Scrivere in LaTeX

Laboratorio di Introduzione alla Matematica Computazionale

1 Primi passi con LaTeX

Un documento LaTeX è costituito da uno (o più, come vedremo in seguito) file di testo, solitamente con estensione `.tex`. Supponiamo, in queste note, di voler creare un file di appunti, e di chiamarlo `appunti.tex`.

Esercizio 1

Si crei un file `appunti.tex` con il seguente contenuto:

```
\documentclass[a4paper]{article}
\title{Appunti di Analisi}
\author{Me stesso}
\begin{document}
  \maketitle
  Testo di prova
\end{document}
```

A questo punto si può procedere in due modi.

- **Compilazione sulla propria macchina.** Si apra un terminale nella cartella in cui si trova il file e si esegua `pdflatex appunti.tex`. Se la compilazione termina correttamente, nella stessa cartella verrà creato il file `appunti.pdf`, insieme ad alcuni file ausiliari.
- **Compilazione online.** Si acceda a `https://overleaf.unipi.it`, si crei un nuovo progetto (oppure se ne apra uno esistente) e si copi il contenuto nel file principale del progetto, ad esempio `main.tex`. Premendo *Recompile*, Overleaf esegue la compilazione sul server e mostra direttamente il PDF ottenuto. In questo caso non è necessario eseguire manualmente il comando `pdflatex` da terminale.

Dopo aver completato l'Esercizio 1 avremo compilato il nostro primo documento LaTeX! Una cosa da notare è che non sono stati creati solamente il file `appunti.pdf`, ma anche dei file ausiliari come `appunti.log` e `appunti.aux`.

Se si compila in locale, questi file compaiono nella stessa cartella del sorgente; su Overleaf vengono invece gestiti automaticamente nell'area di compilazione del progetto e, nella maggior parte dei casi, non è necessario occuparsene direttamente. Per il momento non li discuteremo.

Analizziamo invece i vari comandi che abbiamo dato nel documento. Un comando (o, più precisamente, una macro) in LaTeX ha la forma:

```
\nomecomando[opt1,opt2][opt3]{arg1}{arg2}{arg3}
```

dove il carattere `\` indica l'inizio di un comando, i parametri `opt1`, `opt2` e `opt3` sono parametri opzionali, che potrebbero essere omessi, mentre `arg1`, `arg2` e `arg3` sono parametri obbligatori. Il numero di comandi opzionali e/o obbligatori dipende dal comando in considerazione. Analizziamo il caso del documento minimale da noi prodotto.

`\documentclass` serve a specificare le caratteristiche del documento. Ad esempio, nel nostro caso abbiamo usato `\documentclass[a4paper]{article}` per impostare la dimensione della pagina ad A4 (lo standard usuale in Europa), e il tipo di documento ad `article`. Esistono altri tipi di documenti, come `minimal`, `book`, ...

`\title` **e** `\author` servono ad indicare titolo ed autore del documento. Sono entrambi comandi che prendono esattamente un parametro. Attenzione: questi comandi non producono il testo, ma impostano solamente questi parametri all'interno del documento, per poter essere utilizzati in seguito.

`\begin{document}` **ed** `\end{document}` si utilizzano per iniziare e concludere la parte del documento che contiene il documento vero e proprio. Tutto quello che precede il comando `\begin{document}` non viene riprodotto sul documento (serve ad impostare parametri globali), e tutto quello che segue `\end{document}` viene ignorato. La parte che precede il comando `\begin{document}` si chiama preambolo.

`\maketitle` è un comando che stampa titolo, autore e data. L'output preciso dipende dal tipo di documento che è stato scelto (`book`, `article`, ecc.).

Il testo inserito dopo il comando `\maketitle` è semplice testo che viene inserito nel documento. Questo testo verrà formattato con il carattere standard all'interno del documento.

I comandi del tipo `\begin{...}`, `\end{...}` servono a delimitare gli ambienti. Un ambiente è una porzione di codice dove, solitamente, vengono applicati degli stili particolari, e che può essere utilizzato per entrare in qualche altra modalità del TeX. Ad esempio, ci sono ambienti per creare liste puntate e numerate, o per allineare il testo. Ci sono anche ambienti che permettono di inserire figure, tabelle, ecc., come si vedrà nella sezione dedicata a grafici, immagini e tabelle.

Esercizio 2

Si modifichi il testo precedente racchiudendo la stringa "Testo di prova" all'interno di un ambiente di tipo `center`, ovvero ottenendo:

```
\begin{center}
Testo di prova
\end{center}
```

Cosa succede (non troppo inaspettatamente)?

LaTeX ignora gli spazi ripetuti: uno spazio o molti sono rappresentati come un unico spazio, e così vale anche per le nuove linee. L'unica eccezione a questa regola è il caso in cui si lascia una riga vuota, che viene interpretato come l'istruzione "Comincia un nuovo paragrafo".

L'usuale convenzione tipografica è di iniziare un nuovo paragrafo per separare le varie parti logiche del discorso. Questo crea automaticamente un rientro a sinistra per la prima riga. È possibile forzare una nuova linea con il comando `\\`, e questo non crea un nuovo paragrafo (ovvero: nessun rientro per la prima riga); è però sconsigliato a meno che non ci siano buone ragioni per farlo.

1.1 Formattare il testo

Abbiamo visto in precedenza come formattare il testo passando al grassetto o all'italico in TeX. I comandi `\bf` e `\it` che abbiamo usato stanno per "Bold Face" e "Italic", che si usano in tipografia per indicare il grassetto ed il corsivo (nel senso di font inclinati, indicati anche come slanted in inglese).

Questa sintassi è di solito evitata in LaTeX, e si preferisce utilizzare dei comandi che prendono come argomento il testo da modificare, invece che costringersi a racchiudere il testo all'interno di blocchi.

Ad esempio, invece di scrivere:

```
Questa {\bf parola} e' in grassetto.
```

si utilizza:

```
Questa \textbf{parola} e' in grassetto.
```

Similmente, si utilizza `\emph` invece di `\it`. Esiste anche il comando `\textit`, che si allinea più fedelmente al comando `\it`, mentre il comando `\emph` ha più precisamente lo scopo di enfatizzare il testo. Renderà dunque corsivo del testo all'interno di una regione con font normale, e invece rimuoverà il corsivo ad una parola in una zona dove il font è già corsivo (ad esempio all'interno dell'enunciato di un teorema). Questo ha lo scopo di far spiccare il testo rispetto a quello che lo circonda, e si usa spesso nelle definizioni.

Esercizio 3

Provate a riscrivere un breve esempio di testo formattato usando i comandi `\textbf` e `\emph`, seguendo le convenzioni sopra.

1.2 Suddividere il testo

Durante la stesura di un documento, è naturale suddividerlo in sezioni. Questa suddivisione viene gestita automaticamente da LaTeX: è sufficiente indicare la struttura del documento, e titoli di sezioni, indici, ecc., verranno gestiti in maniera completamente automatica.

Ad esempio, si provi a dare il comando `\section{Nuova sezione}`. Questo si occuperà di stampare sul testo il titolo di una nuova sezione, con la corrispondente numerazione. Nel caso volessimo creare una nuova sezione senza numero, potremmo usare il modificatore `*`; il comando `\section*{Sezione senza numero}` creerà una sezione con il nome specificato, ma senza incrementare e stampare il numero della sezione corrente.

Esercizio 4

Si aggiungano sezioni, sottosezioni e paragrafi al vostro documento utilizzando i comandi `\section`, `\subsection`, `\subsubsection` e `\paragraph`. Provate anche il modificatore `*` per creare alcune sezioni senza numero.

In alcuni tipi di documenti può essere utile produrre un sommario all'inizio che lista le sezioni presenti nel documento. Questo può essere ottenuto tramite il comando `\tableofcontents`, che funziona in modo simile a `\maketitle`.

Attenzione: Perché il comando `\tableofcontents` funzioni nel modo corretto, potrebbe essere necessario compilare il documento due volte. Questo succede perché il compilatore LaTeX salva la lista delle sezioni all'interno del file `appunti.aux`, che verrà letto solo durante la successiva compilazione.

Se si lavora **in locale**, dopo aver aggiunto o modificato il sommario conviene quindi eseguire `pdflatex` almeno due volte. Su **Overleaf**, invece, il sistema di compilazione esegue normalmente in automatico i passaggi aggiuntivi necessari per aggiornare indice e riferimenti; se il risultato dovesse restare non aggiornato, è sufficiente ricompilare il progetto, eventualmente usando una ricompilazione completa dai file temporanei.

1.3 Riferimenti nel testo

Una volta create le sezioni, ci si potrebbe trovare nel caso in cui sia necessario riferirsi ad una sezione precedente, tramite il suo numero. Ad esempio, in una frase del tipo: "Come visto nella Sezione 2.1". È immediatamente chiaro che scrivere direttamente il numero della sezione non è una soluzione robusta a questo problema: nel caso aggiungessimo un'altra sezione precedente dovremmo andare a cercare e sostituire tutti i riferimenti precedentemente inseriti.

LaTeX propone la seguente soluzione: ad ogni elemento citabile si assegna un'etichetta tramite il comando `\label`, e poi si utilizza il comando `\ref` per citarla. Ad esempio:

```
\section{Introduzione} \label{sec:intro}
```

```
[...]

Come abbiamo visto nella Sezione~\ref{sec:intro}, [...]
```

Qualche osservazione:

- Il label viene assegnato all'ultimo comando che ha creato un elemento "citabile". Per il momento abbiamo visto solamente le sezioni, ma altri esempi sono le equazioni, tabelle, figure, ecc. Volendo, persino gli elementi di una lista numerata!
- È importante che `\label` segua il comando a cui si vuole assegnare l'etichetta: il LaTeX processa il documento dall'inizio alla fine, e dunque può etichettare solo l'output di comandi precedenti.
- È buona norma, come nell'esempio sopra, separare il nome dell'oggetto citato (nel nostro caso "Sezione") e il numero ottenuto dal comando `\ref` con il simbolo `~`. Questo rappresenta uno "spazio indistruttibile", che il LaTeX non potrà mai separare. Possiamo quindi garantire che non avremo nel testo una nuova linea fra "Sezione" e "2.1".
- Il nome del label può essere una qualunque stringa di testo. In questo caso abbiamo scelto di utilizzare il prefisso `sec:` per sottolineare che si tratta di una sezione, ma non è obbligatorio.

Come per l'indice, dopo aver aggiunto o modificato un riferimento può essere necessaria più di una compilazione: in locale si rilancia `pdflatex`, mentre Overleaf gestisce normalmente in automatico le compilazioni successive necessarie a risolvere i riferimenti.

1.4 Sistema di pacchetti

Una delle grosse novità del LaTeX rispetto al TeX è la possibilità di ottenere funzionalità aggiuntive tramite un sistema di pacchetti ben organizzato. Sebbene anche in TeX sia possibile caricare estensioni aggiuntive (in un certo senso il LaTeX stesso lo è), i pacchetti LaTeX sono organizzati in modo coerente e disponibili all'interno della distribuzione standard.

Ad esempio, possiamo caricare i pacchetti dell'AMS (American Mathematical Society) utilizzando il comando

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

Il precedente comando carica quattro pacchetti, con i nomi riportati nella lista sopra. Questi contengono varie funzionalità, come ad esempio il carattere per indicare i campi o anelli (ad es., $\mathbb{C}$ per i complessi o $\mathbb{R}$ per i reali).

Il comando `\usepackage` va utilizzato nel preambolo, ovvero prima del comando `\begin{document}`.

Il comando da scrivere nel sorgente è lo stesso sia in locale sia su Overleaf. La differenza è che, lavorando sulla propria macchina, il pacchetto deve essere presente nella distribuzione TeX installata; su `https://overleaf.unipi.it` i pacchetti

disponibili sono quelli installati sul server e vengono caricati automaticamente durante la compilazione.

1.5 Lettere accentate e caratteri speciali

Dovendo scrivere in italiano, vi capiterà spesso di dover inserire caratteri accentati. Noterete quasi subito che inserirli direttamente nel testo però non ha nessun effetto: questo caratteri non appaiono nel documento finale!

Per ovviare a questo problema ci sono (almeno) due soluzioni:

1. La prima consiste nell'inserire le lettere accentate utilizzando dei comandi appositi. Ad esempio, per scrivere è possiamo utilizzare il comando `\‘e`, mentre per inserire è utilizziamo `\‘e`. Il comando `\’` significa "metti l'accento alla prossima lettera".

2. Possiamo caricare un pacchetto apposito, per "insegnare" a LaTeX a leggere questi caratteri speciali. Nel nostro caso, questo va fatto con il comando `\usepackage[utf8x]{inputenc}`.

Storicamente, la prima strategia è stata sempre considerata preferibile, perché nello scambio di file non richiede che mittente e destinatario siano in grado di interpretare caratteri speciali. Tuttavia, nei sistemi operativi moderni questo è sempre verificato, e dunque anche la soluzione 2. può essere considerata senza troppi problemi. Notiamo però che la soluzione 1. ci permette di inserire anche lettere accentate con accenti che non abbiamo sulla tastiera. Ad esempio, se volessi scrivere ä potrei digitare `\"a`, oppure se volessi scrivere ŏ potrei inserire `\v{o}`, e così via. Attenzione, non sempre questi caratteri vengono riconosciuti nel modo corretto quando si effettua una ricerca nel testo!

