tan(0+0)$` | $\rightarrow$ | $cos0 + tan(0 + 0)$ | Mal |
| `cos 0 + tan(0+0)` | $\rightarrow$ | $\cos\ 0 + \tan(0+0)$ | Mal |
| `$\cos 0 + \tan(0+0)$` | $\rightarrow$ | $\cos 0 + \tan(0 + 0)$ | Bien |

De acuerdo, en la del centro quizá habría que decir que está regular. Su problema es que, siendo un poco exigente, el espaciamiento no es el adecuado.

## 2 Fracciones y raíces

Hay algunos comandos LaTeX que necesitan argumentos. Los más comunes son las fracciones. Su formato es `\frac{}{}` donde entre las primeras llaves se pone el numerador denominador y entre las segundas el denominador. Omitir las llaves solo funciona con números de un dígito. La mayor parte de los editores completan el comando para que sea más fácil de usar. LaTeX toma decisiones "inteligentes" acerca del tamaño de la fracción. Por ejemplo, con `E=\frac{3}{4} mc^2` obtenemos $E = \frac{3}{4}mc^2$ en una fórmula *in-line*, pero como *displayed* sería

$$E = \frac{3}{4}mc^2$$

(por cierto, aunque suene a broma, esta fórmula apareció en la física del siglo XX para la energía del electrón antes de la llegada de la relatividad).

También las fracciones de fracciones siguen normas sensatas acerca del tamaño:

$$\frac{x+\frac{1}{2}}{y} \leftrightarrow \texttt{\textbackslash frac\{x+\textbackslash frac 12\}\{y\}}$$

Más adelante veremos como cambiar esto a nuestro gusto.

Las raíces cuadradas responden a `\sqrt{...}` y se generalizan a raíces de índice `n` mediante `\sqrt[n]{...}`. Por ejemplo

$$\sqrt{\frac{1+x}{1-x}} \leftrightarrow \texttt{\textbackslash sqrt\{\textbackslash frac\{1+x\}\{1-x\}\}}, \quad \sqrt[3]{1+2^4} \leftrightarrow \texttt{\textbackslash sqrt[3]\{1+2\^{}4\}}$$

Yo prefiero leer en un libro $(1+x)^{1/5}$ que $\sqrt[5]{1+x}$ o $(1+x)^{\frac{1}{5}}$. La estructura de `\sqrt[n]{...}` no es arbitraria. En LaTeX un posible parámetro opcional se escribe entre corchetes. Buscando la analogía con algunos temas de programación (Python, C++) que probablemente te suenen, el comando `\sqrt` funciona como si asignara por defecto contenido vacío a lo que está encima del "paraguas" mientras que un parámetro opcional sobreescribe el valor por defecto. Tardaremos un tiempo en ver dentro del curso alguna otra cosa en LaTeX que admita un parámetro opcional.

# 3 Operadores con límites

Algunas de las funciones que hemos visto y algunos operadores (comandos) admiten límites. Como regla general los superiores se indican como si fueran superíndices y los inferiores como subíndices. Estos límites se colocan automáticamente dependiendo de si la fórmula es *in-line* o *displayed*. El primer ejemplo básico son los límites, en sentido matemático, que como cabía esperar se indican con `\lim` y, también como cabía esperar, suelen incorporar `\to` para indicar el "tiende a". Por ejemplo

$$\lim_{x\to 0^+} x^{\tan x} = 1$$

se obtiene con

```
\[
 \lim_{x\to 0^+} x^{\tan x}=1
\]
```

y en una línea veríamos $\lim_{x\to 0^+} x^{\tan x} = 1$.

Algo similar se puede decir de máximo, mínimo, supremo e ínfimo. Por ejemplo, ¿sabrías dar una fuente para las siguientes fórmulas?

$$\inf_{x>0} \frac{\cos x}{1+x^2} \qquad \text{y} \qquad \max_{n>1}\{\exp(10n - n^2)\}.$$

Los operadores llamados "grandes" admiten también límites superiores. Los más utilizados son

`\sum`, `\prod`, `\bigcup` y `\bigcap`

que sin sus límites dan lugar a

$$\sum, \quad \prod, \quad \bigcup \quad \text{y} \quad \bigcap.$$

Por ejemplo, la famosa identidad del problema de Basilea y una de las leyes de Morgan,

$$\sum_{n=1}^{\infty} \frac{1}{n^2} = \frac{\pi^2}{6} \qquad \text{y} \qquad \bigcup_{j\in J} A_j^c = \Big(\bigcap_{j\in J} A_j\Big)^c,$$

se obtendrían con el código

```
\sum_{n=1}^\infty
\frac{1}{n^2}
=
\frac{\pi^2}{6}
```

y

```
\bigcup_{j\in J}A_j^c
=
\big(
\bigcap_{j\in J}A_j
\big)^c
```

Más adelante veremos los diferentes delimitadores, pero el último ejemplo ya muestra que `\big(` y `\big)` producen paréntesis un poco más grandes de lo normal.

Las integrales siguen la filosofía anterior, así podríamos escribir la consecuencia del teorema fundamental del cálculo

$$\frac{d}{dx}\int_0^x f = f(x)$$

con

```
\frac{d}{dx}
\int_0^x f = f(x)
```

Hay un punto con las integrales sobre el que volveremos más adelante y es que cómo espaciar el diferencial. Por ejemplo, si uno es un poco exigente la fórmula

$$\int_0^1 xdx = \frac{1}{2}$$

que se obtendría con `\int_0^1 x dx= \frac 12` queda un poco fea porque $x$ y $dx$ están pegados. Casi todo el mundo se contenta con $x\,dx$ y algunos quieren ver $x\,\mathrm{d}x$, esto es, con la "d" recta. Si no deseas mirar ahora el código fuente de esta guía, espera a que avance un poco más el curso.

# 4 Operadores binarios y relaciones binarias

Aparte de cosas como `=`, `>` y alguna otra más que podemos teclear directamente con nuestro teclado los operadores que suelen relacionar dos términos de una expresión matemática son

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $\leq$ | `\le` | $\geq$ | `\ge` | $\cup$ | `\cup` | $\cap$ | `\cap` |
| $\in$ | `\in` | $\equiv$ | `\equiv` | $\subset$ | `\subset` | $\supset$ | `\supset` |
| $\times$ | `\times` | $\cdot$ | `\cdot` | $\vee$ | `\vee` | $\wedge$ | `\wedge` |
| $\sim$ | `\sim` | $\simeq$ | `\simeq` | $\approx$ | `\approx` | $\circ$ | `\circ` |
| $\pm$ | `\pm` | $\setminus$ | `\setminus` | $\mid$ | `\mid` | $\perp$ | `\perp` |

Muchos de ellos se pueden negar al precederlos de `\not`. Así $\not\approx$ se obtienen con `\not\approx`. Están las abreviaturas particulares `\notin`, `\ne` y `\nmid` para $\notin$, $\neq$ y $\nmid$. La última es casi obligatoria porque `\not\mid` no da el resultado esperado. Con mucho, `\nmid` es el comando que más uso de los definidos en el paquete `amssymb`, lo que da una idea de que el resto no son muy comunes.

# 5 Otros símbolos

Ya conocemos símbolos como $\infty$ o las letras griegas. Los editores de LaTeX suelen ofrecer en algún apartado una lista de los símbolos más comunes u otros más extraños. Incluso los usuarios más experimentados de vez en cuando los dan vistazo. Dependiendo de los temas sobre los que solamos escribir, unos símbolos se volverán mas o menos comunes. Algunos bastante universales son

$\partial$ `\partial` $\nabla$ `\nabla` $\emptyset$ `\emptyset` $\forall$ `\forall` $\exists$ `\exists`

La negación del último tiene una abreviatura propia `\nexists` que da $\nexists$, porque `\not\exists\verb` no queda bien alineado. Otros preferirán emplear `\lnot\exists` ya que `\lnot` es la negación lógica $\neg$. Aunque el comando por defecto para el conjunto vacío es `\emptyset`, la verdad es que queda más bonito $\varnothing$ obtenido con `\varnothing`.

Personalmente, los siguientes cuatro símbolos que son variantes de letras los uso con cierta frecuencia, sobre todo los dos primeros (para indicar partes reales e imaginarias):

$\Re$ `\Re` $\Im$ `\Im` $\wp$ `\wp` $\ell$ `\ell` $\hbar$ `\hbar`

El último es una necesidad si a uno le gusta la física teórica.

Otros cinco símbolos que apenas uso, pero que a ti te pueden resultar interesantes, son:

$\aleph$ `\aleph` $\angle$ `\angle` $\square$ `\square` $\triangle$ `\triangle` $\heartsuit$ `\heartsuit`

Mi recomendación es que no pierdas el tiempo memorizando muchos símbolos. Al final aprenderás los que uses habitualmente y el resto los olvidarás.

Aunque la aplicación web `detexify` es poco más que un juguete, nos puede sacar de algún apuro si tenemos pulso y no tenemos ganas de buscar en la documentación. Basta dibujar en `https://detexify.kirelabs.org` el símbolo deseado e intentará darnos el código.

# Más sobre el modo matemático

**Lección 3**

## 1 Delimitadores

Con este palabro, me refiero a los símbolos que usamos para indicar bloques en una fórmula. Estos son los paréntesis y cosas parecidas que se abren y cierran. Los cuatro grupos más empleados son $(\dots)$, $\{\dots\}$, $|\dots|$ y $\|\dots\|$. Los paréntesis y las barras simples verticales se teclean directamente. Ya sabemos que las llaves se indican con `\{` y `\}`. Para las dobles barras se emplea `\|`. Algo menos comunes son: $[\dots]$ y $\langle\dots\rangle$, y más especializados aún $\lceil\dots\rceil$ y $\lfloor\dots\rfloor$.

Los que se obtienen directamente con el carácter de una tecla, quizá precedidos de `\` están resumidos en esta tabla:

| Código | `(` `)` | `\{` | `\}` | `\|` | `\\|` | `[` `]` |
|---|---|---|---|---|---|---|
| Resultado | ( ) | { | } | \| | ‖ | [ ] |

Mientras que los que tienen nombre son:

| Código | `\langle` | `\rangle` | `\lceil` | `\rceil` | `\lfloor` | `\rfloor` |
|---|---|---|---|---|---|---|
| Resultado | $\langle$ | $\rangle$ | $\lceil$ | $\rceil$ | $\lfloor$ | $\rfloor$ |

Como cabe suponer, la `l` y la `r` indican *left* y *right*.

Lo que distingue a los delimitadores es la posibilidad de escalarlos en función de las fórmulas que encierren. Hay dos maneras de proceder, una manual y otra automática. La primera consiste en preceder los delimitadores con uno de los siguientes operadores:

`\big` `\Big` `\bigg` `\Bigg`

que están ordenados de forma creciente en el tamaño que inducen sobre el delimitador. Por ejemplo

```
\[
 \bigg\{
 \Big(
 2\cdot
 \big\lceil x_1+x_2\big\rceil
 \Big)
```

```
 \int f
 \bigg\}.
\]
```

da lugar a:

$$\left\{\left(2\cdot\lceil x_1+x_2\rceil\right)\int f\right\}.$$

En esta forma manual no hay necesidad de que los delimitadores estén compensados porque el tamaño lo estamos eligiendo nosotros y LaTeX no necesita saber dónde comienza o acaba la fórmula para obedecernos.

Un error de principiante es confundir el mayor y el menor, $>$ y $<$, con los paréntesis angulares. Los primeros no son delimitadores, en particular no se pueden escalar, y además están regidos por reglas para los espacios que suponen que hay dos miembros que se están comparando. Incluso en ejemplos mínimos, verás la diferencia. ¿Cuál de las siguiente fórmulas preferirías para el producto escalar en $L^2$?

$$\langle f,g\rangle=\int \bar{f}g, \qquad <f,g>=\int \bar{f}g.$$

La primera es la ortodoxa y está hecha con los delimitadores `\langle` y `\rangle`. Por cierto, $\bar{f}$ se obtiene con `\bar f`.

La forma automática deja a LaTeX que decida el tamaño de los delimitadores en función de la fórmula que encierren y por ello es importante que estén compensados. Para ello se precede el inicial por `\left` y el final por `\right` y LaTeX hará el ajuste. Por ejemplo,

```
\[
 \left(x+y\right)\cdot
 \left(
 \frac{x+\frac{z}{t+\frac 12}}{y+\sqrt{\frac{a}{b}+1}}
 \right).
\]
```

da lugar a

$$(x+y)\cdot\left(\frac{x+\frac{z}{t+\frac{1}{2}}}{y+\sqrt{\frac{a}{b}+1}}\right).$$

El primer `\left` y el primer `\right` son innecesarios, el resultado sería el mismo escribiendo simplemente `(x+y)` porque esos paréntesis no necesitan ser escalados ya que encierran algo de la altura habitual de una línea.

El efecto autómatico sobre el tamaño de los delimitadores se hace patente en el siguiente ejemplo:

$$\left(\left(\left(\int_0^\infty e^{-x^2}dx\right)^2\right)^2\right)^2=\frac{\pi^4}{256}$$

que responde a

```
\[
 \left(
  \left(
  \left(
   \int_0^\infty
   e^{-x^2} dx
  \right)^2
  \right)^2
 \right)^2
 =
 \frac{\pi^4}{256}
\]
```

En algunas situaciones se necesita escribir un delimitador sin compensar que de todas formas guarde el tamaño de una porción de fórmula. La manera de conseguirlo es abrirlo o cerrarlo con `\left.` o `\right.` (el punto es importante) que actúan como si allí estuviera el delimitador, pero sin mostrarlo. Por ejemplo, con

```
\[
 \left.
  \frac{\partial F}{\partial x}
 \right|_{x=0}
\]
```

se consigue

$$\left.\frac{\partial F}{\partial x}\right|_{x=0}$$

No es obligatorio que un `\left` y su `\right` correspondiente se apliquen al mismo delimitador. Así la manera más sencilla de obtener

$$\left|\frac{1}{2}\right\rangle$$

es

```
\[
 \left|
  \frac 12
 \right\rangle
\]
```

## 2 Vectores y matrices

Los vectores en álgebra lineal se representan habitualmente poniendo una flechita sobre la letra que los nombra. En LaTeX se utiliza `\vec{...}`. Por ejemplo, la relación entre el llamado triple producto vectorial y el producto escalar

$$\vec{a} \times (\vec{b} \times \vec{c}) = (\vec{a} \cdot \vec{c})\vec{b} - (\vec{a} \cdot \vec{b})\vec{b}$$

se escribiría como

```
\vec{a}\times (\vec{b}\times \vec{c})
=
(\vec{a}\cdot \vec{c})\vec{b}
-
(\vec{a}\cdot \vec{b})\vec{b}
```

Hay algún pequeño defecto con los vectores relativo a espaciamientos (que quizá ya alguien haya advertido en los paréntesis anteriores). Por ejemplo cuando queremos poner un apóstrofo tras un vector llamado $p$ deberíamos escribir `$\vec{p}'$`, pero esto da lugar a $\vec{p}'$ que es muy feo, sin embargo $\vec{a}'$ no da problemas. La mejor manera de evitar esto es definir un nuevo comando, pero eso por ahora está fuera de nuestros conocimientos. Aparte de este defecto, hay que tener en cuenta que la flecha no se ajusta al nombre del vector por ello debemos teclear `$\vec{p}_0$` y evitar `$\vec{p_0}$` pues los resultados son respectivamente $\vec{p}_0$ y $\vec{p_0}$. De igual forma, no es aconsejable (ni habitual) que el nombre de un vector tenga más de un carácter. Incluso hay letras simples que vectorizadas no quedan muy bonitas. Por ejemplo, $\vec{m}$ acusa la diferencia de anchura entre la $m$ y su flecha. Si uno quiere que la flecha se ajuste, debe usar `$\overrightarrow$`. Por ejemplo el vector de $A$ a $B$ se indicaría con `$\overrightarrow{AB}$` que produce $\overrightarrow{AB}$. Hay otras soluciones basadas en paquetes especiales que corrigen y amplían el primitivo tratamiento que da LaTeX a los vectores.

Para las matrices se suele utilizar el entorno `matrix`. Dicho sea de paso, un *entorno* es algo que empieza con un `begin` y acaba con un `end`. La separación entre los elementos de una fila de una matriz se indica con `&` y la separación entre filas con `\\` así que en un primer intento de escribir la matriz identidad $2 \times 2$ usaríamos

```
\[
 \begin{matrix}
  1 & 0
  \\
  0&1
 \end{matrix}
\]
```

donde los espacios se han puesto un poco al azar para mostrar que son irrelevantes. El resultado no es el teníamos en mente:

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

Los paréntesis que nos faltan se pueden añadir como delimitadores, concretamente

```
\[
 \left( \begin{matrix} 1 & 0
  \\   0&1 \end{matrix} \right)
\]
```

(de nuevo se ha jugado con espacios y cambios de líneas arbitrarios) daría lugar al resultado deseado:

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

y, obviamente, si nos gusta más que las matrices aparezcan con corchetes bastaría cambiar `(` y `)` por `[` y `]`. La barra nos da una forma de escribir determinantes:

```
\[ \left|
\begin{matrix} 2&1\\ 5&3\end{matrix}
\right|=1 \]
```

produce $\begin{vmatrix} 2 & 1 \\ 5 & 3 \end{vmatrix} = 1$

En realidad, el entorno `matrix` no es LaTeX puro sino que se encuentra definido en el paquete `amsmath` y no funciona sin él. Como las matrices se escriben prácticamente siempre con sus delimitadores, dentro de este mismo paquete hay definiciones de entornos que ya los incluyen. Los nombres de estos entornos y el tipo de delimitador que introducen son:

| `pmatrix` | `bmatrix` | `Bmatrix` | `vmatrix` | `Vmatrix` |
|---|---|---|---|---|
| paréntesis | corchetes | llaves | barras | dobles barras |

Visualmente su aspecto es, respectivamente:

$$\begin{pmatrix} 1 & 1 & 2 \\ 3 & 5 & 8 \end{pmatrix}, \quad \begin{bmatrix} 1 & 1 & 2 \\ 3 & 5 & 8 \end{bmatrix}, \quad \begin{Bmatrix} 1 & 1 & 2 \\ 3 & 5 & 8 \end{Bmatrix}, \quad \begin{vmatrix} 1 & 1 & 2 \\ 3 & 5 & 8 \end{vmatrix}, \quad \begin{Vmatrix} 1 & 1 & 2 \\ 3 & 5 & 8 \end{Vmatrix}.$$

Por ejemplo, para la tercera matriz se ha usado

```
\begin{Bmatrix}
 1&1&2 \\ 3&5&8
\end{Bmatrix}
```

Los elementos de las matrices siempre aparecen centrados. Así

```
\begin{matrix}
 1 &       2&3&4 \\ 5&6&7&8 \\ 9&1000&11&12
\end{matrix}
```

genera

$$\begin{matrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 1000 & 11 & 12 \end{matrix}$$

Sin necesidad de cargar el paquete `amsmath`, hay un entorno llamado `array` que es más primitivo que `matrix` y obliga a especificar el tipo de alineamiento que uno quiere: centrado, justificado a la derecha o justificado a la izquierda. Lo veremos cuando estudiemos con detenimiento las tablas porque tiene un formato similar.

En ocasiones a uno le gustaría tener una matriz de tamaño menor que el habitual, por ejemplo para mencionarla en una línea de texto (aunque esto es en general muy poco aconsejable). Existe un entorno llamado `smallmatrix` que tiene este efecto. Por ejemplo

```
\[
 \left(\begin{smallmatrix} 0&1\\ 1&0\end{smallmatrix}\right)^2
 =
 \begin{pmatrix} 1&0\\ 0&1 \end{pmatrix}.
\]
```

produce

$$\left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right)^2 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

Hasta donde yo sé, en los paquetes habituales no hay definidos entornos que incorporen los delimitadores a `smallmatrix` por lo que hay que usar `\left` y `\right` con este fin.

# 3 Flechas

En matemáticas se usan a menudo flechas horizontales para indicar por ejemplo implicaciones, límites y funciones. Las más básicas son

$\leftarrow$ `\leftarrow` $\rightarrow$ `\rightarrow` $\leftrightarrow$ `\leftrightarrow`

La segunda es la misma que el `\to` de los límites. Hay ciertas modificaciones de estas flechas que se resumen en que se alargan si precedemos su nombre con `long` y se vuelven anchas como las habituales de implicación si ponemos en mayúsculas la primera letra. Por ejemplo

$$x \to y \Rightarrow 0 \leftarrow x - y \Longleftrightarrow f : A \longrightarrow B.$$

se obtiene con

```
x\rightarrow y
\Rightarrow
0\leftarrow x-y
\Longleftrightarrow
f:A\longrightarrow B.
```

Otra flecha a la derecha que se usa con cierta frecuencia es `\mapsto` con su variante de mayor longitud `\longmapsto` que son similares a lo que se obtendría con `\rightarrow`, pero con una pequeña barra vertical a la izquierda. En matemáticas se emplea para indicar que estamos considerando la imagen por una función de un elemento, no de todo el dominio. Por ejemplo `a\mapsto b` se ve como $a \mapsto b$.

Hay también flechas verticales que tienen unos nombres igualmente intuitivos.

$\uparrow$ `\uparrow` $\downarrow$ `\downarrow` $\updownarrow$ `\updownarrow`

El efecto de poner la primera letra en mayúscula es como antes:

$\Uparrow$ `\Uparrow` $\Downarrow$ `\Downarrow` $\Updownarrow$ `\Updownarrow`

pero no admiten el prefijo `long`.

Existen también flechas en direcciones de los puntos cardinales intermedios (noreste, etc.) las cuales se indican con sus siglas en inglés, por ejemplo `ne` corresponde a *northeast*.

$\nearrow$ `\nearrow` $\searrow$ `\searrow` $\swarrow$ `\swarrow` $\nwarrow$ `\nwarrow`

Hay otras muchas flechas definidas en los paquetes matemáticos usuales[1], pero es poco práctico intentar memorizarlas porque son muy poco frecuentes. Además, los editores normalmente incorporan una ayuda que muestra listas de símbolos. En mi caso en el apartado *Arrows* en la ayuda a la izquierda de Kile veo flechas como $\rightarrowtail$, $\circlearrowleft$, $\rightleftharpoons$, $\rightsquigarrow$ y otras muchas que creo no haber empleado nunca. Quien tenga curiosidad en una de estas cuatro en particular puede mirar la fuente de este documento. Si la curiosidad es mayor se debe ir a la documentación general de LaTeX provista por algún manual o a la específica de los paquetes matemáticos. Casi todas están en `amssymb`.

Un último apunte es que a veces uno querría una flecha, sobre todo a la derecha, de longitud suficiente para albergar cierto texto o explicación encima. Algo como

$$A \xrightarrow{3x^2+4y^2=z^2} B$$

Hay un comando `\xrightarrow` con tal fin, bastante desconocido incluso entre usuarios avanzados. Lo anterior se escribiría

[1] Es medianamente común entre los matemáticos cargar también el paquete `latexsym` que añade una flecha más.

```
A\xrightarrow{3x^2+4y^2=z^2}B
```

El comando admite un argumento opcional, que se indica entre corchetes, por si queremos poner la explicación por debajo. El análogo de lo anterior sería

```
A\xrightarrow[3x^2+4y^2=z^2]{}B
```

Nada impide usar ambos argumentos.

# 4 Recuperándose de los errores

Cuando uno comienza a usar LaTeX va muy lento en gran medida porque las posibilidades de cometer errores son grandes. Los editores actuales con sus colores y autocompletado ayudan bastante, pero es habitual al principio quedarse atascado porque el compilador se queja o se niega a producir un fichero de salida. Estos errores difíciles se producen casi siempre en las fórmulas porque es en ellas donde aparecen más comandos y la sintaxis es más intrincada.

Comencemos provocando un error con `\[ \frac{x+1}{y-1 \]` donde hemos olvidado la llave final. El editor (en este caso Kile) avisa

```
[LaTeX] finished with exit code 1
guia03.tex:0:File ended while scanning use \frac
```

que es bastante informativo. Aquí `guia03.tex` es el nombre del fichero y lo de `exit code 1` es un mensaje de error típico que salta cuando se ha llegado al final del documento sin terminar una tarea.

Aparte de los errores propiamente dichos hay también avisos. El más común en la redacción de un trabajo largo es `Overfull` y con menos frecuencia `Underfull`. En rigor, con la jerga al uso más que un aviso habría que decir que es una `badbox`. El corazón TeX de LaTeX tiene un poderoso algoritmo para distribuir las palabras en párrafos y habitualmente dispondremos en nuestro sistema de un diccionario que da indicaciones sobre la división de palabras en sílabas, pero hay veces en que la tarea de que las palabras quepan en una línea se vuelve imposible para LaTeX e indica una cantidad en puntos[2] en que ha excedido el espacio permitido. El problema suele aparecer cuando incluimos fórmulas en línea que siempre hay que evitar dividir. De todas formas vamos a forzarlo con texto escribiendo

```
\begin{verbatim}
 Estoy forzando a que respete la línea de la fuente y esta es muy larga
\end{verbatim}
```

Con el formato de este documento, en Kile el mensaje que aparece es:

[2] Un punto es cerca de un tercio de milímetro.

```
./guia03.tex:61:Overfull \hbox (48.15683pt too wide)
in paragraph
```

Lo que ha ocurrido es que el entorno `verbatim` obliga a LaTeX a respetar las líneas de la fuente, lo que es imposible sin sobrepasar el margen.

Sobre todo cuando tenemos una fórmula con varias líneas, una estrategia útil para localizar los errores más escondidos es usar el símbolo `%` para comentar una línea y que no tenga efecto. Por ejemplo

```
\[
 %%% Este es un comentario
 x+
 %y}+
 z
\]
```

muestra $x + z$ y no la $y$ que está en una línea que daría el error "`Extra }, or forgotten $`" por la llave de sobra.

La posibilidad de comentar unidades relativamente pequeñas es una razón más, aparte de la legibilidad, para distribuir el código de fórmulas largas en varias líneas.

Al compilar se genera un fichero `.log` que tiene mucha información. La gran mayoría es irrelevante, pero puede dar ciertos detalles finos de los errores, avisos y *overfulls*. Vaya por delante que esos detalles finos normalmente no son demasiado útiles para los principiantes.

En mi caso, al procesar con LaTeX (no PDFLaTeX) una versión casi vacía de este fichero obtuve en el `.log` como primera línea

```
This is pdfTeX, Version 3.1415926-2.5-1.40.14
(TeX Live 2013/TeX Live for SUSE Linux)
```

que da información sobre la versión de TeX empleada[3]. Después de muchas líneas de difícil interpretación se llega a un resumen de la memoria usada[4] La última línea fue

```
Output written on guia03.dvi (1 page, 1488 bytes).
```

lo que indica que se ha generado un fichero `dvi` que se puede visualizar con la herramienta que tengamos (okular, visor de Texworks...), habitualmente integrada en nuestro editor. Con PDFLaTeX pasó a ser

[3] D. Knuth tuvo la humorística idea de nombrar las versiones con cifras de $\pi$.

[4] Ahora resulta increíble, pero en los primeros tiempos recuerdo verme forzado a escribir párrafos no muy largos para que TeX no lanzara el mensaje fulminante `If you really absolutely need more capacity, you can ask a wizard to enlarge me`.

```
Output written on guia03.pdf (1 page, 113204 bytes).
```

seguido de cierta estadística acerca de las propiedades del PDF.

Si repetimos el error de la falta de llave en `\frac{x+1}{y-1` abriendo el `.log` con nuestro editor de texto preferido, incluso con el que usemos para LaTeX, veremos casi al final algo como

```
Runaway argument?
{y-1 \] \par \par \par \par \par \end {document}
\par \par \section \ETC.
! File ended while scanning use of \frac .
<inserted text>
                \par
<*> guia03.tex

I suspect you have forgotten a `}', causing me
to read past where you wanted me to stop.
I'll try to recover; but if the error is serious,
you'd better type `E' or `X' now and fix your file.

! Emergency stop.
<*> guia03.tex

*** (job aborted, no legal \end found)
```

El ``I suspect...'' es bastante ilustrativo. El resto no tanto y hace referencia a comandos de TeX como `\par`, para cambiar de párrafo, o `\end`, para terminar un fichero, que no son familiares a la mayor parte de los usuarios de LaTeX.

Al hilo del fichero `.log`, quizá te haya llamado la atención que hay otros ficheros que se crean al compilar. Siempre tendremos un `.aux` que contiene información sobre las referencias. Dependiendo de algunas herramientas que usemos veremos quizá un `.toc`, un `.bbl`, un `.out` y otros más. En TeXmaker seguramente también veas uno con `synctex` en la extensión. Excepto este último, que es para la comunicación con el editor para búsquedas inversas, el resto son ficheros auxiliares donde LaTeX almacena información auxiliar. Por ejemplo, el `.toc` es para la t*able of* c*ontents*. Como regla, todos se pueden borrar y el único precio a pagar es que quizá haya que compilar más de una vez para que se recupere esa información temporal. Con Kile, uno no se da cuenta de ello porque sabe cuántas veces debe compilar y lo hace sin preguntar[5]. Usando este editor, la única diferencia que notaremos si borramos esos ficheros auxiliares es que al compilar tarda más. Esto es solo apreciable para documentos muy grandes.

[5]Estrictamente, he visto excepciones fuera de `article` trabajando con `beamer`.

# Texto y tipos de letra

**Lección 4**

## 1. Texto y espaciamiento en fórmulas

A pesar de que algunos defiendan que las matemáticas son un lenguaje universal, lo cierto es que en los razonamientos matemáticos aparece una cantidad nada desdeñable de lenguaje natural. Dicho más llanamente, por mucho que lo parezca en la cultura popular, no todo en matemáticas son fórmulas, lo típico es que estén combinadas con palabras. Una primera idea ingenua para poner en práctica esta combinación en LaTeX es teclear el texto en modo matemático. Si, por ejemplo, escribimos las hipótesis del teorema fundamental del cálculo como

```
\[
 F(x)=\int_0^x f
 donde f es continua.
\]
```

Esto no puede funcionar porque LaTeX es incapaz de adivinar que "donde" no es el nombre de una función sino texto y que la segunda $f$ es una expresión matemática. El resultado sería:

$$F(x) = \int_0^x f donde f es continua.$$

Hay entonces dos cuestiones que resolver, la primera indicar qué cosas son texto y la segunda guardar el espaciamiento adecuado. Por si no estuviera claro ya, el modo matemático hace caso omiso de los espacios y si tratamos de hacer párrafos, lanza un mensaje de error.

La forma primitiva de solucionar lo primero es emplear `\mbox{...}`, pero en el paquete `amsmath` está definido `\text{...}` con el mismo propósito que tiene un nombre más sugestivo y por ello está más extendido entre los matemáticos. De esta forma una solución es

```
\[
 F(x)=\int_0^x f
 \text{ donde }f\text{ es continua}.
\]
```

que produce

$$F(x) = \int_0^x f \text{ donde } f \text{ es continua.}$$

Un comentario al margen es que `\text` y `\mbox` admiten encadenar modo matemático *inline* y de texto, entonces con el mismo fin también podríamos escribir `\text{ donde $f$ es continua}`.

Seguramente para el gusto de la mayoría, el "donde" está demasiado cerca de la integral y no hemos solucionado el problema del espaciamiento. Añadir espacios a mano por ejemplo con `\text{ donde }` no tiene ningún efecto porque una de las joyas de LaTeX en lo relativo al texto es que distribuye los espacios entre palabras automáticamente, independientemente del espaciamiento en el fichero fuente: produce prosa aunque escribamos poesía. Una solución de urgencia es utilizar la barra seguida de un espacio `\ ` que fuerza a que aparezca un espacio, pero claramente depender de cosas como `\text{\ \ \ \ \ donde }` para obtener un resultado aceptable no parece una solución seria. Dos comandos muy habituales para aumentar el espaciamiento especialmente, y casi exclusivamente, en las fórmulas son

`\quad` y `\qquad`

dando el primero cierto espaciamiento y el segundo su doble. Si miramos un manual[1] leeremos que `\quad` equivale al ancho de una "M" (mayúscula) con las fuentes que estemos usando para matemáticas. La realidad es que esta información tiene un interés relativo porque casi nadie tiene intuición en esos términos y porque en LaTeX las longitudes tienen habitualmente cierta holgura. Volviendo a nuestra fórmula, parece más aceptable

$$F(x) = \int_0^x f \qquad \text{donde } f \text{ es continua.}$$

que se consigue con `\qquad\text{donde }` etc.

Al hilo de los espacios, una línea con un solo espacio aparecerá naturalmente como una línea en blanco. Así a menudo en los documentos vemos cosas como

```
Una línea

\

Otra línea
```

para crear un doble espacio entre párrafos. De hecho, no es necesario teclear el espacio tras la barra, la barra sola tiene el mismo efecto. Más adelante en el curso, veremos cómo tener más control con el espaciamiento vertical.

[1] Por ejemplo en la magnífica ayuda de Overleaf `https://www.overleaf.com/learn/latex/Spacing_in_math_mode`.

Volviendo al espaciamiento horizontal, a veces se necesita introducir espacios mucho más pequeños. Por orden creciente de tamaño se indican con

$$\texttt{\textbackslash,} \qquad \texttt{\textbackslash:} \quad \text{y} \quad \texttt{\textbackslash;}$$

En la práctica las diferencias son tan poco apreciables que uno (al menos es mi caso) tiende a usar siempre lo mismo. Con estos miniespacios podemos resolver el problema ya mencionado con el diferencial en las integrales. Por ejemplo,

$$\int_0^1 xdx = \frac{1}{2}$$

se obtiene con `\int_0^1 x dx= \frac 12`. Ciertamente queda feo que $dx$ siga inmediatamente a $x$ y se arregla introduciendo un `\,` o alguno de los otros espacios pequeños antes de $dx$. Con cada uno de ellos los resultados serían respectivamente:

$$\int_0^1 x\,dx = \frac{1}{2}, \qquad \int_0^1 x\,dx = \frac{1}{2} \quad \text{y} \quad \int_0^1 x\;dx = \frac{1}{2}.$$

Los más puristas escriben `\text{d}x` en lugar de `dx` para que el diferencial tenga tipo de texto y no se confunda con una variable.

Aunque es mucho menos necesario, también se suelen utilizar estos espacios pequeños para separar los dos puntos que aparecen en la definición de conjuntos o de funciones. Este sería un ejemplo sin `\,`

$$A = \{n : \text{hay números con } n \text{ divisores}\}.$$

Lo mismo con `\,:\,` se vería así:

$$A = \{n \,:\, \text{hay números con } n \text{ divisores}\}.$$

Para dar un ejemplo más de espacios pequeños supongamos que queremos indicar la divergencia en una fórmula con div, como se suele hacer en Cálculo II. Lo ortodoxo, sobre todo si lo vamos a emplear a menudo, es definir un nuevo operador para que el espaciamiento sea automático, pero a nuestro nivel la solución rápida es ponerlo como texto. Así la regla del producto para la divergencia sería

```
\text{div}(f\vec{F})=f \text{div}\vec{F}+\vec{F}\cdot \nabla f
```

donde `\nabla` produce el símbolo del gradiente. El resultado es

$$\text{div}(f\vec{F}) = f\text{div}\vec{F} + \vec{F}\cdot\nabla f.$$

Si comparamos esto con $f\cos F$, obtenido mediante `f\cos F`, vemos que el espaciamiento de la segunda divergencia no es coherente con el de otras operaciones matemáticas. La solución es utilizar `\,\text{div}\,` que da un resultado visualmente más atractivo.

Existe también un `\,` negativo que se indica con `\!` y a veces se emplea para evitar *overfulls* mínimos o porque el espacio tras integrales o sumatorios, sobre todo si los límites son extensos, se nos hacen poco estéticos. Por ejemplo, en

$$\int_0^{\cos x} t\,dt \qquad \int_0^{\cos x} t\,dt \qquad \int_0^{\cos x} t\,dt \qquad \int_0^{\cos x} t\,dt$$

la primera fórmula es `\int_0^{\cos x}t\,dt` y en las siguiente se han introducido respectivamente uno, dos y tres `\!` antes de `t\,dt`.

Como seguramente intuirás, en LaTeX es también posible salirse de estos comandos predefinidos y lograr un ajuste manual fino de los espacios. El comando universal es `\hspace{...}` donde el argumento indica el espaciamiento y debe ir acompañado de cierta unidad. Quizá `mm` (milímetros) es la que te resulte más útil y familiar. En el mundo de la tipografía es muy común `pt` (puntos) que constituye una unidad menor (cerca de 1/3 de milímetro). Por ejemplo,

```
\[ f=g\circ h\hspace{40pt}\text{con $h$ continua}. \]
```

y

```
\[ f=g\circ h\hspace{80pt}\text{con $h$ continua}. \]
```

producen

$$f = g \circ h \qquad \text{con } h \text{ continua.}$$

y

$$f = g \circ h \qquad\qquad \text{con } h \text{ continua.}$$

El argumento de `\hspace{...}` puede ser negativo. Así, volviendo al ejemplo de la integral con límites anchos, con

```
\[
 \int_0^{\cos x}\hspace{-10pt}t\,dt
\]
```

conseguimos una invasión exagerada de la zona reservada al límite superior:

$$\int_0^{\cos x} t\,dt$$

Es muy importante no abusar de `\hspace` porque rellenar nuestro código de ajustes manuales lo hará poco legible y poco ajustable (por ejemplo si cambiamos el tamaño de letra). Además es una renuncia a las buenas capacidades automáticas de LaTeX.

Contradiciendo este último consejo, para finalizar, añadiré una forma un poco chapucera de añadir espacios con el comando `\phantom{...}` que sustituye lo que escribamos dentro por el espacio correspondiente a su longitud. A veces se usa en problemas relacionados con el alineado. Solo a modo de ilustración, con

```
\begin{pmatrix}
 123 & \cos \pi
 \\
 \phantom{12}3 & \phantom{\cos} \pi
\end{pmatrix}
```

lograremos romper la regla de que los elementos de las matrices siempre aparecen centrados por columnas, obteniendo:

$$\begin{pmatrix} 123 & \cos\pi \\ 3 & \pi \end{pmatrix}.$$

Por supuesto esto es una manera muy heterodoxa (y poco generalizable) de conseguir el resultado. Ya veremos qué es lo ortodoxo cuando estudiemos las tablas.

## 2. Fuentes de texto

En breve y sin todo rigor las fuentes de que disponemos sin instalar ni cargar nada son las siguientes:

| Comando | Nombre | Ejemplo |
|---|---|---|
| `\textrm{...}` | `roman` | Normal y corriente |
| `\textbf{...}` | `boldface` | **Negrita** |
| `\textit{...}` | `italic` | *Cursiva* |
| `\textsl{...}` | `slanted` | *Inclinada* |
| `\textsf{...}` | `sans-serif` | Palo seco (no es broma) |
| `\texttt{...}` | `typewriter` | `Courier` |
| `\textsc{...}` | `small caps` | VERSALITA |

El nombre en inglés de la fuente se ha incluido para ver que el comando proviene de una abreviatura.

En TEX no se usaba `\text`*ab* sino `\`*ab* con las llaves necesarias para indicar el ámbito. Esto también funciona en LATEX y así `\textit{Cursiva}` equivale a `{\it Cursiva}`. En realidad hay alguna diferencia de comportamiento en situaciones raras (conflicto entre fuentes) que exceden este nivel. Si se olvidan las llaves entonces el efecto de la versión TEX del comando se extiende desde el momento que se usa en adelante.

Para resaltar algo se utiliza `\emph` (de *emphasize*) que cambia el tipo de letra de forma automática. Así con `Es \emph{importante} recordar` y `\textit{Es \emph{importante} recordar}` obtenemos

Es *importante* recordar y *Es* importante *recordar*.

Aunque no sea una fuente de texto distinta, hablando de resaltar, el comando `\underbar{...}`, como su nombre indica, subraya. El análogo del ejemplo anterior es que con `Es \underbar{importante} recordar` y `\textit{Es \underbar{importante} recordar}` obtenemos

Es importante recordar y *Es importante recordar.*

El control del tamaño de la fuente se lleva a cabo con `{\`*tamaño* `...}` (o el entorno correspondiente) donde las posibilidades para *tamaño* están recogidas en las siguientes dos tablas:

| Comando | `\tiny` | `\scriptsize` | `\footnotesize` | `\small` |
|---|---|---|---|---|
| Ejemplo | Hola | Hola | Hola | Hola |

(hay también un `\normalsize`, pero, lógicamente, es raro usarlo)

| Comando | `\large` | `\Large` | `\LARGE` | `\huge` | `\Huge` |
|---|---|---|---|---|---|
| Ejemplo | Hola | Hola | Hola | Hola | Hola |

Terminemos con algunas maneras de importancia menor para un principiante de cambiar el comportamiento de la fuente de texto.

El entorno `verbatim` ya apareció en una anterior entrega y muestra de manera literal en la fuente `typewriter` todo lo que aparece dentro de él, respetando incluso espacios y renglones. El siguiente texto se escribió tal cual entre dos líneas con `\begin{verbatim}` y `\end{verbatim}`.

```
Seguro que alguien se mofa
si invocando al magisterio
medio en broma medio en serio
digo que esto es una estrofa.
```

Por supuesto, calidad aparte, esta no es la forma buena de escribir poesía, por la fuente y porque hay paquetes especiales para tal fin. Lo mismo se aplica a listados de programas. Posiblemente la principal utilidad de este entorno es escribir manuales de LaTeX. Si queremos usar algo similar, pero dentro de una línea, la alternativa es `\verb!...!`. En realidad podemos usar otros caracteres en vez de `!`, por ejemplo `\verb+...+`, lo que hay que tener cuidado es que ese carácter no aparezca en el interior para no hacer un lío al comando. Así escribiríamos `\verb!calcular \int!` y `\verb+¡calcular \int!+` para obtener `calcular \int` y `¡calcular \int!`

El entorno `verbatim*` hace lo mismo que `verbatim`, pero destacando los espacios con ␣ para que se vean mejor. Hay también un paquete `verbatim` que amplía las posibilidades.

# 3. Fuentes matemáticas

A pesar de que ya hemos aprendido suficiente para crear fórmulas muy complicadas, al copiar un fragmento de un texto de matemáticas pasaríamos un apuro con algo bastante básico: no sabemos cómo escribir las letras que representan los principales conjuntos de números como los reales o los enteros. Tampoco sabríamos reproducir la letra caligráfica que se usa con cierta frecuencia. En algunos temas un poco avanzados también hay tradición de utilizar letra gótica, a pesar de su legibilidad cuestionable. Estos tres tipos de letra se indican en LaTeX con

`\mathbb{...}`, `\mathcal{...}`, y `\mathfrak{...}`.

En caso de que te lo estés preguntando el `bb` es de *blackboard*, las letras tal como se escriben en la pizarra, y `frak` abrevia *fraktur*, el nombre inglés, calco del alemán, de un tipo común de letra gótica. Una observación importante es que `\mathbb` y `\mathcal` están reservados para letras mayúsculas, con minúsculas o números obtendremos símbolos extraños.

Por ejemplo

$$\mathbb{R}\times \mathbb{Q} \longrightarrow \mathcal{A} \longrightarrow \mathfrak{so}(3)$$

se obtiene con

```
\mathbb{R}\times \mathbb{Q}
\longrightarrow
\mathcal{A}
\longrightarrow
\mathfrak{so}(3)
```

Una tentación de los que comienzan a escribir textos matemáticos cuando se tienen todas las posibilidades de LaTeX es utilizar notaciones barrocas. Esto no es nada deseable si se quiere facilitar la claridad y legibilidad. Volviendo a las letras góticas, la "A" obtenida con `$\mathfrak{A}$` se ve como $\mathfrak{A}$ que cualquiera sin experiencia confundiría con una "U". ¿Sabrías por ejemplo qué letra es $\mathfrak{S}$ usada para indicar ciertas sumas?

Si haces un trabajo de fin de grado de matemáticas es muy probable que no utilices ni una sola letra gótica y es posible que pocas letras caligráficas, pero es muy raro que no aparezcan $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$ o $\mathbb{C}$. En algunos de estos trabajos aparecerá por doquier uno de estos conjuntos de números y se vuelve un poco pesado teclear todo el comando cada vez. Vamos a ver un avance de lo más básico de la definición de nuevos comandos. Si incluimos en la cabecera:

```
\newcommand{\R}{\mathbb{R}}
```

estaremos indicando a LaTeX que existe un nuevo comando llamado `\R` que tiene el mismo efecto que `\mathbb{R}` con ello ahorraremos pulsaciones tecleando por ejemplo $\mathbb{R}^n$ como `\R^n` en vez de como `\mathbb{R}^n`.

Lo mismo sirve con otras letras y seguro que muchos matemáticos, como yo, copian de una cabecera a otra la lista:

```
\newcommand{\N}{\mathbb N}
\newcommand{\Z}{\mathbb Z}
\newcommand{\Q}{\mathbb Q}
\newcommand{\R}{\mathbb R}
\newcommand{\C}{\mathbb C}
```

Nótese la pequeña variante con respecto a lo anterior omitiendo las llaves más interiores. Esto se debe a que, como ocurre con otros comandos, si no hay llaves `\mathbb`, `\mathcal` y `\mathfrak` actúan sobre el siguiente carácter distinto de un espacio.

En la tipografía habitual matemática las funciones y los números van en letra normal mientras que las variables van en cursiva. Por ejemplo, con `\tan(3x+y)` obtenemos $\tan(3x+y)$. En ocasiones muy especiales uno quiere variar este comportamiento haciendo que una fórmula o una parte de ella tenga un tipo de letra de texto distinto al predeterminado. Ciertamente esto no ocurre muy a menudo y, en general, es mejor evitarlo en la medida de lo posible. Por si sirve de ilustración, yo a veces lo hago en las fórmulas a pie de una imagen cambiándolas a `sans-serif` para distinguirlas del resto. La forma de modificar el tipo es emplear `\math`*ab* de forma totalmente similar a `\text`*ab* excepto que *ab* no pueden ser ni `sc` ni `sl`.

Por ejemplo, `f^3(x)=f_2(x)` daría el formato habitual $f^3(x)=f_2(x)$ y conseguiríamos

$$\mathsf{f^3(x)=f_2(x)}, \qquad \mathit{f^3(x)=f_2(x)} \qquad \text{y} \qquad \mathtt{f^3(x)=f_2(x)}$$

respectivamente con `\mathsf{f^3(x)=f_2(x)}`, `\mathit{f^3(x)=f_2(x)}` y `f^3\mathtt{(x)=f}_2(x)`.