Un altro pacchetto rilevante nel caso si voglia scrivere in italiano è il pacchetto `babel`, che traduce la maggior parte dei termini in italiano (ad esempio, il comando `\tableofcontents` produrrà un "Indice", invece di una sezione "Contents"); in aggiunta, implementa anche la sillabazione corretta che permette a LaTeX di spezzare correttamente le parole quando deve andare a capo.

2 Scrivere equazioni

La ragione principale per cui il LaTeX è così diffuso nell'ambiente matematico è la facilità con cui si possono scrivere equazioni. In effetti, questo è il motivo principale per cui il software è stato inventato.

2.1 Modi matematici

Per poter realizzare equazioni bisogna passare dal modo testuale normale al modo matematico, in cui il testo che scriviamo viene interpretato in modo diverso. Ci sono tre diversi modi matematici che si possono usare:

1. Il modo in linea, che permette di inserire matematica all'interno del testo, e nel quale si entra e si esce utilizzando il singolo `$`.

2. Il modo display, nel quale le equazioni vengono centrate all'interno della pagina. Si può entrare in questo modo con il comando `\[`, ed uscire con `\]`. In alternativa è possibile rimpiazzare questi due comandi con il doppio dollaro `$$`, che però può produrre spaziature meno precise.

3. L'ambiente `equation`, quasi equivalente alla modalità display, ma nel quale ad ogni equazione viene assegnato un numero. Invece che delimitare l'equazione con `\[` e `\]`, si utilizzano i comandi `\begin{equation}` ed `\end{equation}`. Si può utilizzare `\label` (ad esempio subito dopo `\begin{equation}`, in ogni caso all'interno dell'ambiente) per assegnare un'etichetta all'equazione, da citarsi nel seguito con `\eqref`; attenzione, per quest'ultimo comando serve includere i pacchetti `amsmath`.

Consideriamo un primo esempio di matematica, che contiene un misto di equazioni in linea ed equazioni centrate.

Il prodotto dei polinomi $x^2 + x + 1$ e $x - 1$ è dato da:

$$(x^2 + x + 1)(x - 1) = x^3 - 1.$$

Questo esempio si può ottenere con il seguente codice:

```
Il prodotto dei polinomi $x^2 + x + 1$ e $x - 1$ e' dato da:
\[
   (x^2 + x + 1) (x - 1) = x^3 - 1.
\]
```

Osserviamo che viene utilizzata la sintassi `x^k` per scrivere un esponente. In maniera simile, potremmo utilizzare `x_k` per ottenere il pedice x_k.

Nel caso sia necessario scrivere esponenti e/o pedici (o in generale, argomenti di comandi) composti da più caratteri, sarà necessario utilizzare le parentesi graffe per raggrupparli. Ad esempio, supponiamo di voler ottenere il seguente testo:

Le radici dell'unità, che risolvono $x^n - 1 = 0$, possono essere descritte da ξ_n^j, dove:

$$\xi_n = e^{\frac{2\pi i}{n}},$$

e il simbolo i denota l'unità immaginaria, ovvero $i^2 = -1$.

Possiamo utilizzare i seguenti comandi:

```
Le radici dell'unita', che risolvono $x^n - 1 = 0$,
possono essere descritte da $\xi_n^j$, dove:
\[
   \xi_n = e^{\frac{2 \pi i}{n}},
\]
e il simbolo $i$ denota l'unita' immaginaria, ovvero $i^2 = -1$.
```

Nell'esempio precedente possiamo imparare molti nuovi comandi. Ad esempio, `\xi` produce la lettera greca ξ. In maniera analoga, sono disponibili `\alpha`, `\beta`, `\gamma`, ...

Il comando `\frac` produce una frazione tipo $\frac{a}{b}$. Prende due argomenti, racchiusi fra graffe se composti da più di un carattere, che sono rispettivamente numeratore e denominatore.

Supponiamo ora di voler essere più precisi, e dire che le radici ξ_n^j sono quelle di $x^n - 1$ sul campo $\mathbb{C}$. Possiamo ottenere il simbolo che usualmente denota i complessi con il comando `\mathbb C`. Il comando `\mathbb`, che permette di utilizzare il font denominato bb, si utilizza solitamente per anelli e campi, come $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$, $\mathbb{F}_p$, ecc. Questo comando non è disponibile nel LaTeX standard, e per poterlo utilizzare dovremo caricare un pacchetto ausiliari, che contiene le definizioni di nuovi comandi. In questo caso specifico, si tratta del pacchetto `amsfonts`, dove ams sta per American Mathematical Society. Per caricare il pacchetto, dobbiamo inserire il comando `\usepackage{amsfonts}` nel preambolo del documento (prima del comando `\begin{document}`), come spiegato nella sezione precedente dedicata al sistema di pacchetti.

Esercizio 5

Modificare l'esempio di sopra inserendo la nota che le radici stanno all'interno del campo $\mathbb{C}$. Se preferite, invece che scriverlo come testo, potete anche utilizzare la notazione $x \in \mathbb{C}$, utilizzando con il comando `\in`.

I comandi matematici sono moltissimi. Fra i più utilizzati possiamo considerare quelli per creare il simbolo di integrale. Qualche esempio per impararne qualcuno:

$$f(z) = \frac{1}{2\pi i} \int_\Gamma \frac{f(w)}{w - z}\, dw$$

```
\[
  f(z) = \frac{1}{2\pi i} \int_\Gamma \frac{f(w)}{w - z}\ dw
\]
```

$$\int_a^b f(x)\, dx = F(b) - F(a).$$

```
\[
  \int_a^b f(x)\ dx = F(b) - F(a).
\]
```

$$\sum_{j=1}^{\infty} \frac{1}{j} = \infty, \qquad \sum_{j=1}^{n} j = \frac{j(j+1)}{2}.$$

```
\[
  \sum_{j = 1}^\infty \frac{1}{j} = \infty, \qquad
  \sum_{j = 1}^n j = \frac{j(j+1)}{2}.
\]
```

$$\frac{\partial}{\partial x}\sqrt{xy} = \frac{\sqrt{y}}{2\sqrt{x}}$$

```
\[
  \frac{\partial}{\partial x} \sqrt{xy} =
  \frac{\sqrt{y}}{2 \sqrt{x}}
\]
```

$$7 \equiv 1 \mod 3$$

```
\[
  7 \equiv 1 \ \mathrm{mod}\ 3
\]
```

Alcuni comandi sono abbastanza autoesplicativi. Notiamo solo che `\ ` e `\qquad` possono essere utilizzati per produrre spazi piccoli e grandi all'interno di equazioni. Se questi non vengono utilizzati, gli spazi saranno completamente ignorati.

Il comando `\mathrm`, invece, permette di utilizzare il font roman all'interno di un'equazione, e può essere utile per inserire nome di funzioni / operatori (come per mod sopra). Alcune funzioni dispongono già di un comando con il loro nome, come ad esempio quelle trigonometriche (si provino i comandi `\sin`, `\cos`, ...).

2.2 Matrici e vettori

Come possiamo inserire una matrice? Anche per questa domanda non esiste una risposta univoca. Partiamo dalla soluzione più semplice, che richiede di utilizzare il pacchetto `amsmath`. Assicuratevi dunque di avere il comando

```
\usepackage{amsmath}
```

nel preambolo. A questo punto, potete utilizzare gli ambienti `pmatrix` e/o `bmatrix` per inserire delle matrici, come nel seguente esempio:

```
A = \begin{bmatrix}
  1 & 2 & 3 \\
  4 & 5 & 6 \\
\end{bmatrix} = \begin{pmatrix}
  1 & 2 & 3 \\
  4 & 5 & 6 \\
\end{pmatrix}.
```

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}.$$

Gli stessi comandi si possono usare per vettori colonna (come matrici $m \times 1$) e vettori riga (matrici $1 \times n$). Il comando `&` separa le varie colonne, e `\\` inizia una nuova riga.

Potrebbe capitare di volere un controllo più fine sul risultato finale della matrice che stiamo inserendo, ad esempio, come inserire una matrice di questo tipo?

$$A := \left[\begin{array}{cc|c} 1 & 2 & 3 \\ 4 & 5 & 6 \end{array}\right]$$

Il primo passo è fare uso dell'ambiente `array`, che permette di disporre del contenuto in forma di tabella. Richiede un argomento aggiuntivo che gli dica quante colonne ci sono, e come deve venire allineato il contenuto al loro interno. Noi scegliamo 3 colonne centrate, identificate da `ccc`:

```
\begin{array}{ccc}
  1 & 2 & 3 \\
  4 & 5 & 6 \\
\end{array}
```

$$\begin{array}{ccc} 1 & 2 & 3 \\ 4 & 5 & 6 \end{array}$$

Ora, modifichiamo questo esempio per inserire la riga verticale che separa la seconda e la terza colonna. Per farlo, basta modificare l'indicazione delle colonne con `cc|c`.

```
\begin{array}{cc|c}
1 & 2 & 3 \\
4 & 5 & 6 \\
\end{array}
```

$$\begin{array}{cc|c} 1 & 2 & 3 \\ 4 & 5 & 6 \end{array}$$

Adesso non ci rimane che racchiudere questo array fra parentesi quadre. Se mettessimo semplicemente i caratteri `[` e `]` a destra e sinistra questi sarebbero troppo piccoli. Possiamo usare i comandi `\left` e `\right` per generare dei delimitatori che si ingrandiscano abbastanza da racchiudere tutto quello che sta fra di loro:

```
\left[ \begin{array}{cc|c}
1 & 2 & 3 \\
4 & 5 & 6 \\
\end{array} \right]
```

$$\left[\begin{array}{cc|c} 1 & 2 & 3 \\ 4 & 5 & 6 \end{array}\right]$$

Ecco il nostro esempio completo. Osserviamo che non c'è bisogno che i delimitatori per `\left` e `\right` siano coerenti. In particolare, è possibile specificare un delimitatore invisibile scrivendo `\left.` oppure `\right.`; questo può tornare utile nel caso si voglia definire una funzione in modo diverso su diverse parti del dominio:

```
f(x) = \left\{
  \begin{array}{cc}
    1 & x = 0 \\
    \frac{\sin(x)}{x} & x \neq 0
  \end{array}
\right.
```

$$f(x) = \begin{cases} 1 & x = 0 \\ \frac{\sin x}{x} & x \neq 0 \end{cases}$$

In realtà, per questo caso particolare il pacchetto `amsmath` fornisce già l'ambiente `cases` (provatelo!) che permette di ottenere un risultato migliore.

Esercizio 6

Si provi a realizzare una pagina partendo da un modello, senza seguire una procedura dettagliata. Si scarichi il file `pagina-test.pdf` da Moodle, e si provi a replicarlo nel modo più fedele possibile utilizzando LaTeX.

2.3 Teoremi ed enunciati

Esercizio 7

Scegliete un teorema a piacere, e riportartelo nel vostro file `appunti.tex`. Inserite nel preambolo il comando `\newtheorem{theorem}{Theorem}`, che creerà un nuovo ambiente chiamato `theorem`, e poi racchiudete il vostro teorema dentro uno di questi:

```
\begin{theorem}
Sia ... . Allora ...
\end{theorem}
```

Nella maggior parte dei documenti matematici si trovano svariati enunciati, come teoremi, proposizioni, lemmi, ecc., ma anche definizioni, osservazioni, ecc. Il pacchetto `amsthm`, che abbiamo caricato in precedenza, ci permette di strutturarli con una numerazione all'interno del documento. In particolare, potremo usare anche i comandi `\label` e `\ref` per inserire riferimenti in punti successivi del testo.