# 4. Caracteres especiales

Hay una serie de caracteres que en LaTeX tienen un significado diferente al que vemos en el teclado, son:

```
#   $   %   &   ~   _   ^   \   {   }
```

Ya ha aparecido el significado de todos ellos excepto de `#` que se utiliza en la programación y de `~` que se utiliza para forzar que no se separen dos palabras consecutivas en líneas diferentes. A modo de recordatorio, esta es una tabla con los usos del resto:

| | |
|---|---|
| `$` | Modo matemático |
| `%` | Comentarios |
| `&` | Matrices |
| `_` | Subíndices |

| | |
|---|---|
| `^` | Superíndices |
| `\` | Comienzo de comando |
| `{` | Comienzo de bloque |
| `}` | Fin de bloque |

Como regla, si necesitamos escribir estos caracteres los precederemos con \ con la excepción de la propia barra \ que se obtiene en modo matemático mediante `\backslash` ya que la secuencia \\ está ocupada para su uso por ejemplo para indicar el cambio de fila en matrices. También `\^` tiene un comportamiento excepcional que veremos en breve. A veces algunos editores se hacen un pequeño lío con los colores cuando se usan estos comandos especiales, por ejemplo porque no distinguen bien `\%` de una línea de comentario. En cierto modo el espacio es también un carácter especial porque LaTeX no los respeta fielmente, da igual que pongamos uno o veinte. Por ello, como ya hemos visto, también admite la secuencia de escape `\␣` (barra seguida de espacio) similar a la de los otros caracteres especiales para forzar a que aparezca un espacio.

Un ejemplo es que escribiríamos `El 10\% de 20\$ son 2\$` para obtener "El 10 % de 20$ son 2$". Hablando de divisas, el uso de LaTeX se extendió en los años 90 antes de que se introdujera el euro y por tanto es natural que no incluya de serie un comando para mostrar su símbolo. El paquete más empleado para subsanarlo es `eurosym` que define el comando `\euro` y otros relacionados. En definitiva, si en nuestra cabecera incluimos

```
\usepackage{eurosym}
```

entonces `El 10\% de 20{\euro} son 2\euro` produce "El 10 % de 20€ son 2€". Si el símbolo te parece poco estético busca información sobre este paquete, por ejemplo con `\usepackage[gen]{eurosym}` obtendrás una tipografía algo distinta. El comando `\euro`, como es habitual en LaTeX, no distingue los espacios que vienen detrás y es por ello que en el primer `\euro` son necesarias las llaves, también con la variante `\euro{}`.

Para terminar esta sección de caracteres especiales y aunque no tenga demasiado que ver con lo anterior, hay unos "dígrafos" que se usan con cierta frecuencia. Con dos guiones se consigue un guion algo más largo que se utiliza por ejemplo para indicar un intervalo de páginas. Así con `véase pp.7--14` se obtiene "véase pp.7–14" mientras que con el guion simple el resultado sería "véase pp.7-14". El otro ejemplo a destacar aquí son las comillas. Si usamos las del teclado el resultado es muy feo, esencialmente porque LaTeX no sabe cuándo las estamos abriendo o cerrando. Hay editores adaptados (como Kile) que traducen automáticamente esas comillas en los símbolos correctos, siendo las impares de apertura y las pares de clausura. Si no disponemos de esta traducción automática, las primeras se indican con dos acentos graves (los contrarios a los usados en español que están en la tecla a la derecha de la `p`) y las segundas con dos apóstrofos (tecla a la derecha del `0`). Otra posibilidad más complicada, y menos común, es `\lq\lq` y `\rq\rq` (*left quotes* y *right quotes*). En la mayoría de los casos, el último `\rq` vendrá seguido de `{}`.

# 5. Internacionalización

Los acentos y letras propias de un idioma, como la eñe en nuestro caso, en los primeros tiempos del TEX había que teclearlas con secuencias especiales lo cual era bastante farragoso. Enseguida los editores adaptados incorporaron traducciones automáticas, de forma que cuando uno tecleaba "ñ" internamente en la fuente escribía `\~n` que es la secuencia correspondiente.

Afortunadamente, con el desarrollo de LATEX apareció un paquete que resolvía de manera bastante fiable los problemas de internacionalización. Es el paquete `babel` que podemos cargar en la cabecera con un idioma particular, en nuestro caso

```
\usepackage[spanish]{babel}
```

También podemos omitir `[spanish]` e indicar `spanish` en los parámetros de `\documentclass`. Este paquete redefinirá algunos operadores en modo matemático, por ejemplo el símbolo "lím" pasa a tener acento y define otros nuevos como `\sen` y `\tg` aunque `\sin` y `\tan` seguirán funcionando. Estos cambios a veces dan lugar a algún dolor de cabeza por incompatibilidades con otros paquetes. Por ejemplo, aunque casi nadie hagamos caso, la RAE recomienda que usemos las comillas españolas que son como pares de paréntesis angulares (*guillemets*) y consecuentemente `babel` permite que `<<hola>>` dé el resultado «hola», pero resulta que en el paquete más básico de diagramas conmutativos `<<` y `>>` están definidos de otra forma y este paquete dejará de funcionar con `babel`. Una manera un poco drástica de desactivar algunas de estas peculiaridades conflictivas es:

```
\usepackage[spanish,es-noquoting,es-noshorthands]{babel}
```

Otro tema es la coma decimal. Cargando `babel` con español la fuente `$3.14$` produce 3,14 lo cual nos puede incomodar (pensemos por ejemplo en un vector con coordenadas decimales separadas por comas). Una posibilidad para desactivar este comportamiento es poner `\decimalpoint` en alguna línea de la cabecera después de cargar el paquete.

Parece que la norma en español es que las abreviaturas de operadores se acentúan cuando el nombre completo lo está, por ejemplo lím. Si eso no te convence, existe la posibilidad de quitar los acentos globalmente con el comando `\unaccentedoperators` ubicado, de nuevo, en la cabecera después de la carga de `babel`. Si lo pusiéramos antes, haría saltar un error porque está definido en el propio paquete.

En `babel` hay también información acerca de cómo separar las palabras en sílabas por si acaso es necesario dividir alguna con un guion al final de una línea. Si nos hemos inventado una palabra nueva y queremos dar una indicación acerca de cómo dividirla lo podemos hacer introduciendo `\-` en los posibles lugares de separación. Eso no tendrá efecto en cómo vemos la

palabra si no se divide. Por ejemplo, con `svet\-loru\-menim` obtendremos "svetlorumenim" y en caso de tener que dividirla al final de una línea LaTeX tratará dejar en ella "svet-" o "svetloru-". El paquete `hyphenat` da más posibilidades para controlar la división de las palabras. De alguna forma el opuesto de `\-`, pero con pares de palabras es `~`. Si escribimos `tú~y~yo` estaremos indicando que la palabra `tú` vaya en la misma línea que `y` y en la misma que `yo`. Por supuesto si abusamos mucho de ello tendremos todas las papeletas para provocar *overfulls*. A menudo se usa esta virgulilla para evitar que un número o un carácter no alfabético quede al principio de una línea, lo cual resultaría feo. Por ejemplo, uno escribiría `Carlos~III`.

Incluso si solo tenemos pensado en escribir en inglés y en español, es fácil que tengamos ocasionalmente que mencionar nombres extranjeros que tengan una acentuación peculiar o letras especiales. Aquí hay algunos ejemplos que no es posible teclear directamente porque no aparecen en nuestro teclado o no son reconocidos por la opción en español. Ten en cuenta que, recíprocamente, si envías una fuente LaTeX a alguien y quiere usarla sin cargar el paquete de español y la codificación adecuada, debería reemplazar los acentos, etc. por unas secuencias de escape con no aparecen aquí.

| Nombre | Fuente |
|---|---|
| l'Hôpital | `l'H\^opital` |
| Erdős | `Erd\H os` |
| Sebastião | `Sebasti\~ao` |
| Français | `Fran\c cais` |
| Łaba | `\L aba` |
| Krzyżanowski | `Krzy\.zanowski` |

| Nombre | Fuente |
|---|---|
| Tụy | `T\d uy` |
| Gårding | `G\r arding` |
| Vrănceanu | `Vr\u anceanu` |
| Vopěnka | `Vop\v enka` |
| Størmer | `St\o rmer` |
| Yıldırım | `Y\i ld\i r\i m` |

El primer ejemplo ilustra que `\^` es un poco excepcional entre los caracteres especiales porque se superpone al siguiente. Si queremos un circunflejo solo podríamos emplear `\^\ ` para situarlo sobre un espacio. Los acentos graves y agudos se pueden teclear con un teclado español, pero solo sobre vocales, precedidos por la barra lograremos que aparezcan sobre otras letras. Por ejemplo "Sierpiński" se obtiene con `Sierpi\'nski`.

## 6. Acentos matemáticos

Aparte de los acentos propiamente dichos correspondientes a los diferentes idiomas, también hay lo que se podrían llamar "acentos" matemáticos que modifican a algunos símbolos. Los más usados son lo que normalmente llamamos *tilde* y *gorro*. Así es tradición que la transformada de Fourier de una función se escriba con un circunflejo sobre su nombre. Podríamos emular esto pasando provisionalmente a modo texto con el tipo de letra correspondiente y añadiendo el circunflejo como en la sección anterior. Por ejemplo

para obtener $\hat{f}(\xi)$ escribiríamos `$\textit{\^f}(\xi)$`. Es fácil sospechar que esto es demasiado complicado (aparte del espaciamiento raro). Así es, en LaTeX ya hay comandos que permiten decorar caracteres o expresiones con lo que podría calificarse como acentos. Los más empleados son:

```
\tilde    \hat    \bar    \underline    \dot    \ddot
```

Su efecto sobre la $f$, la notación típica para representar funciones, es

$$\tilde{f} \quad \hat{f} \quad \bar{f} \quad \underline{f} \quad \dot{f} \quad \ddot{f}$$

En todos los casos podemos indicar el ámbito de comando siguiéndolo de un bloque entre llaves, pero en la práctica eso es solo útil con `\underline`, por lo que se indica en el párrafo siguiente. Si deseamos subrayar más de una letra emplearemos `\underline{...}`.

Si repetimos el mismo ejemplo con una letra más ancha como la $m$ el resultado de los tres primeros no es muy estético:

$$\tilde{m} \quad \hat{m} \quad \bar{m} \quad \underline{m} \quad \dot{m} \quad \ddot{m}$$

Los resultados serían todavía menos aceptables si queremos que el "acento" matemático aparezca sobre más de una letra. Por ejemplo `$\tilde{wm}$` da lugar a $\tilde{wm}$. Para arreglarlo hay versiones ajustables de `\tilde` y `\hat` y para `\bar` se puede usar el hermano de `\underline`. Los comandos correspondientes son:

```
\widetilde    \widehat    \overline
```

Por ejemplo

```
\widetilde{w}(\xi)= \widehat{as}(\xi)=\overline{me}(\xi).
```

produce

$$\widetilde{w}(\xi) = \widehat{as}(\xi) = \overline{me}(\xi).$$

Por si te lo estás preguntando, no es adecuado emplear `\underbar` en modo matemático en vez de `\underline` porque es el método para subrayar textos y por tanto supone que lo de dentro está en este modo cambiando el tipo de letra. Por ejemplo, `$\underbar{f}$` da $\underbar{f}$ y `$\underbar{f^2}$` hace saltar un error porque `f^2` no es válido en modo texto.

No se me ocurre una situación natural en que uno quiera poner un acento normal (agudo) encima de una letra matemática, pero se puede emular fácilmente utilizando `\text` con el tipo de letra que aparece en la fórmula, que es típicamente cursiva.

# Referencias

**Lección 5**

## 1. Referencias a fórmulas

Una de las razones que propició que algunos pasáramos del primigenio TEX a LATEX fue la numeración automática de fórmulas y la posibilidad de referirse a ellas con un sistema flexible de etiquetas. Para que una fórmula aparezca numerada basta con que esté dentro de un entorno `equation`. Por ejemplo, si quiero que numere la fórmula de Euler para poliedros, escribiré

```
\begin{equation}
 V+C=A+2.
\end{equation}
```

que da lugar a

$$V + C = A + 2. \tag{1}$$

Una curiosidad es que el entorno `equation*` (añadir un asterisco tras `equation` en lo anterior) genera una fórmula no numerada y por tanto equivale a `\[...\]`. La única utilidad sensata que se me ocurre de ello es eliminar el número de forma rápida al revisar una versión añadiendo solo un par de asteriscos.

Cada vez que empleemos el entorno `equation` obtendremos un número consecutivo. Por ejemplo,

```
\begin{equation}
 x^n+y^n\ne z^n\qquad\text{si }n>2
 \quad\text{y}\quad x,y,z\in\mathbb{Z}-\{0\}
\end{equation}
```

genera

$$x^n + y^n \neq z^n \qquad \text{si } n > 2 \quad \text{y} \quad x, y, z \in \mathbb{Z} - \{0\} \tag{2}$$

La numeración por defecto aparece a la derecha. Muchos consideran que eso no es muy adecuado y puede llevar a confusiones en fórmulas largas porque el número puede considerarse parte de la fórmula con más probabilidad si aparece a la derecha. Para cambiarlo basta incluir el parámetro `leqno` (de `l`eft `e`quation `n`umero) en el `\documentclass`. Por ejemplo, el de este documento es

```
\documentclass[11pt,a4paper,leqno]{article}
```

Supongamos que queremos referirnos a una fórmula, que es la situación natural si la hemos numerado. Lo primero que debemos hacer es asignarle un nombre interno, ponerle una *etiqueta*. Esto se hace con `\label{...}`, por ejemplo podríamos escribir en la fórmula de Euler para los poliedros antes mencionada:

```
\begin{equation}\label{f_Euler}
 V+C=A+2.
\end{equation}
```

Un convenio muy purista que sorprendentemente tiene bastante aceptación entre los usuarios es que el nombre de la etiqueta nos dé indicaciones del tipo de objeto. Por ejemplo, que las ecuaciones, las únicas a las que sabemos hacer referencia por ahora, siempre lleven etiquetas que comiencen con `eq:` así el nombre purista anterior sería `\label{eq:f_Euler}`. Los siguientes objetos a los que se suelen hacer más referencias, bibliografía aparte, son secciones y figuras, cuyos nombres se preceden con `sec:` y `fig:` según este convenio. En mi opinión, esto es un poco exagerado porque la gran mayoría de las referencias son a fórmulas y no hay una ventaja apreciable en distinguir sus nombres de otros.

La manera básica de referirnos a una etiqueta es con `\ref{...}` que genera su número. Entonces al escribir `Por (\ref{f_Euler})` obtenemos "Por (1)". Este `\ref{...}` es genérico y sirve para otras cosas como números de teoremas o de secciones, pero muchísimas veces lo aplicaremos en nuestros documentos a fórmulas y por tanto la combinación `(\ref{...})` aparecerá muy a menudo. Por ello existe `\eqref{...}` que evita teclear los paréntesis, aunque debemos teclear dos caracteres más. De esta forma, lo anterior es totalmente equivalente a `Por \eqref{f_Euler}`.

En los textos largos es poco práctico (y poco estético) que las fórmulas solo tengan números consecutivos que pueden ser grandes. Imagina buscar la fórmula (307) en un documento de 1000 páginas. Para ayudar a localizarlas suelen llevar el número de sección o, en un libro, quizá también del capítulo. Para conseguir lo primero en un artículo con secciones, debemos incluir en la cabecera

```
\numberwithin{equation}{section}
```

Si lo haces en la fuente de este documento que estás leyendo, (1) y (2) pasarán a ser (1.1) y (1.2). En un documento sin secciones o con fórmulas anteriores a la primera sección, la numeración sería (0.1), (0.2), etc.

Al hilo de esto, una pequeña digresión que tiene interés en sí misma es que las secciones con su título se indican mediante `\section{...}` y admiten etiquetas al igual que las ecuaciones. También existen divisiones

más pequeñas que se obtienen con `\subsection` y `\subsubsection`, con estructura similar.

El epígrafe de la primera sección de este documento tiene como fuente

```
\section{Referencias a fórmulas}\label{laprimera}
```

y `\ref{laprimera}` generará un uno, porque es la sección número uno. Por ello al escribir

```
En \S\ref{laprimera} estudiamos las referencias
```

se obtiene "En §1 estudiamos las referencias". Como habrás adivinado, `\S` es la instrucción LaTeX para generar el símbolo de sección §.

Bajo la hipótesis de un texto medianamente largo, como puede ser tu trabajo de fin de grado, es fácil perderse y no recordar cómo había llamado uno a una fórmula que quizá incluyó hace semanas. Los que tengáis búsqueda inversa en vuestro editor[1] dispondréis de un método mejor que recorrer la fuente, pero, a mi juicio, es un poco mareante porque después hay que volver al lugar de partida. Una solución mucho más adecuada es el paquete `showkeys`. Yo lo suelo cargar en la cabecera con tres opciones:

```
\usepackage[notcite,notref,color]{showkeys}
```

Su efecto es mostrar en gris claro al lado de las fórmulas el nombre que le hemos asignado.

Con `\usepackage{showkeys}` sin parámetros veremos el nombre de la etiqueta también en cada cita que hagamos de ella (esto es lo que evita el `notcite,notref`) y además sale con el mismo color negro que el texto, lo que dificulta la legibilidad. El parámetro `color` es el que consigue que se muestre en un gris más discreto.

Un paquete que sirve para navegar entre las referencias en el PDF y que ha alcanzado gran popularidad[2] es `hyperref`. En concreto, si incluimos `\usepackage{hyperref}` en la cabera el "1" de "Por (1)" aparecería en el PDF dentro de un recuadro y si pinchásemos en él nos llevaría a la fórmula (1). A veces me he encontrado con algunos problemas de compatibilidad con otros paquetes o simplemente al compilar con LaTeX en vez de hacerlo con PDFLaTeX.

Hay una variante de `\ref{...}`, no muy usada en la práctica, que da el número de página donde se encuentra una etiqueta y es `\pageref{...}`. Así podríamos teclear

[1] La búsqueda inversa conecta al editor de la fuente LaTeX con el visor DVI o PDF de modo que al hacer click sobre algo en este tipo de ficheros lleva aproximadamente a la línea de la fuente que lo genera. En Overleaf está activada mientras que en algunos editores hay que hacer algún cambio en la configuración.

[2] Está cargado en la plantilla que se proporciona a los estudiantes de matemáticas para hacer su TFG.

```
Por la fórmula \eqref{f_Euler} de la página \pageref{f_Euler}
```

para obtener "Por la fórmula (1) de la página 39".

En TEX no había numeración automática, pero el comando `\tag{...}` en una *displayed formula* permitía ponerle una marca. Esto funciona en LATEX y a veces se emplea para señalar una fórmula sacándola de la numeración. Por ejemplo

```
{\sl Las fórmulas \eqref{normal1} y \eqref{normal2} son
normales y {\rm (*)} es rara:}
\begin{equation}\label{normal1}
 1+1=2,
\end{equation}
\[
 0<640320-\sqrt[3]{e^{\pi\sqrt{163}}-744}<10^{-24},
 \tag{*}
\]
\begin{equation}\label{normal2}
 2+2=4.
\end{equation}
```

Daría lugar a:

*Las fórmulas* (3) *y* (4) *son normales y* (*) *es rara:*

$$1 + 1 = 2, \tag{3}$$

$$0 < 640320 - \sqrt[3]{e^{\pi\sqrt{163}} - 744} < 10^{-24}, \tag{*}$$

$$2 + 2 = 4. \tag{4}$$

Se ha usado `{\rm (*)}` en lugar de simplemente `(*)` para que el tipo de letra coincida con en que aparece en la fórmula.

## 2. Referencias bibliográficas

Vaya por delante que cuando uno tiene un trabajo de cierta extensión lo mejor para elaborar una bibliografía es usar una herramienta llamada BibTEX que veremos más adelante en el curso. Esta opción, antes minoritaria, cada vez tiene más adeptos porque en los últimos años muchas fuentes bibliográficas han incorporado referencias en el formato BibTEX, lo que permite que en vez de teclear para incluir una referencia sea tan sencillo como un Ctrl-C por aquí y un Ctrl-V por allá.

Mientras tanto, vamos a ver la manera básica original sin utilizar ninguna herramienta. En la práctica también habrá mucho de Ctrl-C y Ctrl-V porque copiaremos títulos, etc. de internet, pero nadie nos librará de teclear, sobre todo para la tediosa tarea de unificar formatos. Desafortunadamente no hay un acuerdo medianamente general acerca de los tipos de letra de los autores, títulos, etc. o sobre el lugar donde debe aparecer el año de publicación o si las iniciales del nombre deben preceder o no al apellido. Cada editorial tiene sus propias reglas a las que deben adaptarse los autores que envíen sus trabajos en LaTeX.

La norma es que las referencias o bibliografía aparezcan al final de un texto, aunque esto no es obligatorio en LaTeX. El entorno para incluir una lista de referencias es `thebibliography` y a diferencia de otros entornos que hemos visto, necesita un parámetro adicional. Si examinamos fuentes de otras personas normalmente veremos un `9` o un `99`. Por ejemplo

```
\begin{thebibliography}{9}
 ...
\end{thebibliography}
```

donde ahora veremos cómo rellenar los puntos suspensivos. Este `9` o `99` o lo que queramos escribir da una pista al editor del ancho que ocupará el nombre de nuestra referencia. Por defecto este nombre es un número y por ello a menudo se escribe un `9` si planeamos menos de 10 referencias (un dígito) y `99` si planeamos más de 9 y menos de 100 referencias (dos dígitos), y así sucesivamente. Lo importante no es el número en sí sino cuantos caracteres tiene.

Dentro de los puntos suspensivos vienen las referencias bibliográficas precedidas por `\bibitem{...}` donde dentro de las llaves incluimos la etiqueta que queramos asignar. En cierto modo `\bibitem{...}` es el `\label{...}` para referencias bibliográficas. Por ejemplo el apartado de referencias de este documento ha sido creado con:

```
\begin{thebibliography}{9}
\bibitem{knuth}
D. E. Knuth. \emph{The \TeX{} Book}.
Addison-Wesley Professional, 1986.

\bibitem{lamport}
L. Lamport. \emph{\LaTeX: A document preparation system}.
Addison
Wesley, Massachusetts, 2nd ed, 1994.

\bibitem{spivak}
M. Spivak. \emph{The Joy of \TeX: A Gourmet Guide to
Typesetting With the {\AmS-\TeX} Macro Package}.
```

```
Addison-Wesley Professional, 1990.
\end{thebibliography}
```

Dicho sea de paso, estas son tres referencias famosas en la historia del TEX aunque a día de hoy anticuadas. Las dos primeras se deben a los creadores del TEX y LATEX respectivamente. La última fue un manual muy influyente[3] para la difusión de la modalidad más extendida de TEX (yo lo aprendí con él).

Para citar una de estas referencias utilizaremos `\cite{...}` con el nombre de la etiqueta. Siguiendo el paralelismo anterior, `\cite{...}` hace las veces de `\eqref{...}` para referencias bibliográficas. Por ejemplo, con

```
Mi libro preferido de {\TeX} es \cite{spivak}
```

se obtiene "Mi libro preferido de TEX es [3]". Los corchetes evitan la confusión con las referencias a fórmulas.

El comando `\cite` admite un argumento en el que se puede incluir algo que queramos que aparezca dentro del corchete de la cita. Normalmente es un número de página, sección, de teorema, etc. Por ejemplo con `\cite[p.\,15]{lamport}` se obtiene [2, p. 15].

Por otro lado, `\bibitem` también admite un argumento opcional que indica un texto que reemplaza al número. Por ejemplo, el tercer libro aparecería en la bibliografía y las veces que lo citemos como [Sp] si empleamos `\bibitem[Sp]{spivak}`. Es poco estético combinar números y otros nombres. De todos modos, si lo hiciéramos, los otros nombres serían saltados en la cuenta.

## Referencias


[1] D. E. Knuth. *The TEX Book*. Addison-Wesley Professional, 1986.

[2] L. Lamport. *LATEX: A document preparation system*. Addison Wesley, Massachusetts, 2nd ed, 1994.

[3] M. Spivak. *The Joy of TEX: A Gourmet Guide to Typesetting With the $\mathcal{AMS}$-TEX Macro Package*. Addison-Wesley Professional, 1990.


[3] Parte de su éxito se debe a un estilo muy gracioso. El título es una variación de *The Joy of Sex*, un famoso libro de educación sexual de los años 70 (que a su vez es un guiño al libro de cocina *The Joy of Cooking*) y en el texto juega con dobles sentidos a este respecto, además de hacer referencias culinarias. Por supuesto no llega al nivel de escandalizar, es apto para todos los públicos.

# Alineamiento y centrado

**Lección 6**

## 1. Alineamiento de fórmulas

A menudo al escribir un texto matemático hay que alinear o justificar fórmulas. En este aspecto, LaTeX "puro" es un poco deficitario. Afortunadamente en el paquete `amsmath` están definidos varios entornos con este propósito. La filosofía general de todos ellos es que, al igual que con las matrices, con `&` indicamos alinear y con `\\` cambiar de línea.

Supongamos por ejemplo que queremos escribir la demostración de

$$S = 1 + 2 + \cdots + n \quad \Rightarrow \quad S = \frac{n(n+1)}{2}.$$

Para ello, deseamos mostrar el bello procedimiento habitual de sumar la expresión para $S$ consigo misma invirtiendo el orden de sus términos. Con este fin, escribimos los dos pasos en fórmulas separadas:

```
\[
 2S
 =
 (1+n)+(2+n-1)+\dots+(n+1),
\]
\[
 =n(n+1).
\]
```

El resultado no es estético por la falta de alineamiento:

$$2S = (1 + n) + (2 + n - 1) + \cdots + (n + 1),$$

$$= n(n + 1).$$

A esto se unen problemas más sutiles acerca del espacio vertical entre las fórmulas y la posibilidad de un salto de página entre ellas. Por cierto, de paso hemos aprendido que los puntos suspensivos se obtienen con `\dots`. En el modo matemático muestran cierta inteligencia, así en el ejemplo anterior se han elevado automáticamente para linearse verticalmente con los dos signos +.

El paquete `amsmath` incorpora el entorno `align`. Usándolo, si precedemos los símbolos "=" con un `&` indicaremos el alineamiento. De alguna forma, la sintaxis es la de una una matriz $N \times 2$ donde $N$ es el número de ecuación y el 2 se debe a que hay dos miembros en cada ecuación, el primero se justificará a la derecha y el segundo a la izquierda. El entorno `align` actúa como si incluyera dentro de su definición un entorno `equation`[1] por tanto las fórmulas aparecerán numeradas y no hay que usar `\[...\]` ni tampoco `\begin{equation}...\end{equation}`, que producirían errores. En definitiva, con

```
\begin{align}
 2S
 &= (1+n)+(2+n-1)+\dots+(n+1),
 \\
 &= n(n+1).
\end{align}
```

conseguimos

$$\begin{aligned} 2S &= (1+n)+(2+n-1)+\cdots+(n+1), && (1)\\ &= n(n+1). && (2) \end{aligned}$$

El alineamiento no está limitado a dos líneas, por ejemplo

$$\begin{aligned} 2S &= (1+n)+(2+n-1)+\cdots+(n+1), && (3)\\ &= (n+1)+(n+1)+\cdots+(n+1), && (4)\\ &= n(n+1). && (5) \end{aligned}$$

se obtiene insertando

```
 \\
 &= (n+1)+(n+1)+\dots+(n+1),
```

justo antes del `\\` en el ejemplo anterior. La justificación a la derecha de los primeros miembros se observa en la siguiente variante que proviene del código obvio:

$$\begin{aligned} S &= 1+2+\cdots+(n-1)+n, && (6)\\ S &= n+(n-1)+\cdots+2+1, && (7)\\ S+S &= (1+n)+(2+n-1)+\cdots+(n+1), && (8)\\ 2S &= (n+1)+(n+1)+\cdots+(n+1), && (9)\\ 2S &= n(n+1). && (10) \end{aligned}$$

[1]Hay una versión llamada `aligned`, que no veremos aquí, la cual no incluye el modo matemático. Se utiliza para expresiones de cierta complejidad en las que hay varios alineamientos independientes. Sí veremos `split` que está relacionado. Si tienes curiosidad, puedes consultar la documentación del paquete `amsmath`.

La forma original de LaTeX de resolver el alineamiento es el entorno `eqnarray` que funciona de manera similar salvo que hay que incluir entre dos `&` el término por el que queremos alinear. En el ejemplo anterior sería el signo igual. Este entorno es fuente de muchas críticas, pero es tan antiguo que todavía se ve con cierta frecuencia. Esta es la única razón para mencionarlo aquí. En muchos manuales, comenzando por la documentación de `amsmath`, leerás que debes abstenerte de usarlo. El análogo con `eqnarray` del primer ejemplo que hemos visto sería:

```
\begin{eqnarray}
 2S
 &=& (1+n)+(2+n-1)+\dots+(n+1),
 \\
 &=& n(n+1).
\end{eqnarray}
```

que da lugar a

$$\begin{aligned} 2S &= (1+n)+(2+n-1)+\cdots+(n+1), && (11)\\ &= n(n+1). && (12)\end{aligned}$$

Aquí se ve la razón de una de las críticas hacia `eqnarray` que motivan no emplearlo: el espaciamiento alrededor del símbolo "=", el que hemos alineado, no es coherente con lo que observamos en otras fórmulas. En principio no hay ninguna razón para cambiar el espaciamiento en una fórmula solo por el hecho de que la pongamos encima o debajo de otra. Hay además algún otro problema con numeración de las fórmulas en situaciones extremas y con la manera de asignar etiquetas. En definitiva, `eqnarray` te debería servir solo para leer código de otras personas y si lo copias, es preferible que lo cambies por `align` con la correspondiente modificación en la sintaxis.

En ambos entornos quitaremos la numeración añadiendo al nombre un asterisco, es decir, empleando `align*` y `eqnarray*`, esto es similar a lo visto con `equation`. Si lo que queremos es omitir solo algunos números, emplearemos `\notag` o `\nonumber` en la línea correspondiente, cualquiera de los dos vale siempre que tengamos cargado `amsmath`. Por ejemplo,

$$\begin{aligned} 2S &= (1+n)+(2+n-1)+\cdots+(n+1),\\ &= n(n+1). && (13)\end{aligned}$$

se obtiene con

```
\begin{align}
 2S &= (1+n)+(2+n-1)+\dots+(n+1), \nonumber
 \\
 &= n(n+1).
\end{align}
```

Cada una de las subfórmulas alineadas con `align` admite sus etiquetas y sus referencias independientes. Si en el caso anterior añadimos la etiqueta `\label{eq:fin}` tras `&= n(n+1).`, al escribir `Por \eqref{eq:fin}` obtendremos "Por (13)".

Una ventaja más de `align` sobre el original `eqnarray` es que permite alineamiento múltiple. Antes de dar un ejemplo, pasemos por otras dos formas de alineamiento simple. Una es el entorno `split` que es similar a `align`, pero hay que escribirlo dentro de un entorno matemático `\[...\]` o `equation` y cuando se usa este último, para la numeración se considera la fórmula como un todo, apareciendo un solo número. Por ejemplo,

```
\begin{equation}
  \begin{split}
  2S &= (1+n)+(2+n-1)+\dots+(n+1),
  \\
  &= n(n+1).
 \end{split}
\end{equation}
```

genera

$$\begin{aligned} 2S &= (1+n)+(2+n-1)+\cdots+(n+1), \\ &= n(n+1). \end{aligned} \tag{14}$$

Nota que el número queda entre las dos líneas.

El otro tipo de alineamiento es el que utilizaríamos para definir una función a trozos. Con este fin podríamos combinar los delimitadores `\left\{` y `\right.` con el entorno `array` que sirve para hacer tablas matemáticas o con `matrix`, que daría un resultado menos correcto, pero hay una solución más directa que es el entorno `cases`. Por ejemplo la función $f(x) = |x| + |x-1|$ la escribiríamos a trozos como

$$f(x) = \begin{cases} -2x+1 & \text{si } x \le 0, \\ 1 & \text{si } 0 < x \le 1, \\ 2x-1 & \text{si } x > 1. \end{cases}$$

Lo cual se consigue con la fuente:

```
f(x)=
\begin{cases}
 -2x+1 &\text{si }x\le 0,
 \\
 1 &\text{si }0<x\le 1,
 \\
 2x-1 &\text{si }x> 1.
\end{cases}
```

En breve, el entorno `cases` funciona como una matriz de dos columnas con justificación a la izquierda.

Al hilo de las fórmulas que ocupan varios renglones, aunque no es un alineamiento (no contiene `&`), el entorno `multline`, como su nombre indica, permite dividir en múltiples líneas. También admite la variante `multline*` para no numerar. Solo hay que indicar el lugar o lugares donde queremos dividir la fórmula. Por ejemplo,

```
\begin{multline}
 210 = 1+2+3+4+5+6+7+8+9+10\\
 +11+12+13+14+15+16+17+18+19+20
\end{multline}
```

produce

$$\begin{multline} 210 = 1+2+3+4+5+6+7+8+9+10 \\ +11+12+13+14+15+16+17+18+19+20 \end{multline} \tag{15}$$

Añadiendo otro `\\` tras `+15` y cambiando `multline` por `multline*` el resultado es:

$$\begin{multline*} 210 = 1+2+3+4+5+6+7+8+9+10 \\ +11+12+13+14+15 \\ +16+17+18+19+20 \end{multline*}$$

Después de este receso volvamos a `align`. También sirve para obtener varias columnas de fórmulas alineadas. Su uso es como antes con el añadido de que las diferentes columnas deben estar separadas por un nuevo `&`. Lo mejor para entenderlo es ver un ejemplo. Consideremos

$$\begin{aligned} 3 &= 1+1+1, & 3 &= 2+1, & x &= 2023, \\ 2 &= 1+1, & 2 &= 2, & y &= 2024, \\ 1 &= 1, & 1 &= 1, & x+y &= 4047. \end{aligned}$$

Lo podemos obtener con

```
\begin{align*}
 3 &= 1+1+1, &  3 &= 2+1, &  x &= 2023,
 \\
 2 &= 1+1,   &  2 &= 2,   &  y &= 2024,
 \\
 1 &= 1,     &  1 &= 1,   &  x+y &= 4047.
\end{align*}
```

## 2. Justificación y párrafos en texto

La regla general es que debemos dejar hacer a LaTeX en lo relativo a la justificación de los párrafos. El comando `\break` fuerza la ruptura de una línea en un punto y también se pueden hacer chapucillas introduciendo espacios pequeños para que cierta palabra pase a otra línea, pero lo habitual es que esto produzca resultados muy poco estéticos y la única libertad que deberíamos permitirnos es el uso de algunos `\-` para sugerir posibles divisiones de palabras y de algunos `~` para ligar palabras, sin abusar de ello. La justificación de LaTeX tiende a que los espacios sean homogéneos y las líneas tengan la misma longitud. La única excepción habitual a esta justificación es el centrado que resulta de utilidad para poner un título o resaltar una porción de texto. Con este fin, basta incluir en un entorno `center` el párrafo que queramos que aparezca con justificación centrada. Por ejemplo,

```
\begin{center}
 \Huge
 Este es el título de mi magnífico trabajo
\end{center}
```

genera

# Este es el título de mi magnífico trabajo

Seguramente nos resulte fea la división en líneas: la palabra "trabajo" no ha cabido en la primera línea y queda aislada en la siguiente. La forma de arreglarlo es introducir tras la palabra "de" un `\\` que indica un cambio de línea, como ya sugería el formato de las matrices. En este contexto `\\` es equivalente al `\break` antes mencionado, aunque no en general[2]. Entonces escribiremos:

```
\begin{center}
 \Huge
 Este es el título de
 \\
 mi magnífico trabajo
\end{center}
```

[2]Si usamos la doble barra en un párrafo normal no centrado se cambiará de línea como si hubiera un punto y aparte, pero sin introducir sangría (el espacio de separación inicial a la derecha) mientras que con `break` solo indicamos que hay que cambiar de línea por ahí, pero las palabras de antes y de después estarán justificadas tratando de llenar todo el renglón.

que da lugar a

# Este es el título de mi magnífico trabajo

Separar `Este es el título de\\ mi magnífico trabajo` en tres líneas en el código ha sido solo para facilitar la legibilidad de la fuente (esto es, manías mías).

Los entornos `flushleft` y `flushright` aplican justificación a la izquierda y a la derecha. Son de mucha menor relevancia práctica que `center` y apostaría a que hay muchos usuarios medianamente avanzados que ni los conocen. Por ejemplo, con el paquete `lipsum`[3],

```
\begin{flushright}
\it\lipsum[4]
\end{flushright}
```

da lugar a:

*Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.*

Una posible aplicación es incluir una firma al final de un documento. Con

```
\begin{flushright}
 Madrid 23 de octubre
 \\
 Perico el de los Palotes
\end{flushright}
```

se obtiene

Madrid 23 de octubre
Perico el de los Palotes

[3]Este paquete tiene cierta incompatibilidad con `babel`. Recuérdalo si te salen errores raros al emplearlo. En internet puedes ver un pequeño código para evitarlo que he copiado en la cabecera de la fuente de este documento.

Por supuesto en la práctica querremos dejar un espacio para la firma. Vamos a ver dos maneras de hacerlo. La primera se basa en que `\\` admite un argumento opcional que indica la altura a añadir al salto de línea. Hay muchas unidades posibles, entre las que te resultan familiares están `cm` y `mm`. Así podríamos escribir `\\[1.5cm]` o `\\[15mm]` para obtener

Madrid 23 de octubre

Perico el de los Palotes

En general yo prefiero la unidad más pequeña `pt` (punto de impresión) porque es más precisa con números enteros y me he acostumbrado a ella. Equivale a unos 0.35 *mm* por tanto algo casi equivalente a lo anterior sería emplear `\\[43pt]`. El argumento de `\\` puede ser una longitud negativa aunque, por supuesto, en nuestro ejemplo no tiene sentido.

Otra manera de ajustar el espacio a nuestras necesidades es emplear el comando `\vspace{...}` que es el análogo vertical de `\hspace{...}` que ya habíamos visto. Es importante no abusar de estos comandos. Los puntos suspensivos indican una longitud, positiva o negativa, como en el argumento de `\\`. Un ejemplo de este control universal del espacio es:

```
\hspace{15pt}a\hspace{10pt} b
\vspace{7pt}

c\hspace{-13pt} d
```

que produce:

a b

dc

Nótese la inversión de "c" y "d". Si no hubiéramos dejado una línea en blanco tras `\vspace` (o antes) no funcionaría. Sin entrar en detalles, eso se debe a que hay que estar en el modo vertical de TEX para que funcione y este modo se activa con cambios de línea. En el ejemplo de la firma, podríamos poner `\vspace{1.5cm}` tras de `\\` o dejar como aquí una línea en blanco.

En relación con el espacio vertical en texto, recordemos que dejar una línea en blanco produce un cambio de línea lo que incorporará una sangría de comienzo de párrafo con nuestro documento `article`, por cierto para impedirlo en un párrafo en particular basta preceder la nueva línea con `\noindent`. Recordemos también que si queremos un espacio vertical mayor, lo que se llama doble espacio, podemos crear una línea falsa con un solo espacio representado mediante `\ ` (con la barra sola sirve).

Existen también los comandos

`\smallskip`, `\medskip` y `\bigskip`

que añaden 3, 6 y 12 puntos al salto de línea básico (esto es alrededor de 1, 2 y 4 milímetros). Una cosa importante a tener en cuenta es que estas longitudes, como otras en LaTeX, tienen holgura. Esto significa por ejemplo en el caso de `\bigskip` que si el algoritmo se ve muy apurado en la distribución de párrafos en un página (por ejemplo para no dividir una matriz entre dos páginas) entonces puede reducir su tamaño hasta 8 puntos o aumentarlo hasta 16. Lejos de ser un defecto, la holgura permite que el resultado sea más equilibrado de lo que se obtendría con algunos procesadores de texto al uso. Al igual que con `\vspace`, estos comandos deben seguirse de una línea en blanco, es decir, debemos escribir

```
Una línea
\bigskip

Otra línea
```

Si no la dejásemos, lo que veríamos es "Una línea Otra línea".

A caballo entre el alineamiento y el espaciamiento está `\hfill` que es un espacio horizontal que tiene una holgura positiva arbitrariamente grande. Lo podemos usar por tanto para completar una línea con espacios. Imaginemos que queremos poner dos firmas en nuestro documento. Una posibilidad es utilizar

```
\noindent
Madrid 23 de octubre\hfill Madrid 23 de octubre
\\[15mm]
Perico el de los Palotes \hfill Fulanito de Tal
```

El `\noindent` es para impedir la sangría inicial al cambiar de párrafo. El resultado sería:

Madrid 23 de octubre Madrid 23 de octubre

Perico el de los Palotes Fulanito de Tal

Posiblemente el alineamiento de la segunda firma no nos agrade mucho. Una solución chapucera es introducir un `\hspace{...}` tras "Tal" y la otra más ortodoxa (aunque fuera de nuestros conocimientos actuales) es meter los bloques de firma en tablas separadas con un solo `\hfill`.

Existen también las variantes `\hrulefill` y `\dotfill` que completan con una línea horizontal o con puntos. Por ejemplo, con

```
\label{thispag}Esta página \dotfill \pageref{thispag}

Esta página \hrulefill \pageref{thispag}
```

Se obtiene:

Esta página ..... 54
Esta página _____ 54

Más realista es introducir `\hrulefill` en una línea en solitario para separar dos partes de un documento. Sin hacer nada especial la barra es baja por tanto es posible que deseemos emplear

```
Una línea

\hrulefill

\

Otra línea
```

para conseguir dicho separador o quizá cambiando el `\ ` por `\bigskip`.

Existe también un `\vfill` para jugar con espaciamientos verticales. Solo diré que si nuestro propósito es únicamente dejar el resto de la página en blanco y pasar a otra, tenemos `\newpage`.

## 3. Más control vertical en fórmulas

Es posible también controlar espacios verticales en fórmulas. Por cuestiones relacionadas con el modo vertical, `\vspace{...}` no se vuelve muy útil. Para algún ajuste fino se puede usar `\raisebox{...}{...}` donde el primer argumento es la medida y el segundo sobre la que se aplica. En realidad opera sobre cajas de texto y por tanto debemos indicar el modo matemático con `$...$` si queremos mover una fórmula. No parece que sea muy realista tener que jugar con espacios verticales en fórmulas, por tanto el uso de `\raisebox{...}{...}` es anecdótico y, muchas veces, no demasiado aconsejable. Aquí va un ejemplo:

$$x = a^{\mathrm{bc}}{}_{\mathrm{d}} \qquad y = a^{bc}{}_{d}.$$

que se obtiene con

```
\[
 x =  a\raisebox{2pt}{bc}\raisebox{-1pt}{d}
```

```
 \qquad
 y =  a\raisebox{2pt}{$bc$}\raisebox{-1pt}{$d$}.
\]
```

En el primer caso la tipografía de las letras movidas es la de texto y en la segunda la de matemáticas.

Por lo dicho, está claro que `\raisebox` se puede usar en modo texto. Obtendremos el efecto especial l$\raisebox{1pt}{í}\raisebox{1pt}{n}\raisebox{1pt}{e}\raisebox{1pt}{a}$ con

```
\raisebox{1pt}{l\raisebox{1pt}{í\raisebox{1pt}{n%
\raisebox{1pt}{e\raisebox{1pt}{a}}}}}
```

El % es irrelevante, es solo porque no cabía en la línea en este documento. Es para evitar que la `n` se "contamine" con algún posible espacio posterior.

Fuera de estos efectos de fantasía, a veces para dar una explicación nos gustaría que algo apareciera en un tipo menor sobre una expresión matemática o por debajo de ella, como vimos con `\xrightarrow`. Con este fin están los comandos `\overset` y `\underset` que requieren dos argumentos: el primero, la expresión que queremos poner por encima o por debajo y el segundo, la que tomamos como base. Por ejemplo, si $x + y = 4$ y queremos recordar $y = 3$ para deducir $x = 1$ lo podremos escribir como

$$x + y = 4 \underset{y=3}{\Longrightarrow} x = 1$$

empleando

```
\[
 x+y=4 \underset{y=3}{\Longrightarrow} x=1
\]
```

Otro ejemplo plausible es

```
\[
 1+\overset{n\text{ veces}}{\cdots\cdots}+1=n.
\]
```

que produce

$$1 + \overset{n \text{ veces}}{\cdots\cdots} + 1 = n.$$

La razón para usar `\cdots` es conseguir un centrado vertical más adecuado que con `\dots`. Hay ciertas reglas por las que el paquete `amsmath` decide cuándo `\dots` es lo mismo `\cdots` y cuándo no, de modo que muchas veces basta escribir `\dots` sin preocuparse. Normalmente acierta, son las llaves de `\overset` las que le confunden aquí.

# Tablas y listas

**Lección 7**

## 1. Más allá de las matrices en modo matemático

El entorno `matrix` deriva de otro llamado `array` con el que podemos especificar el alineamiento de las columnas mediante los caracteres `l`, `c` o `r` que indican justificado a la izquierda, al centro o a la derecha. Se ha de poner uno por columna en un bloque entre llaves justo después de abrir el entorno. Para ilustrar el uso de `array` y su comparación con `matrix` consideremos

```
\begin{pmatrix}
 1 & 1 & 1 \\ 12 & 12 & 12 \\ 123 & 123 & 123
\end{pmatrix}
\qquad
\left(
\begin{array}{lcr}
 1 & 1 & 1 \\ 12 & 12 & 12 \\ 123 & 123 & 123
\end{array}
\right)
```

que da lugar a

$$\begin{pmatrix} 1 & 1 & 1 \\ 12 & 12 & 12 \\ 123 & 123 & 123 \end{pmatrix} \qquad \left(\begin{array}{lcr} 1 & 1 & 1 \\ 12 & 12 & 12 \\ 123 & 123 & 123 \end{array}\right)$$

Nótese que con `array` hay una separación de los paréntesis algo mayor que, si somos muy maniáticos, podríamos corregir con `\hspace`.

La mayor parte de las veces el propósito de `array` no es escribir matrices numéricas con justificaciones exóticas sino colocar fórmulas. Por ejemplo, habíamos visto ya cómo el entorno `cases` nos permitía definir cómodamente funciones a trozos. Por ejemplo, seguro que te resulta matemáticamente más intuitivo expresar $f(x) = |x| + |x-1|$ como

$$f(x) = \begin{cases} -2x+1 & \text{si } x \leq 0, \\ 1 & \text{si } 0 < x \leq 1, \\ 2x-1 & \text{si } x > 1. \end{cases}$$

Pero si no estamos de acuerdo con la justificación no tenemos libertad para cambiarla con este entorno. Utilizando `array` en la forma

```
f(x)=
\left\{
\begin{array}{rl}
 -2x+1 &\text{si }x\le 0,
 \\
 1 &\text{si }0<x\le 1,
 \\
 2x-1 &\text{si }x> 1.
\end{array}
\right.
```

conseguimos un resultado con una estética alternativa:

$$f(x)=\left\{\begin{array}{rl} -2x+1 & \text{si } x\le 0,\\ 1 & \text{si } 0<x\le 1,\\ 2x-1 & \text{si } x>1. \end{array}\right.$$

Si alguien juzga que la llave está demasiado separada puede insertar tras `\left\{` un `\hspace{-5pt}` o alguna otra cantidad, también nos puede parecer que los renglones están muy juntos y quizá prefiramos cambiar los `\\` por `\\[2pt]`. Recuérdese no obstante que no hay que abusar de los espacios absolutos. Casi siempre lo más cómodo y lo mejor es dejar hacer a LaTeX.