Per poter utilizzare questi ambienti è necessario definirli nel preambolo, ovviamente dopo aver caricato il pacchetto `amsthm`. La sintassi da utilizzare è la seguente:

```
\newtheorem{theorem}{Theorem}[section]
\newtheorem{lemma}[theorem]{Lemma}
\newtheorem{proposition}[theorem]{Proposition}
```

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

\theoremstyle{definition}
\newtheorem{definition}[theorem]{Definition}
```

Soffermiamoci ora sui vari comandi dati sopra:

`\newtheorem{theorem}{Theorem}[section]` indica di creare un nuovo ambiente di nome `theorem`, che creerà degli enunciati con nome "Theorem". La numerazione di questi enunciati seguirà la numerazione delle sezioni. Ad esempio, nella Sezione 2 avremo Theorem 2.1, Theorem 2.2, ecc. Se il parametro opzionale non viene dato, allora verrà utilizzata una numerazione incrementale con 1, 2, ...

`\newtheorem{lemma}[theorem]{Lemma}` indica invece di creare un nuovo ambiente `lemma`, in maniera simile a `theorem`. In questo caso però, il parametro opzionale cambia posizione e dice a questo nuovo ambiente di condividere la numerazione con l'ambiente `theorem` creato in precedenza. In questo modo, se aggiungiamo 2 teoremi con un lemma intermedio questi si chiameranno Theorem 2.1, Lemma 2.2, e Theorem 2.3. Senza quest'accortezza, tutti questi oggetti avrebbero numerazioni indipendenti.

`\theoremstyle` serve a cambiare stile degli ambienti teoremi. La scelta di default è `theorem`, che utilizza il testo in italico, e sono disponibili anche `definition` e `remark`. Il loro comportamento esatto dipende anche dal tipo di documento che si sta compilando.

Esercizio 8

Si seguano le istruzioni sopra aggiungendo almeno un Lemma agli appunti, condividendo la numerazione con il teorema che aggiunto prima. Si inserisca poi una dimostrazione utilizzando i comandi `\begin{proof}` ed `\end{proof}`.

---

3 Grafici, immagini e tabelle

LaTeX gestisce in modo particolare l'inserimento di elementi come grafici e tabelle. In generale, esiste un ambiente utilizzato per creare gli elementi da inserire (ad esempio, `tabular` per le tabelle), ed un altro per posizionarli nella pagina con una didascalia ed i riferimenti corretti (nel caso delle tabelle, l'ambiente `table`).

Questo può causare un po' di frustrazione iniziale perché LaTeX è dotato di un algoritmo per posizionare gli elementi nella pagina in modo "ottimale", secondo opportune regole tipografiche, ed è molto difficile da condizionare. Sebbene sia teoricamente possibile specificare dei suggerimenti per il posizionamento, come vedremo in seguito, questi vengono tenuti molto poco di conto.

3.1 Tabular e table

Per inserire una tabella possiamo usare l'ambiente `tabular`. Si può notare che ci sono molte similitudini con l'ambiente `array` e quelli per inserire matrici, come ad esempio le parole chiave per l'allineamento all'interno delle celle e il carattere `&` per separare le colonne. Ad esempio, il seguente codice:

```
\begin{tabular}{rc|c}
  Colonna 1 & Colonna 2 & Colonna 3 \\ \hline
    1 & $x - y$ & oggi \\
    2 & $e^x$   & domani
\end{tabular}
```

genera il seguente output:

Colonna 1	Colonna 2	Colonna 3
1	$x - y$	oggi
2	e^x	domani

Si può osservare in particolare:

- La specifica delle colonne viene effettuata con la stringa `rcc`, che significa che la prima colonna dev'essere allineata a destra, la seconda e la terza al centro. È possibile specificare anche l'allineamento a sinistra (con la keyword `l`), oppure colonne di larghezza fissata.
- Il comando `\hline` può essere utilizzato per generare una riga divisoria, ad esempio dopo l'intestazione. È possibile utilizzare una riga doppia ripetendolo due volte.
- È possibile utilizzare equazioni all'interno delle tabelle, racchiudendolo in opportuni dollari come nel resto del documento.

Nella maggior parte dei casi è necessario inserire tabelle a cui fare riferimento nel testo, ad esempio con espressione del tipo "Come visto in Tabella 2 . . . ". Per ottenere questo risultato, si preferisce includere l'ambiente `tabular` all'interno di un ambiente più esterno `table`, che può contenere anche una didascalia (tramite il comando `\caption`) e un `\label`. La sintassi per creare una `table` è la seguente:

```
\begin{table}
  \begin{tabular}{...}
    [...]
  \end{tabular}
  \caption{La mia tabella}
  \label{tab:mia}
\end{table}
```

È possibile inserire riferimenti a una tale tabelle all'interno del documento utilizzando il comando `\ref{tab:mia}`, esattamente come fatto per le sezioni.

Come in quel caso, è preferibile inserire uno spazio non interrompibile fra la parola "Tabella" e il riferimento, in modo che LaTeX non vada a capo. Ad esempio, si può scrivere `Tabella~\ref{tab:mia}`.

Quando viene creato un ambiente `table`, non ci sono garanzie che appaia nel punto in cui è stato inserito il codice corrispondente. Anzi, questo succederà molto di rado. LaTeX tenterà di posizionarlo nel posto "migliore possibile". È possibile, tuttavia, specificare una propria preferenza, con un parametro opzionale dell'ambiente `table`, da inserire fra parentesi quadre.

Ad esempio, il comando

```
\begin{table}[h]
 ...
\end{table}
```

genera una tabella chiedendo a LaTeX di inserirla dove è stato scritto il codice, o comunque non troppo lontano. In maniera simile, è possibile utilizzare la lettera `t` per dire che si preferisce averla in cima alla pagina, oppure `b` per il fondo della pagina. Si può utilizzare `h!` per insistere sul fatto che la tabella dovrebbe essere inserita nella posizione corrente. In ogni caso, LaTeX avrà sempre l'ultima parola in proposito.

3.2 Includere immagini esterne

LaTeX è abbastanza pignolo sui formati delle immagini esterne da inserire. Quelli supportati da `pdflatex`, che è il compilatore utilizzato fino ad ora, sono:

- PNG, per la grafica non vettoriale. Le immagini in questa categoria includono le foto scattate con una macchina fotografica, che sono rappresentate sul computer come una matrice di punti colorati (detti pixel).
- PDF, per la grafica vettoriale. Queste immagini, al contrario delle precedenti, sono composte da una loro descrizione geometrica. Questo implica che è possibile scalarle senza alcuna perdita di qualità.

Per inserire una qualsiasi immagine nel documento, è possibile utilizzare il comando

```
\includegraphics{immagine}
```

che caricherà automaticamente un file `immagine.pdf` oppure `immagine.png`. Attenzione, perché il comando sia disponibile è necessario includere il pacchetto `graphicx`, tramite il comando `\usepackage{graphicx}`.

In maniera simile a quanto succede per la tabelle, anche le immagini vengono solitamente posizionate all'interno di un riquadro con una didascalia e un label per riferirsi ad esse in seguito. Questo si ottiene con l'ambiente `figure`, da usare nel seguente modo:

```
\begin{figure}
  \includegraphics{immagine}
```

```
  \caption{Una mia fotografia}
  \label{fig:miafoto}
\end{figure}
```

Esercizio 9

Si scelga una fotografia e la si inserisca nel documento, posizionandola all'interno di una ambiente `figure`. Se si lavora in locale, il file dell'immagine deve trovarsi nella cartella del progetto (o in un percorso relativo indicato nel comando); su Overleaf, invece, l'immagine va caricata tra i file del progetto mantenendo coerente il percorso usato in `\includegraphics`.

Le stesse considerazioni fatte per l'ambiente `table` sul posizionamento valgono anche per l'ambiente `figure`.

3.3 Grafici di funzioni e dati

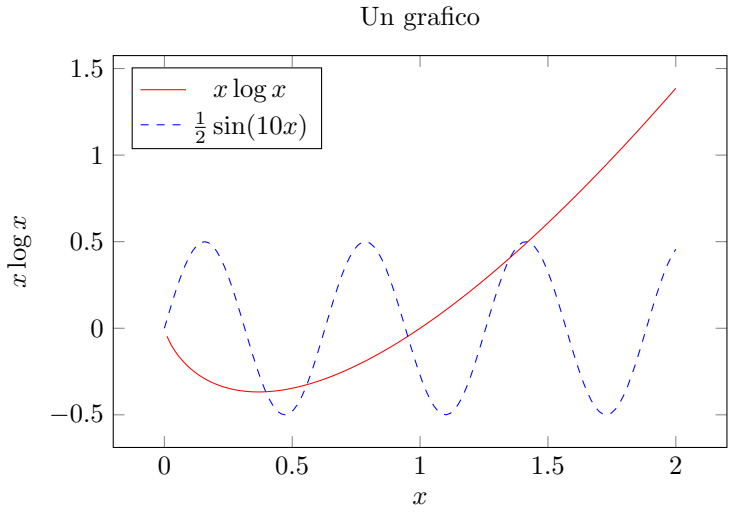
Consideriamo ora un altro task piuttosto comune. Supponiamo di voler produrre un grafico di una funzione, oppure di alcuni dati. Ad esempio, potremmo voler fare un grafico di $f(x) = x \log x$ su $[0, 2]$, oppure plottare dei dati calcolati da un altro programma.

Per questo scopo, è possibile utilizzare il pacchetto `pgfplots`. Come gli altri pacchetti, va incluso con il comando `\usepackage{tikz,pgfplots}`. Questo comando carica il pacchetto `tikz`, che si può utilizzare per produrre grafica di vario tipo, e il pacchetto `pgfplots` che è basato su TikZ e permette di realizzare facilmente grafici di funzione.

Ad esempio, si provi ad inserire il seguente codice nel documento:

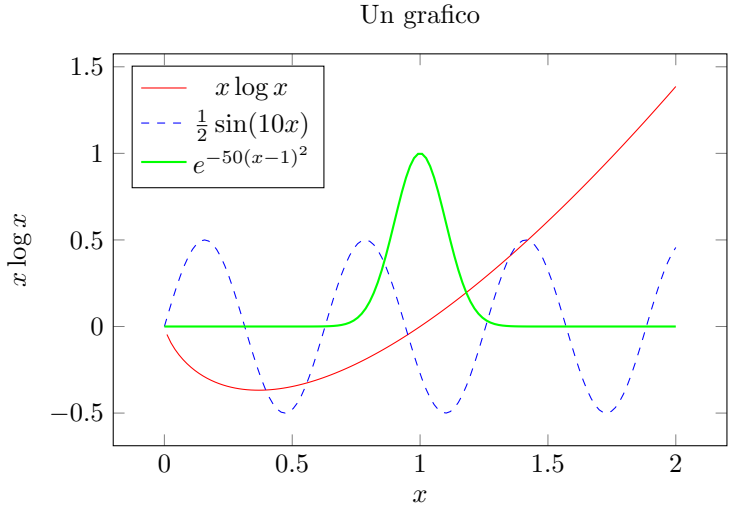
```
\begin{tikzpicture}
\begin{axis}[xlabel = $x$, ylabel = $x \log x$,
             title = {Un grafico}, legend pos = north west,
             width = .8\linewidth, height = .35\textheight]
  \addplot[domain = 0:2, red, samples = 128] { x * ln(x)};
  \addplot[domain = 0:2, blue, dashed, samples = 128]
    {sin(deg(x)*10)/2};
  \legend{$x \log x$, $\frac{1}{2}\sin(10x)$};
\end{axis}
\end{tikzpicture}
```

L'output ottenuto dovrebbe essere simile al seguente:



Esercizio 10

Si provi a modificare il codice sopra per plottare anche il grafico della funzione $e^{-50(x-1)^2}$ (si utilizzi `exp` per indicare l'esponenziale). Utilizzare un tratto molto marcato verde (ottenibile con l'opzione `thick`). Il risultato dovrebbe essere il seguente:



Nel caso si avessero a disposizione dei dati generati da un programma, potremmo volerli plottare come punti. Ad esempio, consideriamo un file `punti.dat` ottenuto calcolando per dei punti fra 0 ed 1 il loro quadrato; potete crearlo

copiando i seguenti valori. Se lavorate in locale, salvate `punti.dat` nella stessa cartella del sorgente; su Overleaf create un nuovo file con questo nome oppure caricatelo nel progetto.

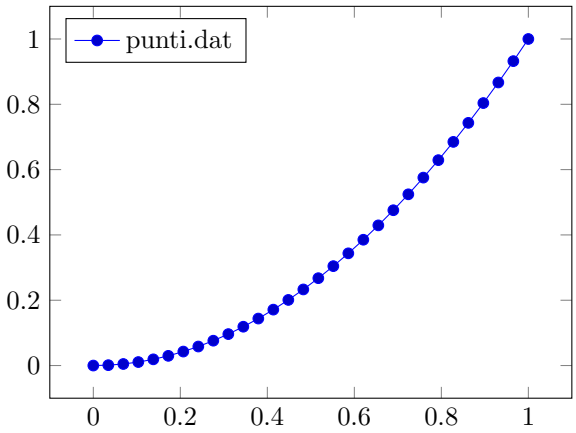
```
0.00000     0.00000
0.03448     0.00119
0.06897     0.00476
0.10345     0.01070
0.13793     0.01902
0.17241     0.02973
0.20690     0.04281
0.24138     0.05826
0.27586     0.07610
0.31034     0.09631
0.34483     0.11891
0.37931     0.14388
0.41379     0.17122
0.44828     0.20095
0.48276     0.23306
0.51724     0.26754
0.55172     0.30440
0.58621     0.34364
0.62069     0.38526
0.65517     0.42925
0.68966     0.47562
0.72414     0.52438
0.75862     0.57551
0.79310     0.62901
0.82759     0.68490
0.86207     0.74316
0.89655     0.80380
0.93103     0.86683
0.96552     0.93222
1.00000     1.00000
```

Esercizio 11

Utilizzare il comando, da inserire all'interno di un ambiente `axis`,