Otro uso de `array` es crear tablas que tengan mucho contenido en modo matemático. Sobre todo en ellas aparecen a veces ciertas finuras que no hemos considerado con el uso básico anterior, por ejemplo romper la justificación para un elemento en particular, agrupar varios elementos, incluir líneas divisorias, etc. En la práctica estas cuestiones surgen más en tablas en el modo de texto que en fórmulas del modo matemático por eso las posponemos al entorno `tabular` que es el hermano de `array` para texto. Casi todo lo que veremos a continuación para `tabular` se puede usar también con `array` aunque es menos probable que lo necesitemos.

## 2. Tablas de texto

Las tablas de texto en LaTeX se crean primordialmente con el entorno `tabular`. También hay varios paquetes para tal fin y constituyen una opción a considerar por pura comodidad o para necesidades especiales (aquí nos limitaremos a mencionar su nombre más adelante). Lo básico del entorno `tabular` es similar a lo mencionado sobre `array`, pero casi siempre en texto necesitamos decorar más el resultado. Pongamos por ejemplo que queremos hacer una tabla con fechas de viajes el origen y el destino y con este fin escribimos

```
\begin{center}
```

```
 \begin{tabular}{ccc}
  \textbf{Fecha}&\textbf{Origen}&\textbf{Destino}
  \\
  31/01/21&Madrid& Barcelona
  \\
  12/06/21&Helsinki& Pekín
  \\
  18/10/21&Camberra& Manila
 \end{tabular}
\end{center}
```

El resultado es el que cabe esperar:

| **Fecha** | **Origen** | **Destino** |
|---|---|---|
| 31/01/21 | Madrid | Barcelona |
| 12/06/21 | Helsinki | Pekín |
| 18/10/21 | Camberra | Manila |

Casi todos juzgaríamos que una tabla de este estilo, ya sea en un libro o en una página *web*, es un poco sosa. Lo primero es que en las tablas que estamos acostumbrados a ver, los elementos no están flotando, hay líneas de separación para favorecer la legibilidad. Las líneas verticales que separan las columnas se indican al mismo tiempo que la justificación en la primera línea del entorno. Así `{c|cc}` significa que queremos una línea vertical entre la primera y la segunda columna y `{|c|c|c|}` que las queremos entre todas y al principio y al final. Las líneas horizontales se indican con `\hline` justo antes de empezar la fila correspondiente. Por ejemplo, al cambiar el entorno `tabular` anterior por

```
 \begin{tabular}{|c|c|c|}
  \textbf{Fecha}&\textbf{Origen}&\textbf{Destino}
  \\ \hline
  31/01/21&Madrid& Barcelona
  \\ \hline
  12/06/21&Helsinki& Pekín
  \\ \hline
  18/10/21&Camberra& Manila
  \\ \hline
 \end{tabular}
```

se obtiene:

| **Fecha** | **Origen** | **Destino** |
|---|---|---|
| 31/01/21 | Madrid | Barcelona |
| 12/06/21 | Helsinki | Pekín |
| 18/10/21 | Camberra | Manila |

Nótese que la última línea horizontal no tiene fila que le siga, `\hline` precede a una línea vacía. Si quisiéramos una línea horizontal arriba habría que poner `\hline` justo después del comienzo del entorno. También se admiten líneas duplicadas tanto verticales como horizontales. Se usan más las horizontales para separar la cabecera de la tabla.

Como ejemplo, si en la fuente anterior cambiamos las primeras líneas por

```
\begin{tabular}{||c|c|c|}
 \hline
 \textbf{Fecha}&\textbf{Origen}&\textbf{Destino}
 \\ \hline\hline
```

el resultado será:

| **Fecha** | **Origen** | **Destino** |
|---|---|---|
| 31/01/21 | Madrid | Barcelona |
| 12/06/21 | Helsinki | Pekín |
| 18/10/21 | Camberra | Manila |

Una variante de `\hline` es `\cline{...}` que tiene como argumento dos números de columna separados por un guión. Lo que indica es que la línea horizontal se trazará entre ellas. Por ejemplo, si quisiéramos quitar la línea horizontal entre las dos primeras fechas porque ambas son de temporada alta, lo podríamos conseguir reemplazando en las líneas

```
31/01/21&Madrid& Barcelona
\\ \hline
```

el `\hline` por `\cline{2-3}`. Si por el contrario quisiéramos eliminar la línea entre Madrid y Helsinki porque ambos son orígenes que están en Europa, pondríamos `\cline{1-1} \cline{3-3}`, que traza una línea horizontal en el primer y tercer lugar, dejando al segundo sin ella. Sobre la tabla original con líneas el efecto sería:

| **Fecha** | **Origen** | **Destino** |
|---|---|---|
| 31/01/21 | Madrid | Barcelona |
| 12/06/21 | Helsinki | Pekín |
| 18/10/21 | Camberra | Manila |

| **Fecha** | **Origen** | **Destino** |
|---|---|---|
| 31/01/21 | Madrid | Barcelona |
| 12/06/21 | Helsinki | Pekín |
| 18/10/21 | Camberra | Manila |

A las justificaciones `l`, `c` o `r` de `array` se añade en el entorno `tabular` otra[1] que se indica mediante `p{...}` donde los puntos suspensivos representan cierta longitud, siempre acompañada de sus unidades. Esto permite incluir párrafos del ancho deseado en una tabla.

Por ejemplo, para la tabla:

[1]Estrictamente también funciona en `array`, pero tiene una utilidad menor porque considera que los elementos correspondientes están en modo texto.

| **Tiempo** | **Propósito** |
|---|---|
| Presente | Lo voy a pasar muy bien y voy a sacar las mejores notas posibles. |
| Futuro | Con mis notazas las mejores empresas se pelearán por mí. |

se ha usado un código del tipo:

```
\begin{center}
\begin{tabular}{|c|p{6.5cm}|}
      ...
\end{tabular}
\end{center}
```

Seguro que a más de uno no le gusta demasiado que "Presente" y "Futuro" aparezcan en la parte superior de la celda y querrían tener control sobre ello. Esto requiere paquetes especiales o hacer ajustes a ojo con `\raisebox`.

Cualquiera que haya creado una tabla en HTML sabe que a menudo es necesario pegar celdas, sobre todo en horizontal. Esto último se consigue en LaTeX con `\multicolumn{...}{...}{...}`. El primer argumento indica el número de celdas que pegamos; el segundo, la justificación que deseamos y el tercero, el contenido de las celdas. Por ejemplo, si queremos unir las celdas de "Origen" y "Destino" en una llamada "Itinerario" reemplazaríamos la línea

```
\textbf{Fecha}&\textbf{Origen}&\textbf{Destino}
```

por

```
\textbf{Fecha}&\multicolumn{2}{c|}{\textbf{Itinerario}}
```

Si pusiéramos `c` en vez de `c|` o `|c|` entonces no se cerraría la celda por la derecha. Aunque en principio suene extraño, usar `\multicolumn` con una sola celda tiene sentido porque nos da control sobre la justificación y sobre la posible barra a la derecha. Por ejemplo, si quisiéramos que "Pekín" apareciera justificado a la derecha bastaría cambiar `Pekín` por

```
\multicolumn{1}{r|}{Pekín}
```

El resultado de estos dos cambios combinados es:

| **Fecha** | **Itinerario** | |
|---|---|---|
| 31/01/21 | Madrid | Barcelona |
| 12/06/21 | Helsinki | Pekín |
| 18/10/21 | Camberra | Manila |

El pegado de celdas en vertical requiere cargar el paquete `multirow` mediante la línea en la cabecera:

```
\usepackage{multirow}
```

Este paquete define un comando `\multirow{...}{...}{...}` cuyos primer y tercer argumentos tienen el mismo significado que en `\multicolumn` mientras que el segundo indica el ancho de la columna. Dicho sea de paso, si alguna vez tienes curiosidad o te enfrentas a una tabla imposible, algunos de los paquetes más famosos que definen nuevos comandos y entornos para controlar las tablas son `array`, `tabularx`, `dcolumn` y `booktabs`.

Las líneas verticales que separan las columnas representadas por `|` en la definición de la justificación se pueden cambiar por cualquier expresión. Basta sustituirlas por `@{...}` donde los puntos suspensivos representan el nuevo separador de columnas. La utilidad práctica es reducida, aunque es fácil imaginar situaciones en que nos ahorra teclear. Por ejemplo, digamos que en nuestra lista queremos que la ciudad origen y destino estén siempre separadas por una flecha, entonces abriríamos el entorno con

```
\begin{tabular}{|c|r@{$\,\to\,$}l|}
```

Se ha utilizado la flecha corta del modo matemático y los espacios pequeños `\,` dejan una separación adicional. La justificación natural de las dos columnas involucradas es `rl` para que el nombre de las ciudades aparezca pegado a la flecha. El resultado es:

| **Fecha** | **Itinerario** |
|---|---|
| 31/01/21 | Madrid $\to$ Barcelona |
| 12/06/21 | Helsinki $\to$ Pekín |
| 18/10/21 | Camberra $\to$ Manila |

Como "Itinerario" tiene su propia justificación, debida a `\multicolumn`, no se ve afectado por `@{...}`.

Una cosa un poco fastidiosa con el entorno `tabular`, sobre todo al incluir alguna fórmula matemática con un operador grande o incluir un tipo de letra mayor que el habitual, es que a veces no considera espaciamientos verticales adecuados. Por ejemplo, si en la tabla anterior hacemos la"B" de Barcelona mayor mediante `{\Large B}`, el resultado es:

| **Fecha** | **Itinerario** |
|---|---|
| 31/01/21 | Madrid $\to$ {\Large B}arcelona |
| 12/06/21 | Helsinki $\to$ Pekín |
| 18/10/21 | Camberra $\to$ Manila |

Una pequeña chapucilla para resolverlo si no queremos cargar paquetes especiales consiste en insertar una línea vertical suficientemente alta de ancho cero (para que sea invisible). El comando `\rule{...}{...}` de TEX genera una línea donde el primer argumento es el ancho y el segundo el alto. Escribiendo entonces `\rule{0pt}{12pt}{\Large B}` forzamos a que la celda

se vuelva más alta para que quepa algo de `12pt` por encima de la línea. El resultado es mucho más satisfactorio:

| **Fecha** | **Itinerario** |
|---|---|
| 31/01/21 | Madrid → Barcelona |
| 12/06/21 | Helsinki → Pekín |
| 18/10/21 | Camberra → Manila |

Una manera más drástica es redefinir el comando que indica la proporción de los espacios verticales extra. Por ejemplo,

```
\renewcommand{\arraystretch}{1.2}
```

multiplica todos los espacios verticales extra por 1,2 y si en la tabla anterior usamos el entorno con

```
\begin{center}
 \renewcommand{\arraystretch}{1.2}
 \begin{tabular}{|c|r@{$\,\to\,$}l|}
```

obtendremos:

| **Fecha** | **Itinerario** |
|---|---|
| 31/01/21 | Madrid → Barcelona |
| 12/06/21 | Helsinki → Pekín |
| 18/10/21 | Camberra → Manila |

Si `\renewcommand{\arraystretch}{1.2}` no estuviera dentro del entorno `center` o de cualquier bloque que separara la definición del resto, tendría un efecto global sobre las tablas siguientes.

Hay además dos longitudes que pueden resultar de utilidad en la presentación de una tabla. En LaTeX las longitudes se asignan por medio de

`\setlength{`nombre de la longitud`}{`valor`}`

La longitud `\arrayrulewidth` mide el grosor de las líneas empleadas en la tabla y `\tabcolsep` la separación básica entre columnas, asignándole el valor `0pt` conseguiríamos contenidos de celdas pegados a sus límites verticales. Como ejemplo, para obtener

| **Fecha** | **Itinerario** |
|---|---|
| 31/01/21 | Madrid → Barcelona |
| 12/06/21 | Helsinki → Pekín |
| 18/10/21 | Camberra → Manila |

se ha empleado un código de la forma

```
\begin{center}
 \setlength{\arrayrulewidth}{1pt}
 \setlength{\tabcolsep}{30pt}
 \begin{tabular}{|c|r@{$\,\to\,$}l|}
     ...
 \end{tabular}
\end{center}
```

El cambio de longitudes necesita estar también de un bloque si no queremos que tenga un efecto global sobre lo que viene después. Si no empleásemos el entorno `center` y quisiéramos que el resultado solo afectase a la tabla en curso, podríamos abrir una llave antes de la asignación de las longitudes y cerrarla tras `\end{tabular}`.

Los ejemplos que hemos visto muestran que una tabla medianamente compleja requieren un código algo elaborado y repetitivo, posiblemente con muchos `\multicolumn` en las celdas, lo que motiva la existencia de algunos asistentes para llevar a cabo gráficamente la elaboración de tablas, a menudo incluidos en los editores. Una aplicación *web* a tener en cuenta es el generador de tablas *online* de `https://www.tablesgenerator.com/`. Puede resultar un poco pesado escribir una tabla grande allí, pero nos sacará de más de un apuro si la usamos para recordar cómo se hace algo construyendo algún ejemplo mínimo.

## 3. Listas

A pesar de que desde el punto de vista del código LaTeX las tablas y las listas no tienen absolutamente nada que ver, es cierto que visualmente las últimas recuerdan muchas veces a las primeras y esto motiva ubicarlas aquí.

Los dos tipos de listas más empleados en LaTeX corresponden a los entornos `itemize` y `enumerate` con la diferencia sugerida por el nombre: el primero no numera y el segundo sí. En ambos cada ítem debe ir precedido por el comando `\item`.

Por ejemplo,

- Desayunar
- Ir a clase
- Aburrirme

se obtiene con

```
\begin{itemize}
 \item Desayunar
 \item Ir a clase
 \item Aburrirme
\end{itemize}
```

Mientras que si cambiamos `itemize` por `enumerate` obtendremos

1. Desayunar
2. Ir a clase
3. Aburrirme

El contenido de cada punto no está limitado a lo que quepa en una línea, se pueden incluir uno o varios párrafos que quedarán justificados tras el punto o el número que corresponda al entorno (también pueden incluir fórmulas centradas). Por ejemplo, con un par de párrafos en el primer punto obtendríamos:

- Desayunar, pero desayunar bien fuerte para tener mucha energía, no vale con un par de galletas y un vaso de leche.

  Al menos eso es lo que dice mucha gente aunque pocos lo hagan.

- Ir a clase
- Aburrirme

Antes de seguir, no sé si es cosa mía, pero siempre me parece que estos entornos dejan demasiado espacio entre un ítem y el siguiente. Hay una longitud llamada `\itemsep` que indica cuánto difiere esta separación de cierta cantidad. Cambiando esta longitud justo a continuación de abrir el entorno se consigue que su ámbito sea local[2]. La separación entre el punto o número y el ítem sí me parece adecuada, pero por si alguien quiere modificarla, se llama `\labelsep`. En ejemplo con `enumerate` escribiendo tras `\begin{enumerate}`

```
 \setlength{\itemsep}{-5pt}\setlength{\labelsep}{20pt}
```

se obtiene:

1. Desayunar
2. Ir a clase
3. Aburrirme

[2]Por si te lo estás preguntando, en `tabular` esto no es posible por alguna razón y de ahí lo de aislar las longitudes que hemos visto antes en un bloque.

Si necesitas cosas más sofisticadas sobre márgenes y centrados de listas, el paquete `enumitem` ofrece soluciones a muchos problemas.

Si por alguna razón queremos que no se respete el punto o el número que debiera aparecer, basta indicar la expresión que preferimos entre corchetes como argumento de `\item`. En el caso de `enumerate` cada ítem especial será saltado en la cuenta. Por ejemplo,

```
\begin{enumerate}\setlength{\itemsep}{-3pt}
 \item Desayunar
 \item[*.] Ir a clase
 \item Aburrirme
\end{enumerate}
```

daría lugar a

1. Desayunar

*. Ir a clase

2. Aburrirme

Emplear una expresión de unos cuantos caracteres es lícito y puede tener cierto sentido sobre todo en `itemize`, pero, si no usamos el paquete antes citado, correremos el riesgo de que haya texto que se escape por el margen. En ese caso quizá el entorno `description` (mucho menos usado) nos parezca más adecuado que `itemize`. Para entender la diferencia, consideremos

```
\begin{itemize}\setlength{\itemsep}{-5pt}
 \item[\bf Uno.] Primer punto.
 \item[\bf Dos.] Segundo punto.
 \item[\bf No lo olvides.] Ojo al margen.
\end{itemize}
```

que genera

**Uno.** Primer punto.
**Dos.** Segundo punto.
**No lo olvides.** Ojo al margen.

Cambiando `itemize` por `description` pasaría a ser:

**Uno.** Primer punto.
**Dos.** Segundo punto.
**No lo olvides.** Ojo al margen.

En el resto de la sección analizaremos las listas múltiples y una suerte de cambio global de los argumentos de `\item`. Antes de ello, hay que notar que aunque el paquete `babel` estrictamente no es incompatible con lo que vamos a ver, sí provoca rarezas a evitar en ciertos idiomas. Con este fin,

es conveniente desactivar las decisiones que toma sobre las listas en español incluyendo el parámetro `es-nolists` al cargar `babel`. En el caso de este documento, la cabecera incluye

```
\usepackage[spanish, es-nolists]{babel}
```

Esta opción de `babel` unifica los resultados. Sin entrar en detalles, una de las rarezas que se evita con ello es que los símbolos que dan entrada a los diferentes ítem no dependan del idioma.

Cuando en una lista uno de los ítem es a su vez una lista o la contiene, LaTeX juiciosamente decide utilizar símbolos o numeraciones independientes. Por ejemplo, al compilar el siguiente código[3]

```
\begin{itemize}\setlength{\itemsep}{-3pt}
 \item Voy a la tienda
 \item Cuando llego
      \begin{itemize}\setlength{\itemsep}{-1pt}
       \item Busco la sección de helados
       \item Compro legumbres
      \end{itemize}
 \item Vuelvo a casa
\end{itemize}
```

se obtiene

- Voy a la tienda
- Cuando llego
  - Busco la sección de helados
  - Compro legumbres
- Vuelvo a casa

y para `enumerate` el código análogo produciría

1. Voy a la tienda
2. Cuando llego
   (a) Busco la sección de helados
   (b) Compro legumbres
3. Vuelvo a casa

[3]La separación del margen del segundo bloque `itemize` en la fuente es solo para favorecer la legibilidad y la localización de posibles errores, esos espacios son ignorados. Se podría escribir el entorno `itemize` justo a continuación de `llego`. De la misma forma, en la fuente en una misma línea puede haber varios `item`.

Ya sea en una lista normal o en una lista de lista uno puede no estar de acuerdo con el símbolo usado por LaTeX. Ya hemos visto cómo modificarlo puntualmente con un argumento de `\item`, pero hay algo más potente: Los comandos `\labelitemi`, `\labelitemii`, etc. (hasta cuatro niveles) guardan el tipo de símbolo que introduce `itemize` y lo podemos cambiar a voluntad con

```
\renewcommand{\labelitemi}{...}, etc.
```

Si esta redefinición del comando la hacemos después de abrir el entorno solo tendrá efecto para la lista en curso y sus sublistas. En el último ejemplo de `itemize` cuando utilizamos como primeras líneas

```
\begin{itemize}\setlength{\itemsep}{-3pt}
 \renewcommand{\labelitemi}{$\looparrowright$}
 \renewcommand{\labelitemii}{$\to$}
```

el resultado es:

- ↫ Voy a la tienda
- ↫ Cuando llego
  - → Busco la sección de helados
  - → Compro legumbres
- ↫ Vuelvo a casa

El análogo de `\labelitemi`, etc. para `enumerate` es `\labelenumi`, etc. En este caso lo que podemos indicar son cinco formas diferentes de numeración:

| | |
|---|---|
| `\arabic` | Números |
| `\alph` | Letras minúsculas |
| `\Alph` | Letras mayúsculas |
| `\roman` | Números romanos en minúscula |
| `\Roman` | Números romanos en mayúscula |

La forma de numeración escogida se debe aplicar a un contador[4] que según el nivel de la lista se llama `enumi`, `enumii`, etc.

Lo mejor para indicar cómo se hace es dar un ejemplo. Si la lista inicial la quisiéramos numerar con números romanos en mayúscula seguidos de una flecha utilizaríamos

```
\begin{enumerate}\setlength{\itemsep}{-3pt}
 \renewcommand{\labelenumi}{\Roman{enumi}$\to$}
 \item Desayunar
 \item Ir a clase
 \item Aburrirme
\end{enumerate}
```

[4]Un contador es una variable que almacena un número entero.

El resultado es:

I→ Desayunar
II→ Ir a clase
III→ Aburrirme

Los contadores en LaTeX se asignan por medio de `\setcounter`, lo cual permite comenzar la cuenta donde queramos. Por ejemplo, si en una lista anterior hemos terminado en IV, quizá queramos comenzar la nueva en V. Para ello, usaríamos en la primera línea

```
\begin{enumerate}\setcounter{enumi}{4}
```

obteniendo

V→ Desayunar
VI→ Ir a clase
VII→ Aburrirme

Si empleamos `\setcounter` antes de un `\item` modificaremos la cuenta a nuestra voluntad, por ejemplo, con `\setcounter{enumi}{20}` antes del segundo `\item` el resultado es:

V→ Desayunar
XXI→ Ir a clase
XXII→ Aburrirme

Por supuesto, es bastante dudoso que en la práctica deseemos incluir estos saltos de numeración.

# Enunciados y programación

**Lección 8**

## 1. Definición y numeración de enunciados

El paquete más usado para dar formato a los enunciados matemáticos y sus pruebas es `amsthm`. Por consiguiente, comenzamos incluyendo en la cabecera

```
\usepackage{amsthm}
```

Según la documentación hay que hacerlo después de que esté cargado el paquete `amsmath`.

Es importante que sepas que utilizando el `\documentclass` con el estilo de una editorial o quizá la plantilla de tu TFG, este paquete podría estar cargado de antemano y los comandos indicados en esta sección podrían tener interacciones indeseadas.

El uso de `amsthm` es muy sencillo. En la cabecera, después de haberlo cargado y antes del `\begin{document}`, escribiremos `\newtheorem` con dos argumentos entre llaves, el primero indica el nombre del entorno que queremos usar y el segundo el nombre que aparecerá en el documento. Por ejemplo, para un artículo en español podemos emplear:

```
\newtheorem{theorem}{Teorema}
```

que crea un entorno llamado `theorem` que mostrará la palabra "Teorema" cuando se invoque a él. Es decir, con

```
\begin{theorem}
 Los poliedros convexos cumplen $V+C=A+2$.
\end{theorem}
```

conseguiremos:

**Teorema 1.** *Los poliedros convexos cumplen* $V + C = A + 2$.

El nombre del entorno es convencional, si queremos ser originales

```
\newtheorem{un_teorema}{Teorema}
```

tendría el mismo efecto siempre que abramos con `\begin{un_teorema}` los enunciados de los teoremas y los cerremos con `\end{un_teorema}`.

La numeración es automática y consecutiva, así

```
\begin{theorem}
 Si fuera un mapa en un toro, $V+C=A$.
\end{theorem}
```

genera

**Teorema 2.** *Si fuera un mapa en un toro,* $V + C = A$.

Para un documento largo la numeración anterior es demasiado simple, y, si el documento es muy largo, poco útil. Las razones son las mismas que en la numeración de ecuaciones. Por ejemplo, si uno tuviera que buscar el Teorema 503 en un volumen gigantesco seguramente estaría un buen rato pasando páginas. Por ello, en un libro lo más habitual es que el número de un enunciado incorpore alguna información acerca del lugar donde se ubica. Por ejemplo, los teoremas anteriores podrían numerarse con **1.1** y **1.2** para indicar que están en la primera sección. Recordemos que las secciones en LaTeX se indican mediante `\section{`*título*`}`. Por ejemplo, en este documento la primera sección se creó con

```
\section{Definición y numeración de enunciados}
```

y también existen, en la clase `article`, `\subsection` y `\subsubsection` que actúan de la misma forma para crear subsecciones y lo que corresponde al palabro "subsubsecciones". Dicho sea de paso, el formato `book` que, como su nombre indica, se aplica a libros, incorpora todavía dos divisiones más amplias: `\chapter` y `\part`.

Para que los teoremas, o enunciados en general, incorporen el número de sección, basta añadir `[section]` al final de la definición del enunciado en la cabecera. Esto es:

```
\newtheorem{theorem}{Theorem}[section]
```

Si no hubiera secciones porque no hemos empleado el comando `\section` entonces la numeración sería **0.1** y **0.2** porque el contador de secciones está internamente a cero, lo cual quizá te parezca conveniente para una sección introductoria. Los nombres de otras divisiones se pueden emplear de manera análoga en la numeración de enunciados en lugar de `section` y cada una incluye las numeraciones de las divisiones que las contienen (en formato `book` hasta `chapter`). Con ello, si en un documento como este empleamos

```
\newtheorem{theorem}{Teorema}[subsubsection]
```

antes del número del teorema aparecerá el número de sección, subsección y subsubsección (por ese orden). De esta forma un teorema podría llamarse **Teorema 2.3.5.7** y en formato `book` se transformaría en **Teorema 1.2.3.5.7** si está en el capítulo 1. Por supuesto, rara vez se da tanto detalle sobre la ubicación de un teorema y en formato `article` el argumento `[section]` es más común que cualquier otro.

Con `\newtheorem` podemos definir tantos entornos como deseemos para lemas, corolarios, observaciones o enunciados de nuestra propia cosecha. Si lo hacemos como antes, por ejemplo

```
\newtheorem{lemma}{Lema}
```

o

```
\newtheorem{lemma}{Lema}[section]
```

la numeración será independiente, es decir, habrá Lema 1 (o 1.1) y también Teorema 1 (o 1.1). Algunos textos de matemáticas (una minoría) siguen esta política. Personalmente me parece poco adecuada porque los resultados son difíciles de localizar: cerca del Teorema 20 puede estar el Lema 42. La manera de impedirlo es forzar a que la numeración de los lemas (y otros enunciados) continúen la misma numeración especificada en los teoremas. Para indicar que un enunciado depende de un entorno anteriormente definido, digamos `theorem` en nuestro caso, la estructura es

`\newtheorem{`nombre del entorno`}[theorem]{`nombre en el texto`}`

Con ello el nuevo entorno continuará la numeración de `theorem` y de cualquier entorno ligado a él de la misma forma. Dicho esto, es común ver en la cabecera de un documento LaTeX en español cosas como:

```
\newtheorem{proposition}[theorem]{Proposición}
\newtheorem{lemma}[theorem]{Lema}
\newtheorem{corollary}[theorem]{Corolario}
```

Siempre habiendo definido antes `theorem`. Por ejemplo, con esto el código

```
\begin{theorem}
 Si $a$ y $n$ son coprimos, $n$ divide a $a^{\varphi(n)}-1$.
\end{theorem}
\begin{corollary}
 Si $p$ es primo, $a^p-a$ es siempre múltiplo de $p$.
\end{corollary}
```

daría la numeración consecutiva deseada:

**Teorema 3.** *Si $a$ y $n$ son coprimos, $n$ divide a $a^{\varphi(n)} - 1$.*

**Corolario 4.** *Si $p$ es primo, $a^p - a$ es siempre múltiplo de $p$.*

Hay una variante que tiene algún sentido para observaciones o corolarios, pero que, en mi experiencia, no es demasiado usada en la práctica. Si escribimos `[theorem]` al final, en vez de en medio, entonces la numeración incluirá la del último enunciado de tipo `theorem`. Por ejemplo, si con las líneas anteriores en la cabecera incluimos allí también

```
\newtheorem{remark}{Observación}[theorem]
```

Entonces, salvo el tipo de letra, las líneas

```
\begin{proposition}
 Para $a,b\in\mathbb{R}$ se tiene $ab=ba$.
\end{proposition}

\begin{remark}
 También se cumple para números complejos.
\end{remark}

\begin{remark}
 No se cumple en general para matrices.
\end{remark}
```

producirán:

**Proposición 5.** *Para $a, b \in \mathbb{R}$ se tiene $ab = ba$.*

*Observación* 5.1. También se cumple para números complejos.

*Observación* 5.2. No se cumple en general para matrices.

Hay coherencia en la sintaxis del comando con lo visto anteriormente: cuando poníamos `[section]` se incorporaba al enunciado el número de sección y ahora que ponemos `[theorem]` se incorpora el de teorema, a pesar de que las secciones y teoremas son para nosotros, e internamente para LaTeX, objetos bien distintos.

Seguramente te hayas percatado de que el ejemplo anterior es un poco tramposo porque el tipo de letra de las observaciones es diferente del de la proposición, tanto en el nombre como en el contenido, cuando el comportamiento por defecto es que todos los enunciados tengan el mismo formato. Hay un mínimo control sobre ello empleando:

`\theoremstyle{`*estilo*`}`

donde el *estilo* es `plain` (por defecto), `remark` o `definition`. Es posible crear nuevos estilos, pero no lo veremos aquí[1]. Al incluir un `\theoremstyle` en la cabecera, los `\newtheorem` definidos a continuación hasta un nuevo `\theoremstyle` se verán afectados por dicho estilo. Así en la cabecera de este documento aparece:

```
\theoremstyle{remark}
\newtheorem{remark}{Observación}[theorem]
```

Como muestran los ejemplos anteriores, `plain` pone el nombre y número en negrita y el texto en cursiva, mientras que `remark` pone el nombre y número en cursiva y el texto en letra normal. Por otro lado, `definition` pone el nombre y número en negrita y el texto en letra normal.

[1] Un sustituto más poderoso que veremos más adelante es la definición de un nuevo entorno no asociado a `amsthm`.

## 2. Referencias y pruebas

Siendo las matemáticas deductivas, continuamente en nuestros documentos habrá referencias a enunciados anteriores. Las etiquetas de teoremas y otros enunciados se definen con `label`, como las de las ecuaciones. Por ejemplo, si al enunciar nuestra proposición escribimos

```
\begin{proposition}\label{conmut}
 Para $a,b\in\mathbb{R}$ se tiene $ab=ba$.
\end{proposition}
```

entonces con

```
 Por la Proposición~\ref{conmut}
```

se obtiene "Por la Proposición 5".

El entorno para las demostraciones es `proof`. Por ejemplo, la prueba del primer teorema:

*Demostración.* Se sigue de la conmutativa porque $\mathbb{R}$ es un cuerpo. □

viene de la fuente

```
\begin{proof}
 Se sigue de la conmutativa porque $\mathbb{R}$ es un cuerpo.
\end{proof}
```

El nombre "*Demostración*" depende de `babel` y si cambiamos de idioma se modificará consecuentemente. De esta forma, si cambiamos `spanish` por `english` cuando se carga `babel` en la cabecera de la fuente de este documento se obtiene "*Proof*" y si lo cambiamos por `german` se obtiene "*Beweis*".

A veces, la prueba está lejos del teorema y queremos hacer referencia a él. La manera habitual de hacerlo es emplear el entorno `proof` con un parámetro en la forma `{proof}[...]` que sustituye "Demostración" por lo que escribamos entre los corchetes. Por ejemplo, con

```
\begin{proof}[Demostración de la Proposición~\ref{conmut}]
 Se sigue de la conmutativa porque $\mathbb{R}$ es un cuerpo.
\end{proof}
```

obtenemos

*Demostración de la Proposición 5.* Se sigue de la conmutativa porque $\mathbb{R}$ es un cuerpo. □

No hay restricciones sobre el texto que sustituye a "Demostración" y podemos usar `\ref` para referirnos a partes de un texto, como `\section` o `\subsection`, que hayamos etiquetado con `\label`. Por ejemplo, si al introducir esta sección hemos escrito

```
\section{Referencias y pruebas}\label{re_pr}
```

entonces con

```
\begin{proof}[Prueba del resultado de la sección \ref{re_pr}]
 Se sigue de la conmutativa porque $\mathbb{R}$ es un cuerpo.
\end{proof}
```

obtenemos

*Prueba del resultado de la sección 2.* Se sigue de la conmutativa porque $\mathbb{R}$ es un cuerpo. □

## 3. Definición de comandos

Ya habíamos visto algún ejemplo simple de definición de comandos. La estructura básica es:

`\newcommand{`*nombre*`}{`*significado*`}`

Así con `\newcommand{\R}{\mathbb{R}}` conseguíamos abreviar la expresión `\mathbb{R}` por `\R` lo cual será muy conveniente si estamos escribiendo un texto de análisis real. Lo ortodoxo y muy recomendable es poner las definiciones de los comandos en la cabecera para favorecer la legibilidad de la fuente y darles ámbito global, pero no es estrictamente obligatorio.

Alguna vez querremos redefinir un comando presente en LaTeX porque la abreviatura que hemos elegido coincide con un comando que no usamos o dicho comando se emplea internamente con algún propósito y queremos que personalizarlo. Un ejemplo de esto último es `\qedsymbol` que es un comando, normalmente de uso interno, que da el símbolo que indica el final de una demostración. Por defecto está definido como un cuadrado sin rellenar. Para redefinir este u otro comando se utiliza `\renewcommand`. Por ejemplo si quisiéramos ser originales y que en vez del cuadrado apareciera un círculo, emplearíamos:

```
\renewcommand{\qedsymbol}{$\bigcirc$}
```

Para que el cuadrado se rellene una posibilidad es

```
\renewcommand{\qedsymbol}{$\blacksquare$}
```

Dicho sea de paso, como `\rule{...}{...}` da una línea de cierto ancho y alto sustituyendo los puntos suspensivos por longitudes iguales tendremos control sobre el tamaño del cuadrado relleno. Así, sustituyendo `$\blacksquare$` por `\rule{4mm}{3mm}` obtendremos un rectángulo apaisado de $4 \times 3$ milímetros.

Incidiendo sobre lo dicho antes, un `\renewcommand{\qedsymbol}` que esté dentro de un entorno `proof` solo actuará sobre el símbolo de la demostración correspondiente, dejando el resto sin cambios.

Este uso de `\newcommand` y `\renewcommand` es equivalente a lo que conseguiríamos con un editor básico reemplazando cadenas de caracteres. Algo más complicado (aunque no fuera del alcance de editores avanzados) es definir o redefinir comandos con argumentos. La estructura es ahora

`\newcommand{`*nombre*`}[`*$n^o$. argumentos*`]{`*significado*`}`

Los argumentos se indican en el *significado* con `#1`, `#2`, etc.

Por ejemplo, supongamos que voy a escribir muchos ejemplos de integrales racionales entre cero y uno para los alumnos de Cálculo I y para ello me gustaría definir un comando en el que bastase introducir numerador y denominador. Una posibilidad es

```
\newcommand{\intr}[2]{\int_0^1\frac{#1}{#2}\;dx}
```

Con ello,

```
\[
 \intr{x+1}{x^2+2},\qquad \intr{x}{x-2},\qquad
 \intr{1}{x+2}.
\]
```

da lugar a:

$$\int_0^1 \frac{x+1}{x^2+2}\,dx, \qquad \int_0^1 \frac{x}{x-2}\,dx, \qquad \int_0^1 \frac{1}{x+2}\,dx.$$

Hay todavía otra forma más general de `\newcommand` (y `\renewcommand`) que responde a

`\newcommand{`*nombre*`}[`*$n^o$. arg.*`][`*opcional*`]{`*significado*`}`

Con ello `#1` tomará el valor de *opcional* a no ser que se especifique otra cosa como un parámetro entre corchetes. Volviendo al ejemplo anterior, imaginemos que casi todas las integrales que quiero escribir tienen numerador $x+1$, pero hay algunas excepciones. Definiríamos

```
\newcommand{\inte}[2][x+1]{\int_0^1\frac{#1}{#2}\;dx}
```

y con

```
\inte{x^2+2},\qquad \inte{x+7},\qquad \inte[x^2+1]{x+1}.
```

conseguiríamos:

$$\int_0^1 \frac{x+1}{x^2+2}\,dx, \qquad \int_0^1 \frac{x+1}{x+7}\,dx, \qquad \int_0^1 \frac{x^2+1}{x+1}\,dx.$$

En el último caso `[x^2+1]` es el argumento opcional que cambia el valor $x+1$ por defecto de `#1`.

# 4. Definición de entornos

Recordemos que un entorno es algo que está en un bloque

\begin{*nombre*}...\end{*nombre*}

Crear nuevos entornos es más complicado que crear nuevos comandos, pero la idea sobre su definición o redefinición es similar. La plantilla básica es:

\newenvironment{*nombre*}{*código inicial*}{*código final*}

y también hay un `\renewenvironment` con estructura análoga. Su efecto es que se copiará el código inicial, después lo que hayamos escrito dentro del entorno y después el código final. Por ejemplo, si me quiero inventar una nueva forma de matriz que en vez de paréntesis tenga dobles llaves, una posibilidad es definir un nuevo entorno `BBmatrix` de la siguiente forma:

```
\newenvironment{BBmatrix}
{\left\{\left\{\begin{matrix}}
{\end{matrix}\right\}\right\}}
```

Si escribimos por ejemplo (en modo matemáticas)

```
\begin{BBmatrix}
 2 & 1 \\ 3& 4
\end{BBmatrix}
```

el resultado será:

$$\left\{\left\{\begin{matrix} 2 & 1 \\ 3 & 4 \end{matrix}\right\}\right\}$$

Los entornos, igual que los comandos, admiten argumentos y argumentos opcionales. Los argumentos no opcionales, al menos en mi experiencia, no son tan comunes. Quizá una razón para ello es que, por razones técnicas, el uso de los argumentos está limitado al código inicial[2].

Para ilustrarlo, supongamos que tenemos que elaborar una lista de ejercicios y tareas que incorporan una indicación de la dificultad. Ensayemos diferentes versiones. Para practicar supongamos que queremos el texto del ejercicio o tarea en letra inclinada y el nombre en versalita. Una posibilidad muy simple es:

```
\newenvironment{ej_a}[2]{\textsc{#1}. (Dificultad: #2).\sl}{}
```

Al escribir:

[2]Hay maneras indirectas de evitar esta restricción; por ejemplo con el paquete `xparse` o definiendo en el código inicial un comando que guarde el argumento y que se use en el código final.

```
\begin{ej_a}{Ejercicio}{0}
 Calcula $1+1$.
\end{ej_a}

\begin{ej_a}{Tarea}{$\star\star\star$}
 Halla una fórmula para $1^2+2^2+\dots+n^2$.
\end{ej_a}
```

obtendremos:

Ejercicio. (Dificultad: 0). *Calcula* $1+1$.

Tarea. (Dificultad: $\star\star\star$). *Halla una fórmula para* $1^2+2^2+\cdots+n^2$.

Si vamos a escribir muchos ejercicios y ocasionalmente una tarea, convendría definir en su lugar:

```
 \newenvironment{ej_b}[2][Ejercicio]{\textsc{#1}.
 (Dificultad: #2).\sl}{}
```

y entonces se conseguiría el mismo resultado que antes ahora con

```
\begin{ej_b}{0}
 Calcula $1+1$.
\end{ej_b}

\begin{ej_b}[Tarea]{$\star\star$}
 Halla una fórmula para $1^2+2^2+\dots+n^2$.
\end{ej_b}
```

Antes de pasar a algo más complicado, vamos con un poco de ajuste fino del entorno. Si tecleamos

```
Resuelve la siguiente lista de ejercicios fáciles:
\begin{ej_b}{0}
 Calcula $1+1$.
\end{ej_b}

\begin{ej_b}{0}
 Calcula $2+2$.
\end{ej_b}
```

el resultado dista seguramente de lo que teníamos en mente:

Resuelve la siguiente lista de ejercicios fáciles: Ejercicio. (Dificultad: 0). *Calcula* $1+1$.

Ejercicio. (Dificultad: 0). *Calcula* $2+2$.

Y más todavía si no dejamos la línea en blanco entre los entornos. Lo habitual en las listas de ejercicios es que estén separados algo más que el interlineado habitual. Una alternativa es:

```
\newenvironment{ej_c}[2][Ejercicio]{\par\textsc{#1}.
(Dificultad: #2).\sl}{\medskip}
```

El efecto es que incluye un cambio de línea inicial (`\par`) y un salto extra al final. El resultado sería:

Resuelve la siguiente lista de ejercicios fáciles:

Ejercicio. (Dificultad: 0). *Calcula* $1 + 1$.

Ejercicio. (Dificultad: 0). *Calcula* $2 + 2$.

Con un `\medskip` tras de `\par` también separaríamos de la línea inicial.

Ahora vamos a una situación más realista en la que deseamos numerar automáticamente los ejercicios. Con esto, estamos cerca de lo que hacíamos con los enunciados usando `amsthm`, pero ahora tenemos más libertad permitiendo argumentos. Los enunciados que hemos visto antes son un caso particular de la definición de entornos.

Sin entrar en muchos detalles, aquí están los comandos más relevantes:

| **Plantilla** | **Efecto** |
|---|---|
| `\newcounter{`*nombre*`}` | Crea el contador *nombre* |
| `\the`*nombre* | Muestra el valor de *nombre* |
| `\refstepcounter{`*nombre*`}` | Incrementa *nombre* |
| `\setcounter{`*nombre*`}{`*valor*`}` | Asigna *valor* a *nombre* |

Los contadores por defecto tienen asignado inicialmente el valor cero y entonces el último comando solo se emplea si queremos romper la numeración natural.

En nuestro caso, creamos un contador `ejer` que se incremente al llamar al entorno y se muestre al lado del primer argumento. Para ello incluimos en la cabecera:

```
\newcounter{ejer}
\newenvironment{ej_d}[2][Ejercicio]{\refstepcounter{ejer}%
\par\textsc{#1 \theejer}. %
(Dificultad: #2).\sl}{\medskip}
```

Seguro que te estás preguntando a qué vienen los `%` al final de las líneas. No son necesarios, podríamos haber escrito todo en una línea como antes, pero si miras fuentes de usuarios avanzados verás esto muy a menudo. El propósito es asegurarse de que no se cuela ningún espacio de fin de línea que no vemos. En un editor si veo `\theejer}.` no estoy seguro de si hay un salto de línea justo tras el punto o hay un espacio (que es lo que quiero).

Con o sin `%` el resultado de usar `ej_d` como antes es:

Ejercicio 1. (Dificultad: 0). *Calcula* $1 + 1$.

Ejercicio 2. (Dificultad: 0). *Calcula* $2 + 2$.

Las divisiones de un documento, como `section` y `subsection` (también `page`), tienen sus propios contadores. Imaginemos que quisiéramos retocar el entorno anterior para que mostrase también el número de sección. Lo más lógico sería emplear:

```
\newenvironment{ej_e}[2][Ejercicio]{\refstepcounter{ejer}%
\par\textsc{#1 \thesection.\theejer}. %
(Dificultad: #2).\sl}{\medskip}
```

El resultado con ello sería:

Ejercicio 4.3. (Dificultad: 0). *Calcula* $1 + 1$.

Ejercicio 4.4. (Dificultad: 0). *Calcula* $2 + 2$.

El comando `\newcounter` admite un parámetro entre corchetes que es otro contador de manera que *nombre* se pone a cero cuando ese segundo contador varía. Esto que suena tan raro es para permitir cosas como `\newcounter{ejer}[section]` que en el ejemplo anterior haría que la cuenta se reiniciase cuando pasamos a otra sección. En otro caso, el primer ejercicio en la siguiente sección aparecería numerado con un segundo número diferente de 1.

Un apunte final es que si buscas en la documentación verás que hay varias maneras de incrementar un contador, la elegida aquí, `\refstepcounter` debe el "`ref`" de su nombre a que es posible utilizar `\label` en los entornos definidos para que `\ref` refleje su valor.

# Colores, imágenes y presentaciones

**Lección 9**

## 1. Colores

El paquete básico para utilizar colores en LaTeX se llama `color`, pero es muy habitual emplear en su lugar `xcolor` que añade algunas características más. Se carga de la forma habitual incluyendo en la cabecera

```
\usepackage{xcolor}
```

Este paquete añade el comando `\color{...}` que cambia al color especificado por los puntos suspensivos. Este cambio se aplicará a partir del comando y hasta que se termine un bloque. Si esto suena muy críptico, seguro que estos ejemplos lo aclaran:

```
{El cielo es \color{cyan} azul celeste} cuando no está nublado

El cielo es {\color{cyan} azul celeste} cuando no está nublado
```

producen ambos "El cielo es azul celeste cuando no está nublado". Por supuesto, lo más normal es proceder de la segunda manera. Si no pusiéramos las llaves, todo el texto a partir de ese momento se volvería color cian. En realidad si haces una prueba con un texto muy extenso verás que se vuelve al color negro original cuando se cambia de página. Seguramente porque internamente hay bloques que incluyen cada página.

El uso de colores en fórmulas es similar, tanto en las que están en línea como en las centradas. Por ejemplo, con

```
El polinomio {\color{violet}cromático} de $\color{pink}K_3$ es
\[
 P(\triangleleft,t) =
 {\color{red}t}({\color{green}t-1})({\color{blue}t-2}).
\]
```

se consigue: El polinomio cromático de $K_3$ es

$$P(\triangleleft, t) = t(t-1)(t-2).$$

Una pregunta natural es de qué colores disponemos, la respuesta breve es de todos los que se nos ocurran. Ciertamente, tal afirmación no da ningún indicio acerca de los colores `cyan` y `violet` usados anteriormente. Con el paquete sin parámetros adicionales y sin definir nada por nuestra cuenta, hay 19 nombres que podemos usar correspondientes a los colores recogidos en esta tabla:

| | | | | |
|---|---|---|---|---|
| black | darkgray | lime | pink | violet |
| blue | gray | magenta | purple | |
| brown | green | olive | red | yellow |
| cyan | lightgray | orange | teal | |

Antes de que pienses que te he timado porque solo hay 18 colores, si miras la fuente observarás que entre violet y yellow está `{\color{white}white}`, lo que pasa es que el blanco sobre blanco no se ve. Otra observación es que si compilas con LaTeX, para producir un DVI, en lugar de con PDFLaTeX, para obtener un PDF, tu visor DVI hará de las suyas y no representará los colores de manera totalmente fidedigna. Incluso con PDFLaTeX, por muy buenas que sean tu pantalla y tu impresora, tampoco debes esperar una concordancia entre ambas. En la impresión profesional hay todo un mundo alrededor de la determinación de los colores y la calibración de los dispositivos.