```
\addplot table {punti.dat};
```

per ottenere un plot dei punti contenuti nel file `punti.dat` appena creato. Il risultato dovrebbe essere simile al seguente:



Si cerchi in rete il manuale di `pgfplots`, e si modifichi il codice in modo da cambiare il tipo di punti da pallini a triangoli.

Commento: Alcuni programmi che possono essere d'aiuto per generare questo tipo di dati sono MATLAB, GNU/Octave, Maple, oppure Mathematica o Maxima. Fra questi Octave e Maxima sono open source e gratuiti, e di MATLAB esiste una licenza d'ateneo. Una versione vetusta ma funzionante di Maple è disponibile sulle macchine dell'Aula 4. Alcuni di questi software possono anche produrre direttamente dei grafici da poi importare con `\includegraphics`, ma che spesso hanno una più bassa qualità tipografica ed integrazione in LaTeX.

4 Definire nuovi comandi

Spesso ci si trova a dover usare un simbolo, o qualche funzionalità evoluta di LaTeX ripetutamente all'interno di un documento. Supponiamo ad esempio di voler definire un operatore:

$$\begin{aligned} \mathfrak{R}: \quad [0,1] &\to [0,1] \\ x &\mapsto \sqrt{2}\cdot x \bmod 1, \end{aligned}$$

ma di non essere ancora sicuri del simbolo da usare per denotarlo. Vorremmo evitare di trovarci più avanti a dover sostituire all'interno di tutto il documento la notazione usata precedentemente.

Per ovviare a quetso inconveniente, si può utilizzare il comando LaTeX `\newcommand`, che permette di definire nuovo comando. Ad esempio, nel caso precedentemente avremmo potuto inserire nel preambolo:

```
% Operatore R(x)
\newcommand{\myop}{\mathfrak{R}}
```

Questo ci permette di scrivere l'operatore precedente semplicemente utilizzando il comando `\myop(x)`. Se nel futuro cambiassimo idea, e volessimo denotare questo

operatore con, ad esempio, $\Gamma(x)$, sarebbe sufficiente cambiare la definizione del comando nel preambolo. È possibile anche definire comandi che accettino un certo numero di parametri. Ad esempio,

```
\newcommand{\mycommand}[2]{Ciao #1, io mi chiamo #2}
```

si può chiamare con la sintassi `\mycommand{Giovanni}{Piero}`, e produrrebbe l'output "Ciao Giovanni, io mi chiamo Piero".

Esercizio 12

Si scriva un comando `\norm` che accetti due parametri: un vettore x ed un intero p, e che produca il simbolo della norma p del vettore x:

`\norm{x}{2}` $\rightsquigarrow$ $\|x\|_2$.

Si possono usare i comandi `\lVert` e `\rVert` per produrre le stanghettte verticali (rispettivamente a sinistra e a destra del singolo). Si modifichi poi il comando per adattarsi all'altezza del contenuto, e testarlo sul seguente esempio:

```
\norm{
  \begin{bmatrix}
     x_1 \\ x_2
  \end{bmatrix}
}{2}
```

$$\left\| \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \right\|_2$$

---

5 Utilizzare la bibliografia

Quando si realizza un documento come un articolo scientifico, oppure una tesi di laurea, è necessario includere nel documento i riferimenti bibliografici che si sono utilizzati. Le distribuzioni LaTeX includono un software, chiamato BibTeX, che permette di realizzarla con poco sforzo.

L'uso di BibTeX si organizza in questo modo:

- Si crea un file che contiene il database della bibliografia da utilizzare, con estensione `.bib`; ad esempio, potremmo creare un file `biblio.bib`.
- In questo file, si inseriscono i riferimenti da utilizzare. Questi vanno scritti in un formato particolare compreso da BibTeX, di cui forniremo dettagli in seguito. Ogni elemento bibliografico è identificato da un nome univico.
- Nel documento, si citano i lavori con il comando `\cite{ID}`, dove ID è l'identificatore menzionato sopra.

- Alla fine del documento, si sceglie lo stile da usare per la nostra bibliografia, e si specifica il nome del file "database" da utilizzare. Ad esempio:

  ```
  \bibliographystyle{plain}
  \bibliography{biblio}
  ```

  Lo stile `plain` è abbstanza comune, ma ne esistono molti altri disponibile (provare ad esempio `alpha`).

Gli elementi da inserire all'interno del file `biblio.bib` (che è un semplice file di testo, hanno il seguente formato:

```
@book{knuth1997art,
  title={The art of computer programming},
  author={Knuth, Donald Ervin},
  volume={3},
  year={1997},
  publisher={Pearson Education}
}
```

Nell'esempi sopra, per citare il lavoro di Knuth, potremmo usare nel documento LaTeX il comando `\cite{knuth1997art}`. Gli elementi da inserire nel database non si scrivono (usualmente) a mano, ma si possono scaricare da alcuni portali. Uno di questi è offerto dell'AMS (American Mathematica Society) e si chiama MathSciNet. Sfortunatamente, è accessibile solamente dai computer collegati alla rete dell'Ateneo. In alternativa, è possibile utilizzare il sito web `https://scholar.google.com/`. Su quest'ultimo è possibile cercare libri e articoli scientifico, e citarli cliccando sul simbolo delle virgolette, che permette di scaricare la bibliografia anche in formato BibTeX.

Per utilizzare BibTeX, la procedura dipende da dove si compila il documento. Supponiamo che il file principale si chiami `miofile.tex`.

- **Compilazione sulla propria macchina.** Da terminale si eseguono, nell'ordine,

  ```
  pdflatex miofile.tex
  bibtex miofile
  pdflatex miofile.tex
  pdflatex miofile.tex
  ```

  Il primo passaggio crea il file `miofile.aux`; `bibtex miofile` usa le informazioni contenute in tale file per costruire la bibliografia; le ultime due compilazioni con `pdflatex` aggiornano citazioni, numeri e riferimenti incrociati.

- **Compilazione su Overleaf.** Su `https://overleaf.unipi.it` è sufficiente avere nel progetto il file `.bib`, i comandi `\bibliographystyle` e

`\bibliography` nel sorgente, e premere *Recompile*. Il sistema di compilazione esegue normalmente in automatico BibTeX e le successive compilazioni di LaTeX quando sono necessarie. Se citazioni o riferimenti restano non aggiornati, si può forzare una ricompilazione completa del progetto.

Esercizio 13

Si inserisca una bibliografia, e si citino il lavoro di Knuth riportato sopra e il LaTeX Companion (libro da trovarsi su Google Scholar).

---

6 Scrivere algoritmi e codice

Nei documenti di carattere scientifico può essere utile inserire frammenti di codice sorgente, ad esempio per descrivere un algoritmo, mostrare l'implementazione di un metodo numerico oppure documentare il funzionamento di un programma.

Naturalmente sarebbe possibile utilizzare un ambiente `verbatim`, che riproduce il testo senza interpretarlo come codice LaTeX. Questa soluzione, però, offre un controllo molto limitato sull'aspetto del codice.

Il pacchetto `listings` mette a disposizione strumenti specifici per la visualizzazione di codice sorgente. Per utilizzarlo è sufficiente aggiungere nel preambolo

```
\usepackage{listings}
```

Il principale ambiente fornito dal pacchetto è `lstlisting`. Ad esempio:

```
\begin{lstlisting}[language=Python]
def quadrato(x):
    return x*x

print(quadrato(3))
\end{lstlisting}
```

Il parametro opzionale `language=Python` dice a `listings` quale linguaggio di programmazione stiamo utilizzando. In questo modo il pacchetto può riconoscere parole chiave, commenti, stringhe ed altri elementi del linguaggio.

Sono supportati numerosi linguaggi, fra cui C, C++, Python, Java, Matlab e molti altri.

Esercizio 14

Inserite nel vostro documento il seguente programma Python utilizzando l'ambiente `lstlisting`:

```
1 def somma(n):
2     s = 0
3     for k in range(1, n+1):
4         s = s + k
5     return s
6
7 print(somma(100))
```

Provate quindi a rimuovere l'opzione

`language=Python`

e confrontate il risultato.

Quali informazioni utilizza `listings` per distinguere, ad esempio, una parola chiave come `for` da un normale nome di variabile?

6.1 Personalizzare la visualizzazione del codice

Il comportamento dell'ambiente `lstlisting` può essere modificato tramite numerose opzioni. Ad esempio,

```
\begin{lstlisting}[
  language=Python,
  numbers=left,
  frame=single,
  breaklines=true
]
...
\end{lstlisting}
```

produce un riquadro attorno al codice, aggiunge i numeri di riga sulla sinistra e permette di spezzare automaticamente le righe troppo lunghe.

Fra le opzioni più comuni troviamo:

`language` specifica il linguaggio utilizzato;

`numbers=left` inserisce i numeri di riga sulla sinistra;

`numbers=right` inserisce i numeri di riga sulla destra;

`frame=single` disegna un riquadro attorno al codice;

`breaklines=true` permette di spezzare le righe troppo lunghe;

`showstringspaces=false` evita di evidenziare gli spazi all'interno delle stringhe;

`basicstyle` permette di scegliere il carattere utilizzato per il codice.

Ad esempio,

`basicstyle=\ttfamily\small`

utilizza un carattere monospaziato di dimensione leggermente inferiore a quella del testo normale.

Esercizio 15

Scrivete con `listings` il seguente programma in C:

```
#include <stdio.h>

int main(void)
{
    int n = 10;
    int somma = 0;

    for (int k = 1; k <= n; ++k)
        somma += k;

    printf("%d\n", somma);
    return 0;
}
```

Modificate poi le opzioni dell'ambiente in modo che:

1. vengano visualizzati i numeri di riga;
2. il codice sia racchiuso in un riquadro;
3. le righe troppo lunghe possano essere spezzate automaticamente;
4. gli spazi nelle stringhe non siano evidenziati.

6.2 Definire uno stile

Se dobbiamo inserire molti frammenti di codice non è conveniente ripetere ogni volta le stesse opzioni. Possiamo definire uno stile nel preambolo:

```
\lstdefinestyle{codice}{
  basicstyle=\ttfamily\small,
  numbers=left,
  numberstyle=\tiny,
  numbersep=8pt,
  frame=single,
  breaklines=true,
  showstringspaces=false,
  columns=fullflexible
}
```

Lo stile può essere utilizzato scrivendo

```
\begin{lstlisting}[style=codice,language=Python]
...
\end{lstlisting}
```

Se vogliamo utilizzarlo come stile predefinito per tutto il documento possiamo invece scrivere nel preambolo

```
\lstset{style=codice}
```

In questo caso non sarà più necessario specificare `style=codice` per ogni singolo frammento.

Esercizio 16

Definite nel preambolo uno stile chiamato `codice` e fate in modo che tutti i successivi ambienti `lstlisting` lo utilizzino automaticamente.

Inserite quindi nel documento almeno due frammenti di codice scritti in linguaggi diversi, ad esempio uno in Python ed uno in C.

Le caratteristiche grafiche dei due frammenti dovranno essere le stesse, mentre il linguaggio dovrà essere specificato separatamente.

6.3 Descrivere un algoritmo

Possiamo ora utilizzare `listings` per presentare l'implementazione di un semplice algoritmo.

Consideriamo, ad esempio, l'algoritmo di Euclide per il calcolo del massimo comun divisore di due numeri interi. Una possibile implementazione in Python è:

```
1 def mcd(a, b):
2     while b != 0:
3         r = a % b
4         a = b
5         b = r
6     return a
```

Il vantaggio di riportare il codice all'interno del documento è che possiamo descrivere matematicamente il problema e mostrare subito dopo una sua implementazione.

Esercizio 17

Scrivete un breve paragrafo che descriva l'algoritmo di Euclide e inserite successivamente la sua implementazione Python con `listings`.

Provate il procedimento a mano con

$$a = 252, \qquad b = 105$$

e riportate nel testo la sequenza dei resti ottenuti.

Aggiungete infine al programma una chiamata

```
print(mcd(252, 105))
```

e verificate che il risultato coincida con quello calcolato a mano.

6.4 Un algoritmo numerico

Consideriamo ora il problema di approssimare una soluzione dell'equazione

$$f(x) = 0.$$

Se sappiamo che f è continua nell'intervallo $[a, b]$ e che

$$f(a)f(b) < 0,$$

possiamo utilizzare il metodo di bisezione. Ad ogni iterazione si calcola

$$c = \frac{a+b}{2}$$

e si sostituisce uno dei due estremi con c, mantenendo un intervallo nel quale la funzione cambia segno.

Esercizio 18

Scrivete in Python una funzione

```
bisezione(f, a, b, toll)
```

che implementi il metodo di bisezione.

Una possibile struttura del programma è:

```
1 def bisezione(f, a, b, toll):
2     while b - a > toll:
3         c = (a + b) / 2
4 
5         # decidere quale intervallo mantenere
6 
7     return (a + b) / 2
```

Completate il codice e utilizzatelo per approssimare una radice di

$$f(x) = x^2 - 2$$

nell'intervallo $[1, 2]$.

Confrontate il risultato con $\sqrt{2}$.

Nel documento dovranno comparire sia una breve descrizione matematica del metodo sia il codice che lo implementa.

6.5 Inserire codice all'interno di una frase

Per brevi frammenti di codice non è necessario utilizzare un intero ambiente `lstlisting`. Il comando

```
\lstinline|codice|
```

permette di inserire codice direttamente nel testo.

Ad esempio, il sorgente

```
La funzione \lstinline|range(10)| genera gli interi da 0 a 9.
```

permette di scrivere il nome della funzione utilizzando lo stesso stile del codice sorgente.

Il carattere `|` utilizzato nell'esempio è semplicemente un delimitatore. Può essere sostituito da un altro carattere che non compaia nel frammento di codice.

Esercizio 19

Scrivete un paragrafo di almeno cinque righe che descriva il funzionamento di un ciclo `for` in Python.

Utilizzate `\lstinline` ogni volta che nel testo compaiono istruzioni o nomi appartenenti al linguaggio Python, come ad esempio `for`, `range` e `print`.

6.6 Didascalie e riferimenti

Gli ambienti `lstlisting` possono essere numerati e possono avere una didascalia. Ad esempio:

```
\begin{lstlisting}[
  language=Python,
  caption={Calcolo del fattoriale},
  label={lst:fattoriale}
]
def fattoriale(n):
    ...
\end{lstlisting}
```

Il codice può quindi essere citato nel testo utilizzando

```
Listato~\ref{lst:fattoriale}
```

nello stesso modo in cui abbiamo utilizzato `\label` e `\ref` per sezioni, equazioni, figure e tabelle.

Esercizio 20

Scrivete una funzione Python che calcoli il fattoriale di un intero $n \geq 0$.

Inseritela nel documento assegnandole:

1. una didascalia;
2. un'etichetta con `\label`;
3. i numeri di riga.

Nel paragrafo immediatamente successivo fate riferimento al listato tramite il comando `\ref`, senza scrivere esplicitamente il suo numero.

6.7 Inserire il contenuto di un file esterno

Quando il codice diventa più lungo, copiarlo direttamente nel sorgente LaTeX presenta uno svantaggio importante: si finisce per avere due copie dello stesso programma, una nel file sorgente originale ed una nel documento.

Il pacchetto `listings` permette invece di leggere direttamente un file esterno mediante il comando

```
\lstinputlisting{nomefile.py}
```

Ad esempio,

```
\lstinputlisting[
  language=Python,
  caption={Metodo di bisezione},
  label={lst:bisezione}
]{bisezione.py}
```

inserisce nel documento il contenuto del file `bisezione.py`.

In questo modo il documento mostra sempre la versione corrente del programma: se si modifica `bisezione.py`, alla compilazione successiva anche il listato nel PDF viene aggiornato.

Esercizio 21

Create un file separato chiamato

```
newton.py
```

contenente un'implementazione del metodo di Newton per risolvere

$$f(x) = 0.$$

La funzione potrà avere, ad esempio, la forma

```
newton(f, df, x0, toll)
```

dove `df` rappresenta la derivata di f.

Inserite poi il programma nel documento utilizzando `\lstinputlisting` invece di copiarne manualmente il contenuto.

Se lavorate sulla vostra macchina, il file `newton.py` deve trovarsi nella directory indicata dal comando `\lstinputlisting`.

Se lavorate su `https://overleaf.unipi.it`, aggiungete `newton.py` fra i file del progetto. Il comando `\lstinputlisting` potrà quindi leggerlo direttamente durante la compilazione.

6.8 Selezionare solo alcune righe

Non è sempre necessario mostrare un intero file. Le opzioni `firstline` e `lastline` permettono di selezionare un intervallo di righe:

```
\lstinputlisting[
  language=Python,
  firstline=5,
  lastline=12
]{programma.py}
```

Questa possibilità è utile quando un programma è lungo ma si vuole discutere soltanto una sua parte.

Esercizio 22

Partendo da uno dei programmi scritti negli esercizi precedenti, inserite nel documento:

1. una prima volta il file completo;
2. una seconda volta soltanto le righe che costituiscono il corpo dell'algoritmo.

Utilizzate le opzioni `firstline` e `lastline` per realizzare il secondo listato.

6.9 Esercizio riassuntivo

Esercizio 23

Realizzate una breve relazione su uno dei seguenti algoritmi:

- algoritmo di Euclide;
- metodo di bisezione;
- metodo di Newton;
- ordinamento per selezione;

- ordinamento per inserzione;
- ricerca binaria.

La relazione dovrà contenere:

1. una descrizione del problema da risolvere;
2. una descrizione a parole dell'algoritmo;
3. almeno una formula matematica, se pertinente;
4. un'implementazione dell'algoritmo in un linguaggio a scelta;
5. il codice inserito mediante `listings`;
6. una didascalia ed un'etichetta per il listato;
7. almeno un riferimento al listato nel testo;
8. almeno un esempio di esecuzione dell'algoritmo;
9. un breve commento sul risultato ottenuto.

Il codice dovrà essere mantenuto in un file separato e incluso nel documento con `\lstinputlisting`.

Chi lavora sulla propria macchina dovrà assicurarsi che il file sorgente del programma sia nella posizione corretta rispetto al file `.tex`; chi lavora su `https://overleaf.unipi.it` dovrà caricare il file sorgente all'interno dello stesso progetto.

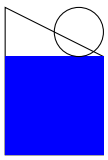
---

7 Disegnare in LaTeX

Abbiamo già visto nelle sezioni precedenti come generare dei grafici direttamente in LaTeX. Questo è solo un piccolo esempio di uso di un pacchetto molto potente, chiamato TikZ, che permette di disegnare all'interno del nostro documento. In ogni momento, è possibile iniziare un disegno tramite l'ambiente `tikzpicture`, e produrre elementi grafici con appositi comandi. Per usare il pacchetto, occorre inserire il comando `\usepackage{tikz}`.

Si consideri il seguente esempio, utilizzando i comandi `\draw` e `\fill` che permettono di disegnare linee e riempire delle regioni.

```
\begin{tikzpicture}
  \draw (0,0) -- (2,0) --
     (2,2) -- (0,3) -- cycle;
  \fill[blue] (0,0) rectangle (2,2);
  \draw (1.5,2.5) circle (.5);
\end{tikzpicture}
```



È spesso utile anche il comando `\filldraw`, che permette di disegnare contemporaneamente il contorno ed il riempimento di una regione. Tutti questi comandi possono accettare molte opzioni specificate fra parentesi quadre, ad esempio per produrre una freccia:

```
\begin{tikzpicture}
  \draw[->] (0,0) -- (2,0);
\end{tikzpicture}
```


E se volessimo disegnare una freccia curva, ad esempio per denotare una mappa fra due insiemi? Possiamo creare una linea di questo tipo usando i cosiddetti punti di controllo. L'idea è di connettere due punti P_0 e P_1 definendo una curva

$$P(t) := (P_x(t), P_y(t)), \qquad \deg P_x, P_y \leq 3,$$

tali che $P(0) = P_1$, $P(1) = P_1$, e siano assegnate le derivate $P'(0)$ e $P'(1)$. Lo spazio dei polinomi di grado al più 3 ha dimensione 4, e le condizioni citate sono 4 condizioni lineari affini, e linearmente indipendenti. Dunque, la curva polinomiale che soddisfa questi requisiti è unica[1].

Per specificare le derivate si utilizzano dei punti di controllo, $\widehat{P}_0$ e $\widehat{P}_1$, tali che la derivata in 0 sia $\widehat{P}_0 - P_0$, ed in 1 sia $P_1 - \widehat{P}_1$. Ad esempio:

```
\begin{tikzpicture}
  \draw[->] (0,0) ..
    controls (0.5,0.5) and (1.5,0.5)
    .. (2,0);
\end{tikzpicture}
```



Nella figura sopra sono riportate anche le tangenti alla curva nei punti P_0, P_1 per facilitare la comprensione. Per aggiungere del testo possiamo creare dei nodi, che ci permettono di "attaccare del testo" ad alcuni punti. Ad esempio, potremmo modificare l'esempio precedente per indicare P_0 e P_1:

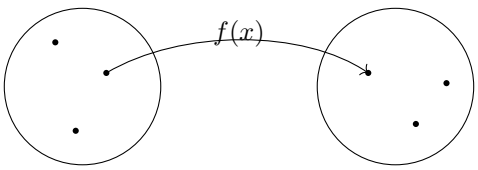
```
\begin{tikzpicture}
  \draw[->] (0,0) ..
    controls (0.5,0.5) and (1.5,0.5)
    .. (2,0);
  \node at (0,-0.25) {$P_0$};
  \node at (2,-0.25) {$P_1$};
\end{tikzpicture}
```



A questo punto abbiamo tutti gli strumenti necessari per realizzare il seguente disegno.

Esercizio 24

Si realizzi il seguente disegno utilizzando TikZ:



[1] a meno di casi degeneri.

Ovviamente non è necessario che sia esattamente uguale, ma solo che siano presenti gli elementi corretti.

Una caratteristica utile di TikZ è che può utilizzare le capacità del TEX come linguaggio di programmazione. Ad esempio, supponisamo di voler disegnare 20 cerchi di raggio piccolo, uno di fianco all'altro. Potremmo ripetere lo stesso codice più volte, oppure utilizzare il seguente codice:

```
\begin{tikzpicture}
  \foreach \i in {0, 0.1, ..., 1.9} {
    \filldraw (\i, 0) circle (0.025);
  }
\end{tikzpicture}
```


Si utilizzi un ciclo per disegnare delle spirali come negli esempi seguenti; a questo scopo, potrebbe essere utile conoscere il comando `\pgfmathparse` che permette di valutare il risultato di un'operazione aritmetica, e `\pgfmathresult` che permette di ritornare il risultato dell'ultima operazione fatta. Si può combinare queste istruzioni con `\let`, che permette di definire una variabile uguale al risultato di un'operazione, ad esempio:

```
\pgfmathparse(1 + 3)
\let\w\pgfmathresult % Ora il comando \w vale 4
```

Ecco alcune spirali da provare a riprodurre:



7.1 Disegnare grafi con `tikz-network`

Un'applicazione particolarmente comune della grafica in matematica e informatica consiste nel disegnare *grafi* e *reti*. Naturalmente è possibile costruire un grafo direttamente con TikZ, utilizzando nodi, cerchi e segmenti, ma quando il numero di vertici e archi cresce può essere comodo utilizzare un pacchetto più specializzato.

Il pacchetto `tikz-network` è costruito sopra TikZ e mette a disposizione alcuni comandi specifici per la rappresentazione di reti. Per utilizzarlo occorre aggiungere nel preambolo

```
\usepackage{tikz-network}
```

L'idea fondamentale è molto semplice: un grafo viene descritto creando prima i suoi *vertici*, mediante il comando `\Vertex`, e poi gli *archi*, mediante il comando `\Edge`.

Consideriamo, ad esempio, il grafo con tre vertici

$$V = \{A, B, C\}$$

e tre archi

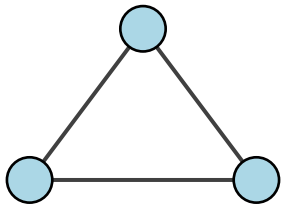
$$E = \{\{A, B\}, \{B, C\}, \{C, A\}\}.$$

Possiamo rappresentarlo con il codice

```
\begin{tikzpicture}
  \Vertex[x=0,y=0]{A}
  \Vertex[x=3,y=0]{B}
  \Vertex[x=1.5,y=2]{C}

  \Edge(A)(B)
  \Edge(B)(C)
  \Edge(C)(A)
\end{tikzpicture}
```

che produce un triangolo i cui vertici sono etichettati con A, B e C.



Nel comando

```
\Vertex[x=0,y=0]{A}
```

l'argomento `A` identifica il vertice, mentre le opzioni `x` e `y` ne specificano la posizione. Le coordinate vengono interpretate all'interno dell'ambiente `tikzpicture`.

Una volta definiti i vertici, il comando

```
\Edge(A)(B)
```

crea un arco fra i vertici `A` e `B`.

Esercizio 25

Utilizzando `tikz-network`, disegnate il grafo ciclo C_4, formato da quattro vertici

$$V = \{A, B, C, D\}$$

e dagli archi

$$\{A, B\}, \qquad \{B, C\}, \qquad \{C, D\}, \qquad \{D, A\}.$$

Disponete i quattro vertici ai vertici di un quadrato.

Aggiungete poi l'arco $\{A, C\}$ e osservate come cambia il grafo.

Etichette dei vertici

Il nome utilizzato internamente per identificare un vertice non deve necessariamente coincidere con il testo visualizzato nel disegno.

Ad esempio:

```
\Vertex[x=0,y=0,label={$v_1$}]{A}
\Vertex[x=3,y=0,label={$v_2$}]{B}

\Edge(A)(B)
```

utilizza i nomi `A` e `B` per fare riferimento ai vertici nel codice, ma visualizza nel disegno le etichette v_1 e v_2.

Questo può essere particolarmente utile quando le etichette contengono formule matematiche:

```
\begin{tikzpicture}
  \Vertex[x=0,y=0,label={$x_1$}]{A}
  \Vertex[x=3,y=0,label={$x_2$}]{B}
  \Vertex[x=1.5,y=2,label={$x_3$}]{C}

  \Edge(A)(B)
  \Edge(B)(C)
  \Edge(C)(A)
\end{tikzpicture}
```

In questo modo possiamo continuare a scrivere semplicemente `\Edge(A)(B)`, senza dover utilizzare le formule matematiche come identificatori dei vertici.

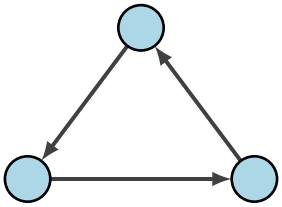
Grafi orientati

Un grafo può essere *orientato*: in questo caso un arco possiede una direzione e viene normalmente rappresentato mediante una freccia.

Con `tikz-network` possiamo ottenere un arco orientato mediante l'opzione `Direct`:

```
\begin{tikzpicture}
  \Vertex[x=0,y=0]{A}
  \Vertex[x=3,y=0]{B}
  \Vertex[x=1.5,y=2]{C}