Si cargas el paquete con

```
\usepackage[usenames,dvipsnames]{xcolor}
```

tendrás además los 68 colores `dvips` que tienen a veces nombres tan poéticos como `CarnationPink` o `WildStrawberry`. Una lista completa la puedes encontrar en la documentación del paquete `xcolor`. Allí se indican también otras opciones que permiten tener nombres de más colores.

En la práctica casi nadie sabe de memoria grandes listas de nombres de colores ni siquiera las consulta en la documentación a la hora de componer textos en LaTeX. Lo más habitual es crearlos uno mismo. Hay varias maneras de hacerlo. Aquí solo veremos dos. La primera, es combinar dos colores ya definidos (por nosotros o por defecto) siguiendo la estructura:

`\color{`*color1*`!`*porcentaje*`!`*color2*`}`

donde el porcentaje corresponde al *color1* y el resto será de *color2*. Considerando `\color{blue!50!yellow}` pasaremos al rango de mito eso que nos decían de pequeños de que azul y amarillo da verde, al menos con el significado informático de estos colores, que es diferente del de los lápices que nos daban para dibujar[1]. La moraleja es que combinar colores puede dar resultados inesperados.

[1] Son los modelos aditivo y sustractivo para mezclar colores. Están relacionados con que en las impresoras los cartuchos sean de colores cian, magenta, amarillo y negro (modelo CMYK) y en los monitores los píxeles sean rojos, verdes y azules (modelo RGB).

Un ejemplo más natural es usar `\color{cyan!60!black}` para oscurecer el color azul celeste combinando un 60 % de cian y el resto, un 40 %, de negro. Veamos el efecto de varios porcentajes poniendo el texto en negrita para que se aprecie mejor el color, es decir, usaremos como prueba

```
El cielo es \textbf{\color{cyan!#!black} azul celeste}
```

donde `#` representa cierto número entre 0 y 100. Algunos resultados son:

| | | |
|---|---|---|
| 20 % | $\longrightarrow$ | El cielo es **azul celeste** |
| 40 % | $\longrightarrow$ | El cielo es **azul celeste** |
| 60 % | $\longrightarrow$ | El cielo es **azul celeste** |
| 80 % | $\longrightarrow$ | El cielo es **azul celeste** |

Otra forma de disponer de un nuevo color es definirlo en la cabecera. Hay varias posibilidades. Aquí nos centraremos en

`\definecolor{`*nombre*`}{rgb}{`$R, G, B$`}`

donde *nombre* es la denominación convencional que queremos dar al color y $R, G, B \in [0, 1]$ indican la cantidad de color en los canales de color primarios rojo (**R**ed), verde (**G**reen) y azul (**B**lue). Seguro que en tu programa de retoque fotográfico favorito puedes elegir en un santiamén un color que te guste y obtener sus coordenadas RGB. Normalmente en informática se dan estas coordenadas como enteros entre 0 y 255 por tanto es muy posible que tengas que dividir el resultado ofrecido por tu *software* entre 255.

Por ejemplo, incluyendo en la cabecera

```
\definecolor{MiAzulOscuro}{rgb}{0,0.08,0.5}
```

tendremos un nuevo color llamado `MiAzulOscuro` que no tiene nada de rojo, una pizca de verde y un azul medio (1 sería el azul con más brillo). Escribiendo

```
El cielo es {\bf\color{MiAzulOscuro} azul celeste}
```

se obtiene: "El cielo es **azul celeste**".

A pesar de la precisión que virtualmente produce este comando, de nuevo, ten en cuenta que los dispositivos pueden establecer diferencias muy apreciables. Por ejemplo, en el texto de las páginas *web* solo se reproducen un subconjunto de los posibles colores RGB. De hecho, hay una variante de `\definecolor` que tiene un `HTML` en lugar de un `rgb` para trabajar con ellos. Su sintaxis es diferente y no la veremos aquí.

## 2. Imágenes

Con ayuda de paquetes auxiliares en LaTeX se pueden incluir imágenes e incluso crearlas (por ejemplo gráficas de funciones o esquemas). Nos centraremos en lo primero porque a efectos prácticos, hay mucho *software* para crear imágenes y por muy fanáticos que seamos de LaTeX puede ser un dolor de cabeza crear con él diagramas medianamente sencillos si no tenemos unos conocimientos relativamente avanzados.

Hay dos paquetes principales para la inclusión de imágenes: `graphicx` y `graphics`. En principio, según la documentación, el primero extiende al segundo pero yo he notado algunos problemas con ficheros `.eps` sin el segundo en mi distribución, por tanto normalmente cargo ambos en la cabecera con

```
\usepackage{graphics}
\usepackage{graphicx}
```

La compilación con LaTeX da problemas con formatos diferentes de `.eps` porque no es capaz de detectar sus dimensiones[2], por ello en lo sucesivo daremos por hecho que la compilación se lleva a cabo con PDFLaTeX, lo que permite manejar imágenes en formatos más habituales como `.jpg` o `.png`. El comando básico es

`\includegraphics[`*parámetros*`]{`*fichero*`}`

El nombre del fichero incluirá el *path* cuando sea necesario. Esto es, escribiremos simplemente el nombre, como `imagen.jpg`, si el fichero está en el mismo directorio que el fichero fuente y cosas como `./imagenes/imagen.jpg`, si hay que bajar al subdirectorio `imagenes`, o `../imagen.jpg`, si está en el directorio superior, o un nombre larguísimo si está en un directorio recóndito[3]. Los parámetros no son estrictamente necesarios, si no los incluimos la imagen se muestra tal cual, con su tamaño original. Rara vez este será el que nosotros deseamos y por ello parámetros muy habituales son

`width=`$l$, `height=`$l$ y `scale=`$r$

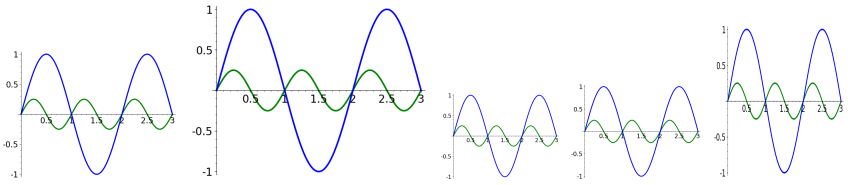
donde $l$ son longitudes y $r$ es un número. Cuando se especifican los dos primeros, se separan mediante comas. Por supuesto, `scale` es incompatible con fijar un ancho o un alto. Por ejemplo, usando la imagen de prueba `fsin.png` que muestra las gráficas de $\text{sen}(\pi x)$ y $\text{sen}(2\pi x)$ en $[0, 3]$ y tiene unas dimensiones originales de $628 \times 468$ en píxeles, con

[2]En realidad, en principio, esto se podía arreglar incluyendo entre los parámetros `natwidth=`$a$ y `natheight=`$h$ donde $a$ y $h$ son respectivamente la anchura y la altura de la imagen, pero por razones que desconozco ha dejado de funcionar en versiones modernas.

[3]Para los más alejados de los sistemas UNIX o Linux, quizá sea conveniente recordar que `./` significa el directorio en curso y `../` el directorio superior.

```
\includegraphics[height=50pt]{fsin.png}
\includegraphics[scale=0.2]{fsin.png}
\includegraphics[scale=0.1]{fsin.png}
\includegraphics[width=50pt]{fsin.png}
\includegraphics[width=50pt, height=60pt]{fsin.png}
```

obtenemos:



Hay que tener precaución al emplear simultáneamente `width` y `height` porque deformará la relación de aspecto. En una foto artística eso puede resultar muy feo.

Un comentario a tener en cuenta, es que se supone que lo ortodoxo (según muchos manuales) es usar `\includegraphics` dentro del entorno `figure`, pero yo rara vez lo hago. Este entorno facilita el poner títulos a las figuras y referencias a ellas y además automatiza su posición. Si no quieres ser tan cabezota como yo, deberías buscar información sobre `figure`. Para animarte a que lo hagas, comprueba el efecto de:

```
\begin{figure}[h]
\centering
\includegraphics[scale=0.2]{fsin.png}
\caption{Gráficas de seno}
\end{figure}
```

Incluyendo un `\label{mifigura}` el número de figura se mostraría con `\ref{mifigura}`. La `h` que aparece como parámetro del entorno indica *here*, es decir, que se olvide de la colocación automática y ponga la figura donde está el código, si es posible. Utilizando `figure` sin argumentos, la figura se reubicará, en general. Mi cerrazón a usar `figure` proviene de que incluso usando `[h]` o la variante más imperativa `[h!]`, a veces no he conseguido colocar la figura en el lugar deseado.

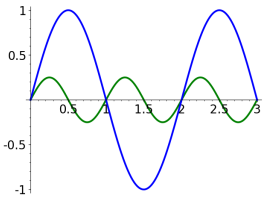
Un truco para lograr una buena alineación horizontal de varias imágenes es incluir un único parámetro `height` igual para todas, porque no aparecerán espacios sobrantes por arriba y por abajo aunque tengan diferentes dimensiones.

Por ejemplo, utilizando la imagen de prueba `fcos.png` que muestra las gráficas de $\cos(\pi x)$ y $\operatorname{sen}(2\pi x)$ en $[0, 2]$ y tiene unas dimensiones originales de $440 \times 468$ píxeles, la fuente

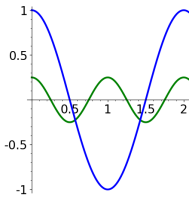
```
\begin{center}
  \begin{tabular}{c}
```

```
    \includegraphics[height=75pt]{fsin.png}
    \\
    \textsf{Gráficas de seno}
  \end{tabular} \qquad
    \begin{tabular}{c}
    \includegraphics[height=75pt]{fcos.png}
    \\
    \textsf{Gráficas de coseno}
  \end{tabular}
\end{center}
```

produce

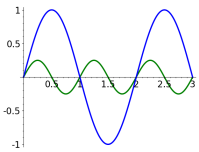



Gráficas de seno

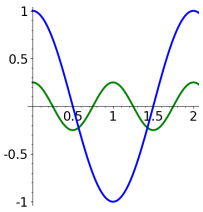



Gráficas de coseno

Si en vez de igualar las alturas lo hiciéramos con las anchuras cambiando `height` por `width`. El resultado sería:

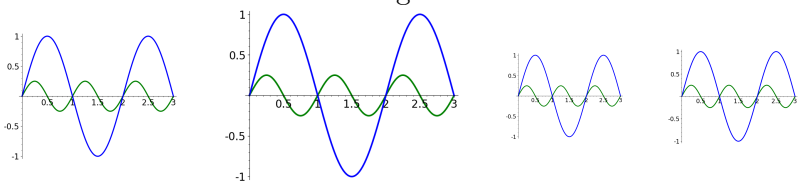


Gráficas de seno




Gráficas de coseno

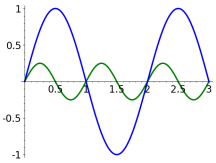
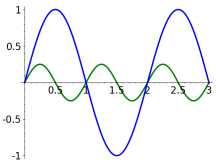
Hablando de alineamientos, si te fijas en el ejemplo de la sucesión de imágenes `fsin.png`, están alineadas por su base. Metiendo las cuatro primeras en un entorno `tabular` conseguimos centrarlas verticalmente:



Por si tienes curiosidad, LaTeX utiliza *cajas* para situar los elementos en cada línea y cada caja tiene una *línea base* que se continua a lo largo de la línea (por ejemplo, la caja de la letra "p" tiene su línea base justo bajo su bucle, por eso vemos "ap" en lugar de "aP". En las imágenes, la línea base es la de abajo mientras que en las tablas pasa por la mitad. En particular,

una imagen en la misma línea que un texto siempre sobresaldrá por arriba mientras que por abajo queda a ras de la línea.

Es útil tener esto en mente sobre todo al combinar imágenes con fórmulas o textos. Por ejemplo,

$y=\sin(\pi nx)\to$  $y=\sin(\pi nx)\to$ 

responde (como *displayed formula*) a

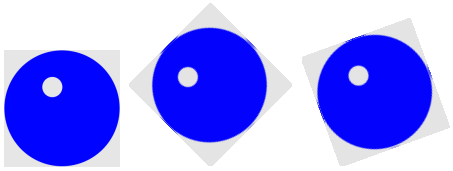
```
y=\sin(\pi nx)\to
\includegraphics[scale=0.1]{fsin.png} \qquad
y=\sin(\pi nx)\to
\begin{tabular}{c}
 \includegraphics[scale=0.1]{fsin.png}
\end{tabular}
```

En la mayor parte de los casos desearemos algo como lo segundo.

Hay otros parámetros posibles en `\includegraphics`, aquí solo veremos los relativos a los giros. Con `angle=`$n$ la figura se girará en sentido antihorario el número de grados especificados por $n$. Por ejemplo,

```
\includegraphics[scale=0.2]{circua.png}
\includegraphics[scale=0.2, angle=45]{circua.png}
\includegraphics[scale=0.2, angle=20]{circua.png}
```

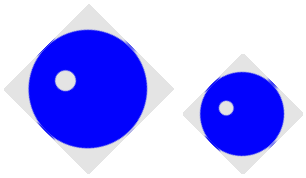
muestra la imagen de prueba `circua.png` reducida a un quinto de su tamaño original y esta misma girada 45° y 20°.



Hay que tener en cuenta que fijar cierta altura y girar después no es lo mismo que girar y fijar cierta altura a continuación. Por ello con

```
\includegraphics[height=45pt, angle=45]{circua.png}
\includegraphics[angle=45, height=45pt]{circua.png}
```

no obtenemos dos figuras iguales:



La altura de la segunda figura es la misma que el lado del cuadrado de la primera: `45 pt` porque los parámetros de leen de izquierda a derecha.

Si nos fijamos en los ejemplos anteriores veremos que por defecto el giro se produce por la esquina inferior izquierda, estrictamente con respecto al llamado *punto de referencia*, situado a la izquierda de la línea base. La manera de modificar este comportamiento es por medio del parámetro `origin=`*c* donde *c* es una cadena de uno o dos caracteres con los siguientes significados:

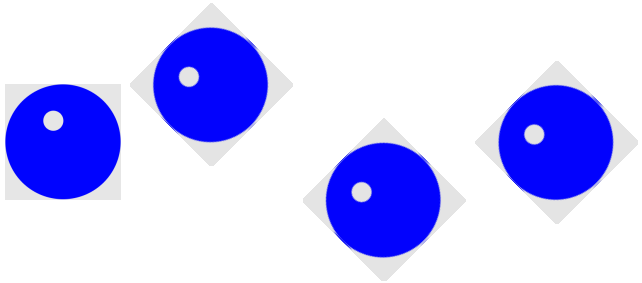
| | | | |
|---|---|---|---|
| `l` → izquierda, | `r` → derecha, | `c` → centro, | |
| `t` → arriba, | `b` → abajo, | `c` → centro, | `B` → línea de base. |

Las de la primera línea dan la alineación horizontal y las de la segunda la vertical, por eso se ha repetido `c`. Como ya se ha mencionado, por defecto la línea base es la parte de abajo de la figura y `B` se convierte en un parámetro innecesario porque equivale a `b`. Con conocimientos avanzados se podría cambiar la línea base y habría diferencia entre el efecto de `b` y `B`.

Se puede abreviar `cc` por `c`. Por ejemplo,

```
\includegraphics[scale=0.2]{circua.png}
\includegraphics[scale=0.2, origin=tl, angle=45]{circua.png}
\includegraphics[scale=0.2, origin=br, angle=45]{circua.png}
\includegraphics[scale=0.2, origin=c, angle=45]{circua.png}
```

aparte de la figura original, muestra los giros de 45° por la esquina superior izquierda, por la inferior derecha y por el centro de la imagen:



Además de `origin=c`, también se admiten expresiones con cualquiera de los otros caracteres en solitario, como `origin=l`. En ese caso se interpreta que el carácter ausente es `c`.

## 3. La clase beamer

La clase más usada para hacer presentaciones es `beamer`. El hecho de que sea una *clase* en vez de un paquete significa que es un posible argumento de `\documentclass`. El resto de la cabera la podemos conservar como siempre, salvo que en ejemplos anteriores con `article` no hemos aprovechado las opciones para poner título y autor y es muy raro que no las necesitemos en una presentación. Un ejemplo bastante escueto de la cabecera de una presentación que vaya a contener símbolos matemáticos es:

```
\documentclass{beamer}

\usepackage{amsmath}
\usepackage{amsfonts}
\usepackage{amssymb}

\usepackage[spanish]{babel}
\usepackage[utf8]{inputenc}


\title{Ejemplo de presentación}
\author{Fernando Chamizo}
\institute{Universidad Autónoma de Madrid}
\date{24 de noviembre de 2023}
```

Como siempre, la codificación depende de tu sistema y quizá prefieras o necesites `\usepackage[latin1]{inputenc}`. También hay que aclarar, sobre todo si revisas código de otras personas, es que una cabecera `beamer` tan breve es muy inusual. Hay tantos parámetros a controlar en una presentación que las cabeceras suelen ser muchísimo más complejas que las de `article`.

La idea básica en la clase `beamer` es que las diferentes diapositivas que conforman la presentación se separan encerrando sus contenidos en un entorno llamado `frame`. Por ejemplo:

```
\begin{frame}
 Mi primera diapositiva
\end{frame}
```

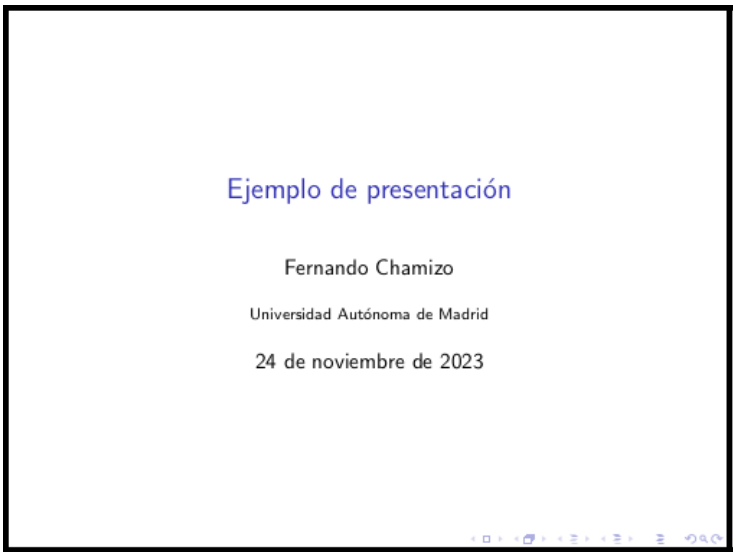
El tipo de letra deja de ser la habitual no solo por la familia empleada (palo seco) sino por el tamaño. No tendría sentido que el tamaño de letra en una presentación fuera similar al de un artículo (aunque a veces algún conferenciante poco avezado utilice uno de sus artículos como parte de su presentación). También vemos que el formato de cada página ha cambiado. Ahora está apaisada para que la visualicemos adecuadamente con un proyector de vídeo (*beamer*, en inglés). Además abajo a la derecha aparecen

unos pequeños símbolos que responden al ratón para una navegación básica por las diapositivas (la verdad es que no los he encontrado útiles en mi experiencia).

Lo normal es comenzar una presentación no por una diapositiva como la anterior sino con el título y autor. El comando `\titlepage` genera esta información a partir de lo escrito en la cabecera anterior. Entonces sustituimos la diapositiva anterior comenzando con:

```
\begin{frame}
 \titlepage
\end{frame}
```

Tampoco es que el resultado sea demasiado espectacular:




El recuadro negro no aparece, es solo para mostrar aquí los límites de la página.

En cada diapositiva se puede poner un título con `\frametitle{...}`. Además varias diapositivas pueden estar dentro de una sección que se indica con `\section{...}`, como ya sabemos. Muchas veces en las presentaciones el texto está en recuadros con un título. Esto se consigue con un entorno `block` cuyo único argumento es el título, que puede dejarse vacío.

Tomemos como continuación de nuestra diapositiva de título una sección llamada "Primera" con dos diapositivas, la segunda con bloques:

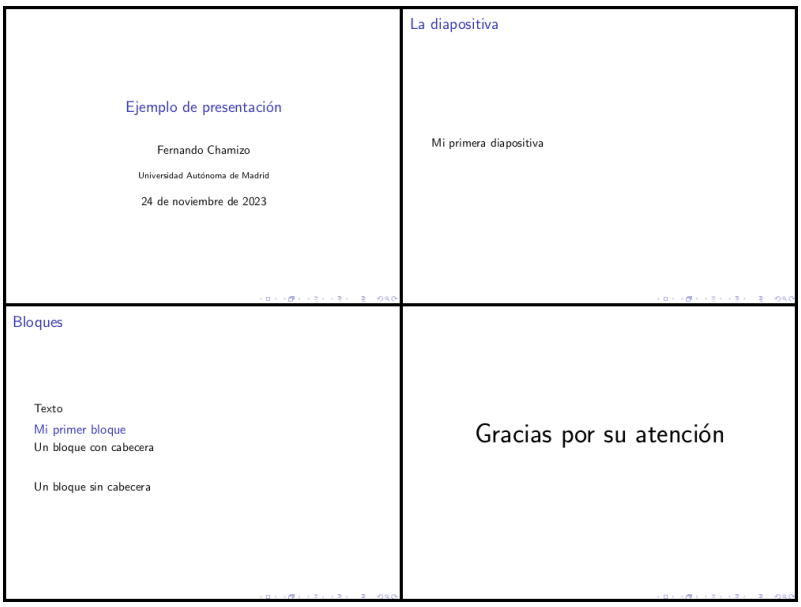
```
\section{Primera}
\begin{frame}
  \frametitle{La diapositiva}
  Mi primera diapositiva
\end{frame}
```

```
\begin{frame}
  \frametitle{Bloques}
  Texto
  \begin{block}{Mi primer bloque}
  Un bloque con cabecera
  \end{block}
  \begin{block}{}
  Un bloque sin cabecera
  \end{block}
\end{frame}
```

y acabemos con una diapositiva de agradecimiento dentro de una sección llamada “Fin”:

```
\section{Fin}
\begin{frame}
  \begin{center}
    \Huge Gracias por su atención
  \end{center}
\end{frame}
```

Si lo compilamos obtendremos cuatro páginas del tipo




Esto es bastante soso y las secciones no se muestran por ningún lado. Para decorar las presentaciones se pueden usar “temas” y “colores de tema”. Lo primero se consigue con `\usetheme{...}` (en la cabecera) y lo segundo con `\usecolortheme{...}`. Esta es una brevísima tabla que solo recoge tres posibilidades para cada uno:

| \usetheme | Frankfurt | Madrid | Warsaw |
|---|---|---|---|
| \usecolortheme | beaver | seahorse | rose |

Hay que elegir a lo más un tema y un color de tema. Si cargamos dos temas es posible que haya incompatibilidades y obtengamos errores al compilar.

Si en el ejemplo anterior cargamos `Frankfurt` con

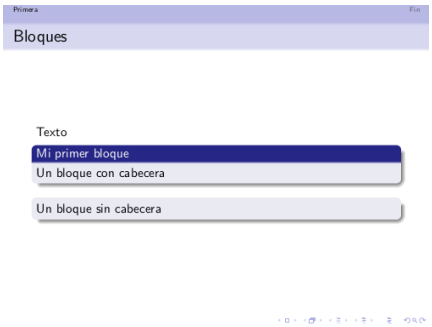
```
\usetheme{Frankfurt}
```

sin especificar ningún color de tema, ya veremos cambios sustanciales. En la primera diapositiva el título aparecerá dentro de un cuadrado azul sombreado y en la parte superior de todas las páginas aparecerá el nombre de sección en pequeño de forma que al pinchar con el ratón pasamos automáticamente a ellas. Por otro lado los bloques aparecerán destacados.