  \Edge[Direct](A)(B)
  \Edge[Direct](B)(C)
  \Edge[Direct](C)(A)
\end{tikzpicture}
```



Si osservi che l'ordine dei due vertici diventa ora importante:

```
\Edge[Direct](A)(B)
```

rappresenta un arco orientato da A verso B, mentre

```
\Edge[Direct](B)(A)
```

rappresenta l'arco nella direzione opposta.

Esercizio 26

Disegnate il grafo orientato

$$A \longrightarrow B \longrightarrow C \longrightarrow D$$

aggiungendo inoltre gli archi

$$A \longrightarrow C, \qquad D \longrightarrow A.$$

Modificate successivamente la direzione di almeno due archi e osservate quali comandi `\Edge` devono essere cambiati.

Archi pesati

In molte applicazioni ad ogni arco viene associato un numero. Ad esempio, in una rete stradale questo numero potrebbe rappresentare una distanza, mentre in un problema di ottimizzazione potrebbe rappresentare un costo.

Possiamo aggiungere un'etichetta ad un arco mediante l'opzione `label`:

```
\begin{tikzpicture}
  \Vertex[x=0,y=0]{A}
  \Vertex[x=4,y=0]{B}
  \Vertex[x=2,y=2]{C}

  \Edge[label={$5$}](A)(B)
  \Edge[label={$2$}](A)(C)
  \Edge[label={$3$}](C)(B)
\end{tikzpicture}
```

Otteniamo in questo modo un grafo pesato. Formalmente possiamo interpretare le etichette mediante una funzione

$$w : E \longrightarrow \mathbb{R},$$

dove $w(e)$ associa un peso ad ogni arco e.

Naturalmente le etichette non devono necessariamente essere numeriche. Possiamo inserirvi anche simboli o formule matematiche.

Esercizio 27

Disegnate il seguente grafo pesato:

$$V = \{A, B, C, D\},$$

con pesi

$$w(A,B) = 4, \qquad w(A,C) = 2, \qquad w(B,C) = 1, \qquad w(B,D) = 5, \qquad w(C,D) = 3.$$

Provate quindi a determinare, osservando il disegno, un cammino di peso minimo che connetta A a D.

Archi curvi

Può capitare che due archi si sovrappongano oppure che si vogliano rappresentare due collegamenti distinti fra gli stessi vertici. In questi casi può essere utile disegnare un arco curvo.

Ad esempio:

```
\begin{tikzpicture}
  \Vertex[x=0,y=0]{A}
  \Vertex[x=4,y=0]{B}

  \Edge[bend=30](A)(B)
  \Edge[bend=-30](A)(B)
\end{tikzpicture}
```

produce due archi curvi fra gli stessi vertici. Il valore di `bend` controlla la curvatura e il suo segno determina il lato verso il quale l'arco viene piegato.

Questa possibilità è utile, ad esempio, nella rappresentazione di grafi orientati nei quali siano presenti contemporaneamente gli archi

$$A \longrightarrow B \qquad \text{e} \qquad B \longrightarrow A.$$

Modificare lo stile della rete

Come TikZ, anche `tikz-network` permette di separare la struttura logica del disegno dalla sua rappresentazione grafica.

Possiamo modificare globalmente l'aspetto dei vertici tramite il comando `\SetVertexStyle`. Ad esempio:

```
\SetVertexStyle[
  FillColor=blue!20,
  LineColor=blue,
  LineWidth=1pt
]
```

Analogamente, possiamo modificare lo stile degli archi mediante `\SetEdgeStyle`:

```
\SetEdgeStyle[
  Color=black!70,
  LineWidth=1pt
]
```

Questi comandi sono particolarmente utili quando dobbiamo disegnare molti grafi nello stesso documento: invece di specificare lo stile di ogni singolo vertice o arco, possiamo scegliere una configurazione uniforme una sola volta.

Ad esempio:

```
\SetVertexStyle[
  FillColor=blue!20,
  LineColor=blue
]

\SetEdgeStyle[
  Color=black!70,
  LineWidth=1pt
]

\begin{tikzpicture}
  \Vertex[x=0,y=0]{A}
  \Vertex[x=3,y=0]{B}
  \Vertex[x=3,y=2]{C}
  \Vertex[x=0,y=2]{D}

  \Edge(A)(B)
  \Edge(B)(C)
  \Edge(C)(D)
  \Edge(D)(A)
  \Edge(A)(C)
\end{tikzpicture}
```

Come nel resto di TikZ, è generalmente preferibile definire uno stile coerente per tutto il documento invece di modificare separatamente ogni elemento.

Combinare `tikz-network` e TikZ

Poiché `tikz-network` è basato su TikZ, possiamo utilizzare nello stesso ambiente anche i normali comandi TikZ.

Ad esempio:

```
\begin{tikzpicture}
  \Vertex[x=0,y=0]{A}
  \Vertex[x=3,y=0]{B}
  \Vertex[x=1.5,y=2]{C}

  \Edge(A)(B)
  \Edge(B)(C)
  \Edge(C)(A)

  \node at (1.5,-0.7) {Un grafo triangolare};
\end{tikzpicture}
```

Il comando `\node` non appartiene a `tikz-network`, ma è un normale comando TikZ. Questa compatibilità permette di aggiungere al disegno testo, formule, frecce, regioni colorate e tutti gli altri elementi grafici che abbiamo visto in precedenza.

Esercizio 28

Realizzate con `tikz-network` il grafo di una piccola rete di collegamenti fra cinque città:

$$V = \{Pisa, Lucca, Livorno, Firenze, Siena\}.$$

Scegliete liberamente la posizione dei vertici e aggiungete almeno sette archi. Associate ad ogni arco una distanza numerica.

Il disegno dovrà soddisfare i seguenti requisiti:

1. ogni città deve corrispondere ad un vertice;
2. gli archi devono avere un'etichetta che rappresenti la distanza;
3. almeno un arco deve essere curvo;
4. l'aspetto dei vertici deve essere modificato rispetto allo stile predefinito;
5. deve essere presente almeno un elemento aggiunto tramite un normale comando TikZ.

Provate infine a interpretare il grafo come un problema di cammino minimo e individuate un possibile percorso fra Pisa e Siena.

Reti di dimensioni maggiori

Quando una rete contiene molti vertici, scrivere manualmente un comando `\Vertex` e un comando `\Edge` per ogni elemento può diventare scomodo. Il pacchetto `tikz-network` mette a disposizione anche comandi per leggere vertici e archi da file esterni.

In questo modo la descrizione dei dati può essere separata dal codice LaTeX che produce il disegno. Questa possibilità è particolarmente utile quando la rete viene generata da un altro programma, ad esempio Python, MATLAB o Julia.

Per reti piccole, tuttavia, l'uso esplicito dei comandi `\Vertex` e `\Edge` rimane spesso la soluzione più semplice e permette di comprendere chiaramente la struttura del grafo.

7.2 Diagrammi commutativi con `tikz-cd`

In algebra, geometria, topologia e teoria delle categorie è molto comune utilizzare dei *diagrammi commutativi*. Un diagramma permette di rappresentare oggetti matematici e mappe fra di essi mettendo in evidenza le relazioni fra diverse composizioni di funzioni.

Naturalmente sarebbe possibile disegnare un diagramma utilizzando direttamente TikZ, creando manualmente nodi e frecce. Per questo tipo di disegni, tuttavia, è spesso più comodo utilizzare il pacchetto `tikz-cd`, che è costruito sopra TikZ e fornisce una sintassi appositamente pensata per i diagrammi commutativi.

Per utilizzarlo è sufficiente aggiungere nel preambolo

```
\usepackage{tikz-cd}
```

L'ambiente principale messo a disposizione dal pacchetto è `tikzcd`. Gli oggetti vengono disposti come gli elementi di una matrice: il simbolo `&` separa le colonne, mentre `\\` passa alla riga successiva.

Ad esempio, il codice

```
\begin{tikzcd}
A & B \\
C & D
\end{tikzcd}
```

dispone i quattro oggetti secondo lo schema

$$A \qquad B$$

$$C \qquad D$$

Naturalmente, per ottenere un diagramma interessante dobbiamo aggiungere delle frecce.

Inserire le frecce

Le frecce vengono create con il comando `\arrow`. La direzione viene specificata utilizzando una o più delle lettere

`r`, `l`, `u`, `d`,

che significano rispettivamente *right*, *left*, *up* e *down*.

Ad esempio:

```
\begin{tikzcd}
A \arrow[r] & B
\end{tikzcd}
```

produce

$$A \longrightarrow B.$$

Possiamo assegnare un nome alla freccia inserendo una stringa fra virgolette:

```
\begin{tikzcd}
A \arrow[r, "f"] & B
\end{tikzcd}
```

ottenendo

$$A \xrightarrow{f} B.$$

La sintassi è particolarmente comoda perché non dobbiamo specificare le coordinate degli oggetti: `tikz-cd` calcola automaticamente la posizione dei nodi e la lunghezza delle frecce.

Esercizio 29

Realizzate con `tikz-cd` il diagramma

$$A \xrightarrow{f} B \xrightarrow{g} C.$$

Modificate quindi i nomi degli oggetti e delle mappe e provate ad aggiungere un quarto oggetto D ed una mappa $h : C \to D$.

Un primo diagramma commutativo

Consideriamo quattro insiemi (o, più in generale, quattro oggetti) A, B, C, D e delle mappe

$$f : A \to B, \qquad g : A \to C, \qquad h : B \to D, \qquad k : C \to D.$$

Possiamo rappresentare queste mappe mediante il diagramma

$$\begin{array}{ccc} A & \xrightarrow{f} & B \\ {\scriptstyle g}\downarrow & & \downarrow{\scriptstyle h} \\ C & \xrightarrow[k]{} & D. \end{array}$$

Il codice che lo genera è:

```
\begin{tikzcd}
A \arrow[r, "f"] \arrow[d, "g"'] &
B \arrow[d, "h"] \\
C \arrow[r, "k"'] &
D
\end{tikzcd}
```

Il simbolo ' dopo un'etichetta permette di spostarla sull'altro lato della freccia. Ad esempio,

```
\arrow[d, "g"']
```

e

```
\arrow[d, "g"]
```

disegnano la stessa freccia, ma dispongono l'etichetta g sui due lati opposti.

Dire che il diagramma precedente *commuta* significa affermare che i due possibili percorsi da A a D producono la stessa mappa, cioè

$$h \circ f = k \circ g.$$

Il diagramma permette quindi di rappresentare graficamente un'uguaglianza fra composizioni di funzioni.

Esercizio 30

Disegnate il diagramma commutativo associato alle mappe

$$f : X \to Y, \qquad g : X \to Z, \qquad h : Y \to W, \qquad k : Z \to W,$$

e scrivete esplicitamente l'uguaglianza fra composizioni che esprime la commutatività del diagramma.

Frecce diagonali

Le direzioni possono essere combinate. Ad esempio,

```
\arrow[dr]
```

indica una freccia diretta verso il basso e verso destra, mentre

```
\arrow[ul]
```

indica una freccia diretta verso l'alto e verso sinistra.

Consideriamo, ad esempio, due mappe $f : A \to B$ e $g : B \to C$ e la loro composizione $g \circ f : A \to C$. Possiamo rappresentare questa situazione con il diagramma triangolare

$$\begin{array}{ccc} A & \xrightarrow{f} & B \\ & \searrow^{g\circ f} & \downarrow{\scriptstyle g} \\ & & C. \end{array}$$

Il codice corrispondente è:

```
\begin{tikzcd}
A \arrow[r, "f"]
  \arrow[dr, "g\circ f"'] &
B \arrow[d, "g"] \\
& C
\end{tikzcd}
```

La commutatività di questo diagramma esprime precisamente il fatto che la freccia diagonale coincide con la composizione delle altre due.

Esercizio 31

Rappresentate mediante un diagramma triangolare tre mappe

$$f : X \to Y, \qquad g : Y \to Z, \qquad h : X \to Z$$

soggette alla relazione

$$h = g \circ f.$$

Provate quindi a spostare le etichette delle frecce sui lati opposti utilizzando il simbolo `'`.

Monomorfismi ed epimorfismi

In molti diagrammi si utilizzano tipi di freccia diversi per sottolineare particolari proprietà delle mappe.

Ad esempio, l'opzione `hook` produce la freccia normalmente utilizzata per rappresentare un'inclusione o, più in generale, un monomorfismo:

```
\begin{tikzcd}
A \arrow[r, hook, "i"] & B
\end{tikzcd}
```

che produce

$$A \xhookrightarrow{i} B.$$

L'opzione `two heads` produce invece una freccia a doppia punta, spesso utilizzata per indicare una mappa suriettiva o un epimorfismo:

```
\begin{tikzcd}
A \arrow[r, two heads, "p"] & B
\end{tikzcd}
```

ottenendo

$$A \xrightarrow{p} B.$$

Possiamo naturalmente utilizzare questi stili all'interno di diagrammi più complessi. Ad esempio:

$$A \xhookrightarrow{i} B \xrightarrow{p} C.$$

Una freccia tratteggiata può invece essere ottenuta mediante `dashed`:

```
\arrow[r, dashed, "f"]
```

Questo tipo di freccia viene spesso utilizzato per indicare una mappa la cui esistenza deve essere dimostrata oppure che viene ottenuta mediante una proprietà universale.

Esercizio 32

Riproducete il diagramma

$$A \xhookrightarrow{i} B \xrightarrow{p} C$$

e modificate le etichette delle tre varietà.

Aggiungete quindi un oggetto D ed una freccia tratteggiata da D a B.

Isomorfismi

Per indicare che una mappa è un isomorfismo è comune utilizzare un'etichetta come $\sim$ sopra la freccia:

$$A \xrightarrow[f]{\sim} B.$$

Ad esempio:

```
\begin{tikzcd}
A \arrow[r, "\sim", "f"'] & B
\end{tikzcd}
```

Se non interessa specificare il nome della mappa, possiamo semplicemente scrivere

```
\begin{tikzcd}
A \arrow[r, "\sim"] & B
\end{tikzcd}
```

Le etichette delle frecce possono quindi essere normali stringhe di testo, simboli matematici oppure espressioni più elaborate.

Sequenze

L'ambiente `tikzcd` non è utile soltanto per i diagrammi bidimensionali. Può essere utilizzato anche per rappresentare sequenze di mappe.

Ad esempio, una successione

$$0 \longrightarrow A \xrightarrow{f} B \xrightarrow{g} C \longrightarrow 0$$

può essere scritta come

```
\begin{tikzcd}
0 \arrow[r] &
A \arrow[r, "f"] &
B \arrow[r, "g"] &
C \arrow[r] &
0
\end{tikzcd}
```

ottenendo

$$0 \longrightarrow A \xrightarrow{f} B \xrightarrow{g} C \longrightarrow 0.$$

In algebra una sequenza di questo tipo potrebbe, ad esempio, rappresentare una successione esatta corta. La proprietà di esattezza non è naturalmente determinata dal disegno: il diagramma rappresenta le mappe, mentre il testo deve specificare quali proprietà matematiche esse soddisfano.

Esercizio 33

Rappresentate con `tikz-cd` la successione

$$0 \longrightarrow U \xrightarrow{i} V \xrightarrow{p} W \longrightarrow 0.$$

Utilizzate una freccia con `hook` per i e una freccia con `two heads` per p.

Diagrammi con più righe

Uno dei principali vantaggi di `tikz-cd` è che diagrammi anche abbastanza complessi possono essere scritti utilizzando una struttura simile a quella di una matrice.

Ad esempio:

$$\begin{array}{ccccc} A & \xrightarrow{f} & B & \xrightarrow{g} & C \\ {\scriptstyle\alpha}\downarrow & & \downarrow{\scriptstyle\beta} & & \downarrow{\scriptstyle\gamma} \\ A' & \xrightarrow[f']{} & B' & \xrightarrow[g']{} & C' \end{array}$$

si ottiene con

```
\begin{tikzcd}
A \arrow[r, "f"] \arrow[d, "\alpha"'] &
B \arrow[r, "g"] \arrow[d, "\beta"] &
C \arrow[d, "\gamma"] \\
A' \arrow[r, "f'"'] &
B' \arrow[r, "g'"'] &
C'
\end{tikzcd}
```

La commutatività del quadrato di sinistra significa

$$\beta \circ f = f' \circ \alpha,$$

mentre quella del quadrato di destra significa

$$\gamma \circ g = g' \circ \beta.$$

Quindi un unico diagramma può rappresentare contemporaneamente diverse relazioni fra composizioni di mappe.

Esercizio 34

Riproducete il diagramma precedente e scrivete sotto di esso tutte le uguaglianze fra composizioni determinate dalla sua commutatività.

Aggiungete quindi una quarta colonna contenente gli oggetti D e D', una mappa $h : C \to D$, una mappa $h' : C' \to D'$ e una mappa verticale $\delta : D \to D'$.

Frecce curve

Come per TikZ, anche in `tikz-cd` è possibile curvare una freccia. Questa possibilità è utile quando esistono più mappe fra gli stessi oggetti.

Ad esempio:

```
\begin{tikzcd}
A
  \arrow[r, bend left=30, "f"]
  \arrow[r, bend right=30, "g"'] &
B
\end{tikzcd}
```

produce

$$A \underset{g}{\overset{f}{\rightrightarrows}} B.$$

Il numero specificato dopo `bend left` o `bend right` controlla l'intensità della curvatura.

Questa rappresentazione è particolarmente comune in teoria delle categorie, dove può essere necessario confrontare diverse mappe aventi lo stesso dominio e lo stesso codominio.

Esercizio 35

Disegnate due oggetti X e Y collegati da tre mappe

$$f, g, h : X \to Y.$$

Utilizzate una freccia rettilinea per f, una curva verso l'alto per g e una curva verso il basso per h.

Due frecce in direzioni opposte

Supponiamo di avere due mappe

$$f : A \to B, \qquad g : B \to A.$$

Una possibilità è rappresentarle mediante due frecce leggermente spostate:

```
\begin{tikzcd}
A
  \arrow[r, shift left, "f"]
&
B
  \arrow[l, shift left, "g"]
\end{tikzcd}
```

ottenendo

$$A \underset{g}{\overset{f}{\rightleftarrows}} B$$

L'opzione `shift left` impedisce alle due frecce di sovrapporsi.

Questo tipo di diagramma compare frequentemente, ad esempio, quando due oggetti sono collegati da mappe in entrambe le direzioni.

Diagrammi di inclusioni

Il pacchetto risulta utile anche per rappresentare inclusioni fra insiemi o sottospazi.

Ad esempio, se

$$A \subseteq B, \qquad A \subseteq C, \qquad B \subseteq D, \qquad C \subseteq D,$$

possiamo scrivere

$$\begin{array}{ccc} A & \hookrightarrow & B \\ \downarrow & & \downarrow \\ C & \hookrightarrow & D. \end{array}$$

Il codice corrispondente è:

```
\begin{tikzcd}
A \arrow[r, hook] \arrow[d, hook] &
B \arrow[d, hook] \\
C \arrow[r, hook] &
D
\end{tikzcd}
```

Esercizio 36

Siano U e V due sottospazi vettoriali di uno spazio vettoriale W. Costruite un diagramma che rappresenti le inclusioni

$$U \cap V \subseteq U, \qquad U \cap V \subseteq V, \qquad U \subseteq U + V, \qquad V \subseteq U + V.$$

Utilizzate esclusivamente frecce di tipo `hook`.

Naturalezza

Un'altra situazione in cui i diagrammi commutativi sono particolarmente utili è quella delle trasformazioni naturali.

Supponiamo di avere due applicazioni F e G e delle mappe

$$\eta_X : F(X) \to G(X), \qquad \eta_Y : F(Y) \to G(Y).$$

La condizione di naturalezza associata ad una mappa $f : X \to Y$ può essere rappresentata mediante il quadrato

$$\begin{array}{ccc} F(X) & \xrightarrow{F(f)} & F(Y) \\ {\scriptstyle \eta_X}\downarrow & & \downarrow{\scriptstyle \eta_Y} \\ G(X) & \xrightarrow[G(f)]{} & G(Y). \end{array}$$

La commutatività del diagramma equivale all'identità

$$\eta_Y \circ F(f) = G(f) \circ \eta_X.$$

Questo è un esempio importante del vantaggio dei diagrammi: una relazione algebrica che richiede una formula relativamente lunga può essere visualizzata immediatamente attraverso un quadrato commutativo.

Esercizio 37

Riproducete il diagramma di naturalezza precedente.

Scambiate successivamente le righe contenenti F e G e modificate opportunamente la direzione delle frecce verticali.

Modificare le dimensioni del diagramma

Per diagrammi molto densi può essere utile aumentare la distanza fra righe e colonne. L'ambiente `tikzcd` accetta a questo scopo le opzioni `row sep` e `column sep`.

Ad esempio:

```
\begin{tikzcd}[
  row sep=large,
  column sep=large
]
A \arrow[r, "f"] \arrow[d, "g"'] &
B \arrow[d, "h"] \\
C \arrow[r, "k"'] &
D
\end{tikzcd}
```

Allo stesso modo, per ottenere un diagramma più compatto si possono utilizzare valori come `small`.

Queste opzioni sono preferibili allo spostamento manuale dei singoli oggetti, perché mantengono uniforme la struttura del diagramma.

Esercizio 38

Costruite il diagramma

$$\begin{array}{ccccc} A & \longrightarrow & B & \longrightarrow & C \\ \downarrow & & \downarrow & & \downarrow \\ D & \longrightarrow & E & \longrightarrow & F \end{array}$$

e provate successivamente le opzioni

```
row sep=small,
column sep=small
```

e

```
row sep=huge,
column sep=huge
```

Confrontate i risultati e scegliete una spaziatura che renda il diagramma facilmente leggibile.

Combinare `tikz-cd` e TikZ

Il pacchetto `tikz-cd` è basato su TikZ. Di conseguenza, molte delle possibilità di personalizzazione offerte da TikZ rimangono disponibili.

Nella maggior parte dei casi, tuttavia, è conveniente lasciare che `tikz-cd` si occupi automaticamente dell'impaginazione e intervenire sullo stile soltanto quando è realmente necessario.

In particolare, un diagramma commutativo dovrebbe prima di tutto rendere chiara la struttura matematica: la disposizione degli oggetti e delle frecce è normalmente più importante di una personalizzazione grafica molto elaborata.

Esercizio 39

Realizzate un diagramma commutativo che contenga almeno:

1. sei oggetti;
2. due righe;
3. tre frecce verticali;
4. almeno quattro frecce orizzontali;
5. un monomorfismo, rappresentato con `hook`;
6. un epimorfismo, rappresentato con `two heads`;
7. una freccia tratteggiata;
8. una freccia diagonale.

Assegnate un nome a tutte le mappe e scrivete nel testo almeno due delle uguaglianze fra composizioni che derivano dalla commutatività del diagramma.

Esercizio riassuntivo

Esercizio 40

Considerate il seguente diagramma:

$$\begin{array}{ccccc} A & \xhookrightarrow{i} & B & \xrightarrow{p} & C \\ \downarrow{\scriptstyle \alpha} & & \downarrow{\scriptstyle \beta} & & \downarrow{\scriptstyle \gamma} \\ A' & \xhookrightarrow[i']{} & B' & \xrightarrow[p']{} & C' \end{array}$$

Riproducetelo utilizzando `tikz-cd` e svolgete le seguenti modifiche:

1. cambiate i nomi degli oggetti;
2. aggiungete un quarto oggetto a ciascuna riga;
3. aggiungete una nuova freccia verticale;

4. aggiungete una freccia diagonale tratteggiata;

5. modificate la distanza fra righe e colonne;

6. scrivete esplicitamente le equazioni che esprimono la commutatività dei quadrati presenti nel diagramma.

Infine spiegate brevemente perché, per un diagramma di questo tipo, `tikz-cd` risulta più comodo dell'uso diretto di coordinate e nodi TikZ.

---

8 Realizzare delle slide

La classe di documento più spesso utilizzata per scrivere appunti, relazioni o brevi articoli è `article`. Quando invece si vuole realizzare una presentazione, ad esempio per un seminario, una lezione o la discussione di una tesi, LaTeX mette a disposizione la classe `beamer`.

Un documento Beamer continua ad essere un normale documento LaTeX, e quindi molti dei comandi che abbiamo visto nelle sezioni precedenti possono essere utilizzati senza modifiche. In particolare possiamo continuare a scrivere formule, utilizzare gli ambienti matematici, definire teoremi, inserire immagini, tabelle e riferimenti bibliografici.

La differenza principale è che il documento non viene organizzato in pagine, ma in *slide*. In Beamer una slide viene normalmente descritta mediante l'ambiente `frame`:

```
\begin{frame}{Titolo della slide}
  Contenuto della slide.
\end{frame}
```

L'argomento fra parentesi graffe dopo `\begin{frame}` diventa il titolo della slide. Esso può anche essere omesso e specificato separatamente con il comando `\frametitle`.

Ad esempio, i due frammenti

```
\begin{frame}{Prima slide}
  Un po' di testo.
\end{frame}
```

e

```
\begin{frame}
  \frametitle{Prima slide}
  Un po' di testo.
\end{frame}
```

producono sostanzialmente lo stesso risultato.

8.1 Un primo documento Beamer

Un documento minimale può essere scritto nel modo seguente:

```
\documentclass{beamer}

\title{Titolo}
\author{Io}
\date{\today}

\begin{document}

  \begin{frame}
    \maketitle
  \end{frame}

  \begin{frame}{Prima slide}
    Contenuto della prima slide.
  \end{frame}

\end{document}
```

Come nei documenti di classe `article`, i comandi `\title`, `\author` e `\date` impostano alcune informazioni globali sulla presentazione. Il comando `\maketitle` utilizza queste informazioni per generare la slide iniziale.

Naturalmente una presentazione può contenere matematica esattamente come un normale documento LaTeX. Ad esempio:

```
\begin{frame}{Un'equazione}
  L'equazione di secondo grado
  \[
    ax^2 + bx + c = 0
  \]
  ha soluzioni
  \[
    x_{1,2} =
    \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.
  \]
\end{frame}
```

Esercizio 41

Si provi a compilare il documento minimale precedente e ad effettuare alcune modifiche, in modo da ottenere almeno tre slide. Una delle slide dovrà contenere un'equazione, una una lista puntata ed una del semplice testo.

Se si lavora sulla propria macchina, si salvi ad esempio il documento nel file `presentazione.tex` e lo si compili con