Hay tantos temas que es imposible dar una idea de todos los posibles resultados. Solo como ilustración se muestra una miniatura de la página que contiene los bloques para todas las combinaciones de los temas y colores de tema antes mencionados. Además se muestra al final la diapositiva del título con `Madrid` y `rose`.


`Frankfurt, beaver`


`Frankfurt, rose`



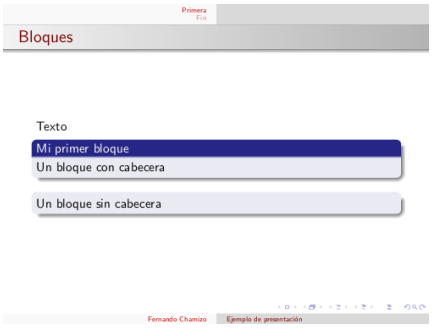

`Frankfurt, seahorse`


`Madrid, beaver`

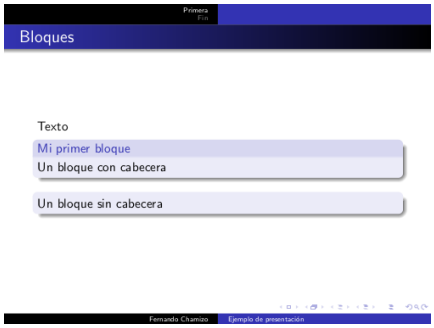

`Madrid, rose`


`Madrid, seahorse`

---

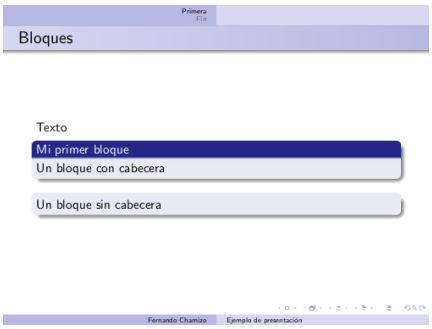



`Warsaw, beaver`




`Warsaw, rose`

---




`Warsaw, seahorse`


`Madrid, rose` (título)

---

En algunos casos, como `Madrid` con `rose` obtenemos *overfulls* porque el texto que resume los datos de la cabecera no cabe bien en la línea inferior. Una manera de evitarlo es poner un parámetro entre corchetes en estos datos que indique qué debe mostrar como versión abreviada. Por ejemplo, en el caso mencionado lo normal sería cambiar

```
\institute{Universidad Autónoma de Madrid}
```

por

```
\institute[UAM]{Universidad Autónoma de Madrid}
```

para abreviar por "UAM", o incluso por

```
\institute[]{Universidad Autónoma de Madrid}
```

si queremos que no aparezca la institución al pie de cada página más allá del título.

Dicho sea de paso, en LaTeX hay un comando llamado `\tableofcontents` que, como su nombre indica, produce una tabla de contenidos. En un artículo corto, con la clase `article`, es de poco uso, pero muchas presentaciones tras la diapositiva del título tienen otra de la forma

```
\begin{frame}
 \tableofcontents
\end{frame}
```

que muestra, numeradas, todas las secciones que hemos definido. Si al compilar no ves la lista, compila de nuevo. Los ficheros con la clase `beamer` a veces necesitan varias pasadas, lo que está relacionado con que generan varios ficheros auxiliares que aparecen en el directorio de la fuente con el mismo nombre y diferentes extensiones.

Las diapositivas de las presentaciones a menudo se muestran haciendo aparecer sus contenidos poco a poco. Una manera muy básica de conseguirlo es con el comando `\pause` que espera una pulsación para seguir avanzando. Más versátil es el comando `\uncover<n>{...}` que descubre la parte contenida en los puntos suspensivos cuando se alcanza la pulsación $n$-ésima dentro de la diapositiva. Si en vez de `<n>` escribimos `<n->`, será visible a partir de la pulsación `n` y si escribimos `<-n>`, hasta antes de ella.

Por ejemplo, si cambiamos el código de la diapositiva que contiene los bloques a

```
\begin{frame}
\frametitle{Bloques}
\pause
Texto
\uncover<3>{\begin{block}{Mi primer bloque}
 Un bloque con cabecera
\end{block}}
\begin{block}{}
 Un bloque sin cabecera
\end{block}
\end{frame}
```

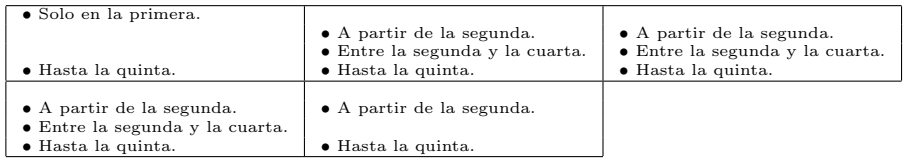
entonces en la primera pulsación que nos lleva a esa diapositiva la veremos vacía salvo el título "Bloques", en la siguiente aparecerán "Texto" y el bloque

sin cabecera, dejando hueco para el bloque titulado “Mi primer bloque” que aparecerá en la tercera y última pulsación.

Este esquema funciona de forma abreviada en las listas y enumeraciones creadas con `itemize` y `enumerate` donde se puede usar `\item<...>`. Por ejemplo el resultado de

```
\begin{frame}
\frametitle{Lista caótica}
 \begin{itemize}
  \item<1> Solo en la primera.
  \item<2-> A partir de la segunda.
  \item<2-4> Entre la segunda y la cuarta.
  \item<-5> Hasta la quinta.
 \end{itemize}
\end{frame}
```

sería una diapositiva con título “Lista caótica” cuya apariencia en miniatura en cinco pulsaciones sería aproximadamente:




Usando el entorno `enumerate`, los números aparecen con un bonito efecto tridimensional.

Hay una serie de colores `beamer` que se pueden modificar para cambiar el comportamiento por defecto de un estilo. Por ejemplo si incluimos en la cabecera, después de `\usetheme` y `\usecolortheme`,

```
\setbeamercolor{title}{fg=black, bg=green!30!white}
\setbeamercolor{frametitle}{fg=magenta, bg = red!10!white}
%\setbeamercolor{author}{fg=black, bg=green!30!white}
```

El fondo (`bg` de *background*) del título estará en un verde claro y el texto (`fg` de *foreground*) en negro. Por otra parte, los títulos de las diapositivas estarán en magenta con un fondo rosa muy claro. El comando comentado con `%` actuaría sobre el nombre del autor, pero seguramente ahí no queremos cambiar el fondo (si lo dudas, prueba a quitar `%`).

También se pueden modificar otras cosas. Por ejemplo, en el tema `Madrid` con color de tema `rose` aparecen abajo a la derecha unos símbolos de navegación que permiten moverse entre las diapositivas, pero que no resultan muy útiles en la práctica porque son pequeños e incómodos. La manera de suprimirlos es incluir en la cabecera:

```
\setbeamertemplate{navigation symbols}{}
```

Es posible hacer también modificaciones globales o locales de los colores de los bloques. Por ejemplo, si en una diapositiva incluimos un bloque en la forma:

```
{\setbeamercolor{block title}{bg=green, fg=red}
\setbeamercolor{block body}{bg=cyan!10, fg=blue}
\begin{block}{Título del bloque}
 Cuerpo del bloque.
\end{block}}
```

entonces ese bloque tendrá el título en fondo verde y texto rojo mientras que el cuerpo del bloque tendrá fondo azul muy claro y texto azul. Si incluimos en la cabecera las líneas

```
\setbeamercolor{block title}{bg=green, fg=red}
\setbeamercolor{block body}{bg=cyan!10, fg=blue}
```

el efecto será global, en todos los bloques. La única precaución al respecto es que, como antes, hay situarlas en algún punto después de `\usetheme` y `\usecolortheme` porque los temas, como hemos visto, actúan sobre los colores.

Con `beamer` podemos emplear todo lo que hemos aprendido[4], pero no hay que perder de vista, que el propósito de una presentación es totalmente distinto del de un artículo o un trabajo. Los principiantes que solo tienen experiencia con estos últimos, tienden a hacer presentaciones muy densas y con muchos símbolos. Este es un error a evitar. Los asistentes a una presentación no pueden pasar páginas y por tanto solo tendrán acceso a lo que vean en la diapositiva en curso y a lo, posiblemente poco, que recuerden de las anteriores. Cada diapositiva debe tener pocas líneas y ser esquemática. Además, es conveniente que sacrifiquemos las definiciones y notaciones que no sean absolutamente necesarias. A lo que debemos aspirar en la mayoría de los casos es a transmitir ideas, no detalles.

Es común entre los expertos utilizar extensivamente en `beamer` el paquete `tikz`. Este es un paquete gráfico bastante complicado que, entre otras cosas, proporciona un control total sobre las posiciones y que, en las presentaciones, permite una colocación precisa de los elementos que integran cada diapositiva. Esta colocación va contra la filosofía básica de dejar actuar a LaTeX, tantas veces elogiada antes. La justificación de esta paradoja es que una presentación difiere típicamente mucho de un texto con fórmulas dividido en páginas, párrafos y líneas, que es el objetivo original de LaTeX. Algunas presentaciones `beamer` de expertos emplean LaTeX fuera de `tikz` para poco más que para incluir fórmulas.

[4]Para decir toda la verdad, hay algún tipo de incompatibilidad ocasional, por ejemplo con el entorno `verbatim`.

# La herramienta BibTeX

**Lección 10**

## 1. Motivación

Recordemos que el entorno `thebibliography` permitía crear una lista de referencias bibliográficas. Por ejemplo, una bibliografía con dos referencias podría ser:

```
\begin{thebibliography}{9}
\bibitem{lamport}
L. Lamport. \emph{\LaTeX: A document preparation system}.
Addison Wesley, Massachusetts, 2nd ed, 1994.

\bibitem{spivak}
M. Spivak. \emph{The Joy of \TeX: A Gourmet Guide to
Typesetting With the {\AmS-\TeX} Macro Package}.
Addison-Wesley Professional, 1990.
\end{thebibliography}
```

El problema es que, como ya se indicó, no hay un acuerdo medianamente general acerca de los tipos de letra y la ordenación de las partes que componen las referencias. Cada editorial tiene sus propias reglas a las que deben adaptarse los autores. Si en nuestro trabajo de fin de grado u otro documento nos impusieran un formato, el procedimiento sería buscar cada referencia en la red, copiarla y ajustarla manualmente de acuerdo con las directrices que nos han dado. Si mañana queremos reciclar las referencias estaremos en un apuro en caso de que nos exijan otro formato. Quizá en el código anterior tengamos que quitar `\emph`, mover el número de año a otra posición o pasar la inicial detrás del nombre.

Si lees artículos de investigación y te fijas en las referencias, te darás cuenta de hasta qué punto se extienden las variantes en el formato. Por ejemplo, en los artículos de matemáticas es casi universal listar la bibliografía por orden alfabético. Sin embargo, en física teórica hay una tradición de listar por orden de cita: en el ejemplo anterior si uno citase antes a Spivak que a Lamport, entonces habría que intercambiar el orden. De este modo, lo que está más arriba en la lista de referencias hay que buscarlo en las primeras páginas. En ciertas publicaciones se prescinde de los títulos en la bibliografía, lo cual a algunos nos parece una barbaridad. Con mi limitada experiencia, mi

impresión es que en matemáticas se tiende a poner referencias más completas y ordenadas que en otras áreas científicas.

Lo ideal para reciclar referencias sería tener una forma "pura" de ellas, sin incluir ningún formato ni ninguna ordenación privilegiada, y en cada situación especificar el formato que nos piden. La herramienta BibTEX va en esa línea. Uno puede tener un fichero enorme con todas las referencias que conoce (u otros más pequeños para temas específicos) e incluir un par de instrucciones en su fuente LATEX para indicar que quiere utilizar referencias de ese fichero en cierto formato. Desde hace algunos años, conseguir en la red referencias científicas (sobre todo matemáticas) en esa forma pura es muy simple y BibTEX se ha popularizado rápidamente. Quizá la mayor desventaja de esta herramienta es que no resulta nada fácil construir formatos propios, por tanto, uno depende de los predeterminados o de los que ofrecen las editoriales. Hay alguna herramienta complementaria reciente para crear formatos, pero por ahora es minoritaria.

## 2. Cómo funciona

L. Lamport, el creador de LATEX, escribió un artículo titulado *How to write a proof*. Supongamos que busco la referencia BibTEX (más adelante veremos dónde hacerlo). Obtendré algo del tipo:

```
@article {MR1349872,
    AUTHOR = {Lamport, Leslie},
     TITLE = {How to write a proof},
   JOURNAL = {Amer. Math. Monthly},
  FJOURNAL = {American Mathematical Monthly},
    VOLUME = {102},
      YEAR = {1995},
    NUMBER = {7},
     PAGES = {600--608},
      ISSN = {0002-9890},
   MRCLASS = {00A35},
  MRNUMBER = {1349872},
MRREVIEWER = {Wann-Sheng Horng},
       DOI = {10.2307/2974556},
       URL = {https://doi.org/10.2307/2974556},
}
```

A decir verdad la mayor parte de los sitios me darán algo más breve, pero eso es indiferente porque nosotros simplemente lo copiamos en un fichero de texto con la extensión `.bib`. Por ejemplo, llamemos a este fichero `mibiblio.bib`. Añadamos otra referencia BibTEX, esta vez un libro sobre algoritmos del creador de TEX:

```
@book {MR1762319,
    AUTHOR = {Knuth, Donald E.},
     TITLE = {Selected papers on analysis of algorithms},
    SERIES = {CSLI Lecture Notes},
    VOLUME = {102},
 PUBLISHER = {CSLI Publications, Stanford, CA},
      YEAR = {2000},
     PAGES = {xvi+621},
      ISBN = {1-57586-211-5; 1-57586-212-3},
   MRCLASS = {68Q25 (01A75 68W40)},
  MRNUMBER = {1762319},
MRREVIEWER = {A. D. Booth},
       DOI = {10.1093/logcom/10.4.621},
       URL = {https://doi.org/10.1093/logcom/10.4.621},
}
```

Lo que aparece en la primera línea entre la llave y la coma es el nombre que deseamos dar a la referencia. Como `MR1349872` y `MR1762319` a nosotros no nos dicen gran cosa, los cambiamos por nombres más sugestivos, por ejemplo por `lamport` y `knuth`, respectivamente.

Ahora ponemos al final de nuestro documento, antes de `\end{document}` las siguientes líneas:

```
\bibliography{mibiblio}
\bibliographystyle{plain}
```

La primera indica el fichero de bibliografía y la segunda el formato. Si el fichero `mibiblio.bib` no está en el mismo directorio que nuestro documento LaTeX, debemos especificar el *path* correspondiente.

Las referencias se citan como sabemos. Digamos que escribimos

```
El libro \cite{knuth} y el artículo \cite{lamport}
```

Para procesarlo debemos compilar con LaTeX, de la forma habitual, y después ejecutar BibTeX. Para que todo se actualice quizá tengamos que compilar con LaTeX dos veces más. El editor Kile detecta automáticamente que hay un `\bibliography` y ejecuta BibTeX compilando las veces necesarias. Lo mismo ocurre con Overleaf. Con TeXStudio y TeXmaker se ejecuta BibTeX pulsando[1] F8 o F11. Para mí es un misterio el número de veces (una o dos) que hay que pulsar estas teclas para que al compilar se actualice la bibliografía.

Al procesar de este modo, en el texto obtendremos "El libro [1] y el artículo [2]" y al final de nuestro documento un apartado del tipo:

[1] Si uno usa BibTeX a menudo, es conveniente configurar la compilación rápida de estos editores para que lo ejecute y no haya que estar pulsando una sucesión de teclas.

## Referencias

[1] Donald E. Knuth. *Selected papers on analysis of algorithms*, volume 102 of *CSLI Lecture Notes*. CSLI Publications, Stanford, CA, 2000.

[2] Leslie Lamport. How to write a proof. *Amer. Math. Monthly*, 102(7):600–608, 1995.

La ordenación es alfabética, como es habitual en matemáticas. Si solo pusiéramos `El libro \cite{knuth}` entonces la segunda referencia no aparecería. Si queremos que una referencia no citada aparezca debemos usar `\nocite{...}`, en este caso `\nocite{lamport}`. Una forma extrema de este comando es `\nocite{*}` que incluye todas las referencias que hay en nuestro fichero de bibliografía `.bib`.

Respecto al formato, el argumento de `\bibliographystyle`, hay unas pocas opciones que seguramente tienes disponibles con tu instalación de LaTeX y las revistas científicas a veces te ofrecen ficheros propios con extensión `.bst` que introducen una nueva posibilidad. Aquí solo veremos como alternativa a `plain`, el formato `alpha` que en vez del número indica una abreviatura del apellido y el año. En el ejemplo anterior daría lugar a:

## Referencias

[Knu00] Donald E. Knuth. *Selected papers on analysis of algorithms*, volume 102 of *CSLI Lecture Notes*. CSLI Publications, Stanford, CA, 2000.

[Lam95] Leslie Lamport. How to write a proof. *Amer. Math. Monthly*, 102(7):600–608, 1995.

Si te estás preguntando qué política sigue el formato `alpha` cuando hay varios autores o cuando en dos referencias coinciden las primeras letras del apellido y el año, te recomiendo que experimentes por tu cuenta.

# 3. Modificaciones y posibles errores

Retocar un poco nuestro fichero `.bib` no cuesta ningún esfuerzo. Por ejemplo, si tenemos muy claro que es preferible no poner nombres de pila en las referencias sin abreviatura, simplemente debemos cambiar “Donald” por “D.” y “Leslie” por “L.”. En otras ocasiones, seremos incapaces de localizar algunas referencias en formato BibTeX porque son trabajos muy antiguos o con poca difusión. En ese caso, crearemos nuestra propia entrada BibTeX ajustando otra que tengamos a mano. Tomando como base el ejemplo anterior, aquí está una plantilla natural para un artículo:

```
@article {referencia,
    AUTHOR = {nombre del autor},
     TITLE = {título del artículo},
   JOURNAL = {nombre de la revista},
    VOLUME = {número de volumen},
      YEAR = {año},
    NUMBER = {número de ejemplar},
     PAGES = {páginas},
}
```

y para un libro:

```
@book {referencia,
    AUTHOR = {nombre del autor},
     TITLE = {título del artículo},
 PUBLISHER = {Editorial},
      YEAR = {año},
}
```

Últimamente se citan muchos artículos del repositorio `arXiv`. Suelo utilizar en ese caso una plantilla del tipo:

```
@UNPUBLISHED {referencia,
    AUTHOR = {nombre del autor},
     TITLE = {título del artículo},
      YEAR = {año},
      NOTE = {código en arXiv},
       URL = {dirección url},
}
```

Usando `plain` no se mostrará la dirección URL, pero a menudo encuentro útil que esté escrita para consultarla.

Otra razón para modificar el fichero `.bib` es que contenga algún comando matemático de versiones antiguas de LaTeX. Por ejemplo, en algún título es posible encontrar `${1\over 2}$` en lugar de `$\frac{1}{2}$`, lo cual no causa estrictamente un error, pero sí hace saltar un aviso recomendando que lo cambiemos. También puede haber problemas ocasionales con acentos o caracteres especiales.

Hay dos ficheros con las extensiones `.blg` y `.bbl` que se generan cada vez que se usa la herramienta BibTeX y que tienen cierta utilidad. En el primero se recogen los detalles técnicos al ejecutar BibTeX y los posibles errores. No es imposible que ocurran porque hayamos copiado mal una referencia o la hayamos duplicado o por otros detalles relacionados con versiones de LaTeX o la codificación. En suma, el fichero `.blg` es el análogo para la bibliografía del fichero `.log` para LaTeX.

Por otro lado, en el fichero con extensión `.bbl` se encuentra una versión de la bibliografía con el entorno `thebibliography`. Es de gran interés si queremos compartir nuestro documento con alguien que no usa BibTEX o simplemente no deseamos hacer público nuestro, quizá enorme, fichero `.bib`. Lo único que tenemos que hacer es reemplazar las líneas de `\bibliography` y `\bibliographystyle` por el contenido de este fichero y todo funcionará igual, pero sin BibTEX.

## 4. Dónde obtener referencias en formato BibTEX

Ya se ha sugerido antes que el éxito reciente de BibTEX está relacionado con que ahora es posible obtener muchas referencias en el formato de esta herramienta. Posiblemente si ponemos en un buscador una referencia medianamente completa en internet, quizá añadiendo "bibtex", nos costará solo unos pocos *clicks* conseguir el formato buscado.

La mayor parte de las revistas científicas prestigiosas no son de acceso abierto, pero sí ofrecen a veces la posibilidad de descargar referencias en formato BibTEX. Por ello, si buscamos un artículo que hayan publicado y nos interese citar, no es raro que descubramos algún botón en sus sitios *web*, con nombres a veces un poco crípticos (como "Export citation" o "Tools"), que nos ofrezca lo que debemos copiar en nuestro fichero de bibliografía.

A continuación se mencionan tres posibilidades fáciles de usar. La primera es de carácter general mientras que las otras están orientadas a matemáticas.

- Google Académico (`https://scholar.google.com/`)
- zbMATH (`https://zbmath.org/`)
- Mathscinet (`https://mathscinet.ams.org/mathscinet/`, necesitas el vpn de la UAM).

En Google Académico buscamos una aproximación de nuestra referencia (por ejemplo, el título) y nos saldrá una lista de trabajos que se ajustan a ello. Al final de cada posibilidad, antes del número de citas, hay unas dobles comillas. Al pinchar sobre ellas veremos diferentes formatos para citar y, obviamente, elegimos BibTEX.

En zbMATH podemos afinar mejor la búsqueda eligiendo los campos. A cambio, si no tenemos suscripción, solo nos ofrecerá los tres resultados más relevantes, con lo cual a veces tendremos que especificar bastante. Pinchando en el botón BibTEX, obtendremos el resultado.

Mathscinet es lo más completo para referencias matemáticas que no sean muy antiguas (digamos de menos de 100 años), pero se necesita una suscripción (la UAM la tiene). Buscando la referencia, señalamos "Select alternative format" y allí BibTEX.

Hoy en día una parte sustancial de nuestra información rápida proviene de la Wikipedia. En cada página en el menú "Herramientas" hay un enlace "Citar esta página" ("Tools" y "Cite this page", en la versión en inglés). Si lo seguimos, llegaremos a una página de la que podemos copiar el formato BibTEX. Allí se aclara que con el paquete `url` obtendremos mejores resultados. Este es un paquete que define el comando `\url{...}` para mencionar páginas web. Como siempre, lo cargamos con `\usepackage{url}` en la cabecera. El argumento de `\url` en las citas de la Wikipedia apunta a una página estática correspondiente al día de la consulta, para que no induzca a confusión tras actualizaciones posteriores.

Seguro que no soy el único al que el aspecto de la cita de páginas de la Wikipedia obtenida de esta manera le parece mejorable. Siempre puede uno ajustar las cosas a su gusto para un número pequeño de referencias, pero sería ridículo perder más tiempo en componer una bibliografía con BibTEX que en utilizar el procedimiento manual con el entorno `thebibliography`.