```
pdflatex presentazione.tex
```

eventualmente ripetendo il comando quando siano presenti riferimenti o altri elementi che richiedono più compilazioni.

Se invece si utilizza `https://overleaf.unipi.it`, si crei un nuovo progetto, si inserisca il codice nel file principale del progetto e si utilizzi il pulsante *Recompile*. Il risultato della compilazione viene mostrato direttamente nel visualizzatore PDF.

8.2 Organizzare una presentazione

Anche una presentazione può essere suddivisa in sezioni e sottosezioni. I comandi sono gli stessi già utilizzati nei normali documenti:

```
\section{Introduzione}
\subsection{Il problema}

\begin{frame}{Motivazione}
  ...
\end{frame}

\section{Metodo}

\begin{frame}{L'algoritmo}
  ...
\end{frame}
```

A seconda del tema scelto, i nomi delle sezioni possono comparire nell'intestazione o nel piè di pagina e possono essere utilizzati per produrre automaticamente un indice della presentazione.

Ad esempio possiamo creare una slide contenente l'indice con

```
\begin{frame}{Indice}
  \tableofcontents
\end{frame}
```

Come sempre in LaTeX, conviene descrivere la *struttura logica* del documento e lasciare che la classe e il tema si occupino dell'aspetto grafico.

8.3 Liste, blocchi e colonne

Le liste funzionano nello stesso modo visto per la classe `article`:

```
\begin{frame}{Una lista}
  \begin{itemize}
    \item Primo elemento;
    \item secondo elemento;
```

```
    \item terzo elemento.
  \end{itemize}
\end{frame}
```

Beamer fornisce inoltre alcuni ambienti particolarmente utili nelle presentazioni. Fra questi troviamo l'ambiente `block`:

```
\begin{frame}{Un risultato importante}
  \begin{block}{Osservazione}
    Il contenuto di questo blocco viene evidenziato
    rispetto al resto della slide.
  \end{block}
\end{frame}
```

Sono disponibili anche, fra gli altri, gli ambienti `alertblock` ed `exampleblock`, il cui aspetto dipende dal tema utilizzato.

Un'altra struttura molto utile è l'ambiente `columns`, che permette di dividere una slide in più colonne. Ad esempio:

```
\begin{frame}{Due colonne}
  \begin{columns}
    \begin{column}{0.48\textwidth}
      Testo nella colonna di sinistra.
    \end{column}

    \begin{column}{0.48\textwidth}
      Testo nella colonna di destra.
    \end{column}
  \end{columns}
\end{frame}
```

Questa struttura è particolarmente utile quando si vuole affiancare una figura ad una spiegazione o confrontare due formule o due procedure.

8.4 Frame contenenti codice

Occorre prestare una piccola attenzione quando una slide contiene del testo in modalità *verbatim*, ad esempio quando si vogliono mostrare comandi LaTeX o frammenti di codice sorgente. In questi casi Beamer deve elaborare il contenuto del frame in maniera particolare.

Si utilizza quindi l'opzione `fragile`:

```
\begin{frame}[fragile]{Un comando LaTeX}
  Il comando \verb|\section{Introduzione}|
  crea una nuova sezione.
\end{frame}
```

L'opzione `fragile` è necessaria, ad esempio, quando all'interno del frame si utilizzano il comando `\verb` oppure ambienti che contengono codice non interpretato normalmente da LaTeX.

Senza questa opzione, un frame contenente materiale verbatim può produrre errori di compilazione.

8.5 Transizioni e overlay

Una caratteristica importante di Beamer è la possibilità di far comparire progressivamente il contenuto di una slide. In questo modo una singola `frame` può generare diverse pagine del PDF, dette spesso *overlay*.

Il caso più semplice si ottiene con il comando `\pause`:

```
\begin{frame}{Un esempio}
  Prima osservazione.

  \pause

  Seconda osservazione.

  \pause

  Terza osservazione.
\end{frame}
```

Alla prima pressione del tasto per avanzare sarà visibile soltanto la prima osservazione; successivamente compariranno anche la seconda e la terza.

Per avere un controllo più preciso si possono usare le specifiche di overlay. Una specifica come

```
<2>
```

indica il secondo overlay, mentre

```
<2->
```

indica il secondo overlay e tutti quelli successivi.

Fra i comandi più utili troviamo:

- `\pause`, che separa in modo semplice due passaggi successivi;
- `\only<n>{...}`, che mostra il contenuto solamente negli overlay specificati;
- `\uncover<n>{...}`, che riserva lo spazio necessario al contenuto anche quando questo non è ancora visibile;
- `\visible<n>{...}`, simile a `\uncover`, ma con un comportamento leggermente diverso nella gestione del contenuto nascosto;
- `\alt<n>{a}{b}`, che mostra `a` negli overlay specificati e `b` negli altri.

Ad esempio:

```
\begin{frame}{Overlay}
  \only<1>{Questo appare solo la prima volta.}

  \only<2>{Questo appare solo la seconda volta.}

  \only<3->{Questo appare dalla terza volta in poi.}
\end{frame}
```

Le specifiche di overlay possono essere utilizzate direttamente anche negli elementi di una lista:

```
\begin{frame}{Una lista progressiva}
  \begin{itemize}
    \item<1-> Primo elemento;
    \item<2-> secondo elemento;
    \item<3-> terzo elemento.
  \end{itemize}
\end{frame}
```

Per il caso molto comune in cui tutti gli elementi devono apparire uno dopo l'altro, Beamer fornisce anche la forma abbreviata

```
\begin{itemize}[<+->]
  \item Primo elemento;
  \item secondo elemento;
  \item terzo elemento.
\end{itemize}
```

Esercizio 42

Sperimentate con i comandi precedenti in modo da ottenere una lista numerata di quattro elementi che, nei quattro overlay successivi, mostri rispettivamente:

- gli elementi 1 e 2;
- solamente l'elemento 2;
- gli elementi 3 e 4;
- solamente l'elemento 3.

Provate dapprima ad ottenere il risultato con `\only`. Successivamente provate a modificare l'esempio utilizzando `\uncover` e osservate la differenza nella posizione degli elementi fra un overlay e il successivo.

8.6 Temi per Beamer

La classe `beamer` separa il contenuto della presentazione dalla sua rappresentazione grafica. Il modo più semplice per modificarne l'aspetto consiste nello scegliere un *tema*.

Un tema si seleziona nel preambolo con

```
\usetheme{nome}
```

Ad esempio:

```
\documentclass{beamer}

\usetheme{Madrid}

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

Il tema decide molti aspetti della presentazione: colori, posizione del titolo, intestazioni, piè di pagina, rappresentazione dei blocchi, simboli delle liste e così via.

Beamer comprende già diversi temi. In maniera indipendente dal tema principale si può anche cambiare lo schema dei colori mediante

```
\usecolortheme{nome}
```

Per avere una panoramica dei temi e degli schemi di colore disponibili si può consultare `https://hartwork.org/beamer-theme-matrix/`.

I temi forniti insieme a Beamer non sono tuttavia gli unici disponibili: un tema può essere distribuito come un normale pacchetto LaTeX, ed è possibile sia installare temi realizzati da altri sia crearne di propri.

8.7 Codice nelle presentazioni Beamer

Il pacchetto `listings` può essere utilizzato anche all'interno di una presentazione Beamer. In questo caso occorre ricordare che un frame contenente un ambiente `lstlisting` deve essere dichiarato `fragile`:

```
\begin{frame}[fragile]{Un programma Python}

  \begin{lstlisting}[language=Python]
def quadrato(x):
    return x*x
  \end{lstlisting}

\end{frame}
```

L'opzione `fragile` è necessaria perché Beamer deve trattare in modo particolare il contenuto testuale dell'ambiente `lstlisting`.

Esercizio 43

Create una presentazione Beamer contenente tre frame:

1. una slide con il titolo della presentazione;
2. una slide che descriva brevemente un algoritmo;
3. una slide che ne mostri l'implementazione tramite `lstlisting`.

La slide contenente il codice dovrà utilizzare l'opzione `fragile`.

Provate infine a rimuovere `fragile` e osservate l'errore prodotto dal compilatore.

8.8 Il tema del Dipartimento di Matematica dell'Università di Pisa

Per le presentazioni relative al Dipartimento di Matematica dell'Università di Pisa è disponibile il tema Beamer `dmpisa`, reperibile all'indirizzo

`https://github.com/Cirdans-Home/dmunipi-slide/releases/tag/v1.0.2`.

A differenza dei temi inclusi direttamente nella normale distribuzione LaTeX, questo tema deve essere scaricato insieme ai file che ne definiscono lo stile e alle risorse grafiche utilizzate dalla presentazione.

Il modo più semplice per iniziare è scaricare dalla pagina delle release l'archivio del progetto. All'interno si trovano, fra gli altri, i file di stile necessari al tema, le immagini nella directory `assets` e un file `main.tex` contenente un esempio minimale.

Per utilizzare il tema è sufficiente scrivere nel preambolo

```
\documentclass{beamer}

\usetheme{dmpisa}
```

purché i file del tema siano disponibili nella directory del progetto o siano stati installati in una posizione nota alla propria distribuzione LaTeX.

Un esempio minimale, ispirato a quello distribuito insieme al tema, è:

```
\documentclass{beamer}

\usetheme{dmpisa}
\usefonttheme[onlymath]{serif}

\title{Titolo della presentazione}
```

```
\subtitle{Un eventuale sottotitolo}
\course{Laboratorio di Introduzione alla Matematica Computazionale}
\author{Nome Cognome}
\date{\today}

\begin{document}

\maketitle

\section{Introduzione}

\begin{frame}{Prima slide}
  Contenuto della presentazione.
\end{frame}

\backmatter

\end{document}
```

Si notino alcuni comandi che non appartengono al Beamer standard ma sono forniti dal tema. Ad esempio, `\course` permette di aggiungere un'informazione relativa al corso, mentre `\backmatter` genera una slide conclusiva. Il template mette inoltre a disposizione altri comandi e personalizzazioni; per una descrizione completa si consiglia di consultare il file di esempio fornito insieme al tema.

Il tema permette anche di utilizzare un'immagine nella pagina del titolo. Ad esempio:

```
\titlebackground{assets/background}
```

imposta l'immagine indicata come sfondo, mentre la variante

```
\titlebackground*{assets/background}
```

utilizza la versione grafica prevista dal tema con l'immagine collocata nella parte laterale della slide del titolo.

Utilizzo sulla propria macchina. Se si lavora in locale, si può scaricare e decomprimere l'archivio della release nella directory di lavoro, mantenendo la struttura delle sottodirectory. In particolare è importante non spostare o eliminare la directory `assets` se il sorgente fa riferimento alle immagini in essa contenute.

A questo punto ci si può spostare con il terminale nella directory del progetto e compilare, ad esempio, con

```
pdflatex main.tex
```

Se il compilatore segnala che non riesce a trovare `beamerthemedmpisa.sty` o un altro file del tema, la prima cosa da controllare è che l'intero contenuto dell'archivio sia stato estratto e che il file `main.tex` si trovi nella posizione prevista rispetto ai file `.sty` e alla directory `assets`.

Utilizzo su Overleaf. Su `https://overleaf.unipi.it` non è necessario installare il tema nel sistema. Il procedimento più semplice consiste nello scaricare l'archivio `.zip` della release dal repository e caricarlo come un nuovo progetto.

Una volta importato il progetto, si può partire dal file `main.tex` fornito nell'archivio, modificarne titolo, autore e contenuto e compilare con *Recompile*. I file di stile e la directory `assets` rimangono all'interno del progetto e vengono trovati automaticamente durante la compilazione.

Questa modalità è spesso più semplice dell'inserimento manuale dei singoli file, perché mantiene automaticamente la struttura delle directory richiesta dal tema.

Esercizio 44

Scaricare la release del tema `dmpisa` e compilare il file `main.tex` fornito insieme al progetto.

Modificare quindi la presentazione in modo da:

1. cambiare titolo, sottotitolo e autore;
2. creare almeno due sezioni;
3. inserire almeno tre frame;
4. inserire una formula matematica;
5. inserire una lista i cui elementi compaiano progressivamente;
6. modificare l'immagine utilizzata nella pagina del titolo, oppure rimuoverla;
7. aggiungere la slide conclusiva mediante `\backmatter`.

Chi lavora in locale compili il documento con `pdflatex`; chi utilizza `https://overleaf.unipi.it` importi direttamente l'archivio del progetto e utilizzi *Recompile*.

8.9 Personalizzare lo sfondo

Oltre a utilizzare un tema completo, Beamer permette di intervenire direttamente su singoli elementi grafici. Ad esempio, lo sfondo delle slide può essere personalizzato modificando il template `background canvas`:

```
\setbeamertemplate{background canvas}{
  % Codice LaTeX o TikZ
}
```

All'interno di questo comando si può inserire anche codice TikZ e quindi realizzare sfondi geometrici, inserire immagini o costruire altri elementi grafici.

In generale, tuttavia, è consigliabile partire da un tema coerente e personalizzarlo soltanto quando necessario: separare il contenuto dallo stile rende infatti più semplice modificare successivamente l'aspetto dell'intera presentazione.